An approach to semantic search on technical documentation based on machine learning algorithm for customer request resolution automation

Artem Kovalev
Institute of Computer Science and Technology
Peter the Great Saint-Petersburg Polytechnic University
Saint-Petersburg, Russian Federation
kov3000@ya.ru

Igor Nikiforov
Institute of Computer Science and Technology
Peter the Great Saint-Petersburg Polytechnic University
Saint-Petersburg, Russian Federation
i.nikiforov@ics2.ecd.spbstu.ru

Pavel Drobintsev
Institute of Computer Science and Technology
Peter the Great Saint-Petersburg Polytechnic University
Saint-Petersburg, Russian Federation
drob@ics2.ecd.spbstu.ru

***Abstract* — The software sustaining phase is one of the essential stages of the software development life cycle. At this stage, customers can contact support and software engineers to request a resolution of any problem they meet during software utilization including questions of how to operate with software, where to find the information about particular functions and other relevant questions on software product. The work is devoted to research in the field of software sustainment automation.**

The distinctive feature of the article is the suggested semantic documentation search approach with the Doc2Vec machine learning algorithm, which allows automating automate customer requests resolution. Proposed semantic search is performed on documentation files, like PDFs, Microsoft Office documents, wiki pages and other text files with relevant information about the product. Documentation files, including page numbers, that have the closest semantic similarity to the textual description of an unresolved customer request, help the developer resolve the incoming request more efficiently and in a shorter time.

The proposed approach is implemented in a software tool to automate the analysis of unresolved customer requests and provide recommendations to help in solving each of those requests. The results show the advantages of using the tool in the process of software product support.

***Keywords* — software sustaining, automation, doc2vec, machine learning, semantic search, documentation.**

I. Introduction

The software maintenance stage is one of the most important and complex stages of the development life cycle [1, 2]. In the process of software maintenance, the developers of the supplier company solve the problems that arise during the operation of the software product on the customer side. At this stage, a technical support engineer or a software developer receives and processes customer requests. A request is usually written in natural language and contains a description of the problem in the software product. For convenient management of such requests, issue tracking systems are usually used [3]. In this paper, we will consider the Jira issue tracking system [4].

In the process of solving the customer's problem, it is necessary to refer to the relevant documentation or knowledge base. The documentation can be presented in the form of local text files of various formats (pdf, doc, rtf, etc.), as well as contained on remote sites. A special case of the site with the documentation is the wiki system, which is considered in this paper.

The documentation can contain a significant amount of information and the search for the necessary pages can take a lot of time, which makes the process of studying the documentation laborious.

Document files are usually searched by keywords. However, this approach is not effective if the query becomes large and complex. The main disadvantage of keyword searches is the inability to define synonyms for words in a search query.

Algorithms of machine learning and neural networks [5], namely, the Doc2Vec algorithm [6], can help solve the problem of semantic search in documentation. With the software based on the Doc2Vec algorithm, it is possible to reduce the complexity and increase the efficiency of the maintenance process. Improving the quality of support helps build a long-term relationship between the software provider and the customer.

This work is aimed at the reduction of the complexity of search over the documentation via an automated approach based on the application of the Doc2Vec algorithm. To achieve this goal, it is necessary to develop a software tool that implements the proposed approach and show the effectiveness of the application of the proposed approach and its implementation in software on actual data.

The relevance of the study lies in the fact that in the process of developing and complicating software products, the volume of written source code increases, as well as the volume of documentation. This inevitably leads to an increase in the number of defects and shortcomings in the software, which is the reason why customers turn to the support service of the supplier company.

Our previous paper [7] showed the effectiveness of using the Doc2Vec algorithm to search for similar already resolved customer requests in the process of maintaining a software product. And the motivation of the current work is to show the application of the same approach to search for useful software documentation that can help in solving the request.

II. Related work

Semantic search is a well-established problem in the computer science research.

The use of ontologies is one of the approaches to semantic search. For example, Kassim et al. [8] proposed the Semantic Search Engine, which consists of Ontology development, Ontology Crawler, Ontology Annotator, Web Crawler, Semantic Search and Query Processor. They are using Ontology to store the structure of words and create domain-related information structures.

Another group of approaches to the search for semantically similar texts in the corpus of documents involves the presentation of documents in the form of numerical vectors, known as document embeddings.

There are many document embeddings techniques, for example, classic methods like bag-of-words, TF-IDF and Latent Dirichlet Allocation (LDA) or supervised and unsupervised machine learning algorithms.

LDA is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics.

Latent Dirichlet Allocation (LDA) [9] can be basically viewed as a model which breaks down the collection of documents into topics by representing the document as a mixture of topics with their probability distributions. The topics are represented as a mixture of words with a probability representing the importance of the word for each topic.

Wei Xing et al. [10] applied LDA algorithm for Ad-hoc retrieval task. They proposed an LDA-based document model within the language modeling framework, and evaluate it on several TREC collections.

Ai Wang et al. [11] proposed a LDA-based cross-language retrieval model that did not rely on word-by-word translation of query or document. The proposed LDA-based retrieval model was compared with three popular retrieval models: LDA-based mono-lingual document model; Mono-lingual TF.IDF retrieval model; Cross-lingual Latent Semantic Indexing retrieval model on CNKI datasets.

Recently, neural network language models have demonstrated promising performance by reducing time complexity and successfully solved many NLP problems. They effectively generate dense and short embeddings, namely word embeddings [12, 13]. For document embeddings, averaging word embeddings in a document could be a way for the representation. Q. Le and T. Mikolov [6] proposed a method to produce document vector, which is similar to word embeddings. The method directly produces document embeddings along with the word embeddings. They found document embedding very effective for the problem of sentiment analysis and document retrieval.

There are research papers that have already reviewed the Doc2Vec method. For instance, Wang S. et al. [14] compared the Doc2Vec and Ariadne document embedding approaches in the context of information retrieval. They evaluated these document embedding techniques in a specific information retrieval use case related to evidence-based medicine guidelines. However, experiment results show that Ariadne performs equally well as Doc2Vec in a specific information retrieval task.

The Doc2Vec often exhibits low accuracy if the training data consists of short sentences. Kurihara K. et al. [15] proposed a new method of supplementing the context of short sentences for the training phase of the Doc2Vec with the PV-DM model. This method uses target-topic IDs instead of sentence IDs as the context. Doc2Vec algorithm was modified with the hypothesis that other posts for the same topic (i.e. reviews for the same movie in online movie review sites) may share the same background. They conducted a large-scale experiment using movie review posts and proved the effectiveness of their approach.

Galke L. et al. [16] compared the performance of several techniques that leverage word embeddings in the retrieval models to compute the similarity between the query and the documents, namely word centroid similarity, paragraph vectors, Word Mover's distance. They also proposed novel inverse document frequency (IDF) re-weighted word centroid similarity and compared it with other approaches. They evaluated the performance using the ranking metrics mean average precision, mean reciprocal rank, and normalized discounted cumulative gain.

Hee Seok Cho et al. paper [17] is devoted to the implementation of a question and answer search system that automatically provides learners with the most similar questions by analyzing the questions and answers based on Doc2Vec embedding technologies.

BERT (Bidirectional Encoder Representations from Transformers) is a recent paper [18] published by researchers at Google AI Language. BERT's key technical innovation is applying the bidirectional training of Transformer models to language modelling. These models generate contextual embeddings of input tokens (commonly sub-word units), each infused with information of its neighborhood, but are not aimed at generating a rich embedding space for input sequences.

Sentence-BERT, presented in [19] aims to adapt the BERT architecture by using siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity.

Jiang Zhuolin at al. [20] in their paper explored the use of the BERT to model and learn the relevance between English queries and foreign-language documents in the task of cross-lingual information retrieval.

None of the reviewed studies applies the approach based on the Doc2Vec algorithm for semantic search in the software documentation during the maintenance phase. Doc2Vec is the robust, research-proven algorithm that shows effectiveness in semantic search and information retrieval tasks. Thus, the distinctive feature of our research is the use of the above algorithm to identify semantically related software documentation pages.

III. Proposed approach

In this paper, it is proposed to reduce the complexity of manual analysis of customer requests. The essence of the manual approach is to get a list of unresolved customer requests in the issue tracking system, and then examine and process each request in an iterative manner. During the request processing, engineers often have to refer to the documentation of the software product. The developer has to look for specific information among a large amount of text data. This process is inefficient in terms of time and can be optimized.

The workflow schema of the proposed automated approach to semantic search for documentation is shown in Fig. 1.

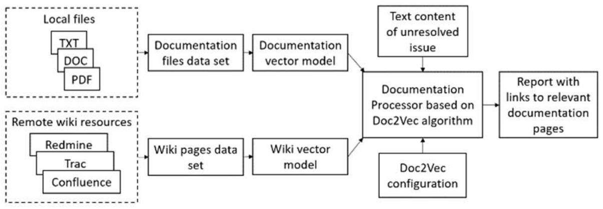

Local files
TXT
DOC
PDF
Remote wiki resources
Redmine
Trac
Confluence
Documentation files data set
Documentation vector model
Wiki pages data set
Wiki vector model
Text content of unresolved issue
Documentation Processor based on Doc2Vec algorithm
Doc2Vec configuration
Report with links to relevant documentation pages

Fig. 1. Workflow schema of proposed approach

The diagram shows that the proposed approach to semantic search in program documentation consists of the following steps:

- Creation of data sets from local and remote resources
- Vector models training
- Handling unresolved issues with the Doc2Vec algorithm

There are two types of data sources for creating data sets:

- Local document files of various formats, such as PDF, DOC (DOCX), RTF, TXT, PPT (PPTX), etc.
- Remote network resources, for instance, wiki web sites (Confluence, Redmine, Trac, etc.)

The structure of the data set is as follows: each line of the file begins with a unique identifier that points to a specific page of a particular document, then there is a separating character (for example, "|") followed by the rest of the text content from the specific page.

In addition to the data set file, a metadata file is created. It consists of lines divided by three columns: a unique identifier for the page, the name of the document, the page number in this document.

The metadata file is required to determine the document name and the page number, where the specific text is presented, by a unique page identifier.

Then for each data set, a vector model is created. When vector models are created, they are used by the Doc2Vec algorithm to find document pages, which are semantically similar to the unresolved issue text content.

As a result, the tool generates html report that contains references:

- To specific pages in local documents
- To specific pages on particular remote wiki sites

After receiving a report with the results, the user can discover the relevant parts of the documentation that will help to deal with the customer's problem quickly. There is no need to search for the necessary page in the document or in the wiki system.

Summing up, the proposed approach can be expressed in the formula 1:

$$U(L(T), I_i) = [Rate(t_0) \quad \dots \quad Rate(t_N)] \qquad (1)$$

The description of the values in the formula is as follows:

T – an array of text data from the documentation;

L(x) – learning function used on text data sets, as a result of the function is a vector model;

I_i – textual data describing a specific issue;

U(x, y) – function of applying a vector model to text data, which results in an array of the most similar documents;

t_i – one of the most similar documents;

N – number of documents found;

Rate(x) – a number that is expressed as a percentage and means the semantic similarity of t_i with the I_i.

IV. Training process of the Doc2Vec algorithm

A. *Vector representation of text data*

Numerical representation of text documents can be used for many purposes, for example, document classification, web search, spam filtering, similar bug report detection [7] and also for searching semantically related documents [21]. A simple and effective method of representing texts in the form of vectors is the use of the Doc2Vec algorithm.

Doc2Vec is an approach in the field of distributional semantics to the representation of documents in the form of vectors with a small fixed size. Doc2Vec is based on the Word2Vec approach [22, 23]. The difference between the Doc2Vec algorithm and Word2Vec is that, as a result of training, in addition to the word vectors W, vector representations of documents D will be obtained.

The schema of using the trained Doc2Vec model for finding similar documents is as follows: a new document is fed to the input of the neural network, and the output is a vector that identifies a document similar to the incoming one.

B. *The process of finding similar texts*

The semantic proximity of two texts is determined by the cosine similarity of their vectors. The cosine similarity [24] measures the cosine of the angle between two nonzero vectors. If the cosine is 1, then the angle between the vectors is 0 degrees. If the cosine is less than 1, then the angle between the vectors is in the interval [0; 0.5π). Thus, this metric determines the location of one vector in space relative to another vector. Two vectors with the same orientation in space have a cosine similarity of 1, and two vectors, which are located at an angle of 90° relative to each other, have a similarity of 0. Two diametrically opposite vectors have a similarity of -1. Cosine similarity is mainly used in positive vector space, in which the result is limited to [0,1]. Unit vectors are as similar as possible if they are parallel and as dissimilar as possible if they are orthogonal (perpendicular).

The coefficient of similarity is calculated by formula 2:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \qquad (2)$$

where A_i and B_i are the components of vectors A and B, respectively.

The cosine similarity measure is applicable for vector spaces with any number of dimensions. This measure is most often used in multidimensional positive spaces.

In the vector representation of text data, each word or document is matched with its unique vector. The cosine similarity between the two vectors of words or documents shows the probability of the semantic similarity of these two words or documents.

In the proposed approach of automated analysis of customer requests, it is proposed to use cosine similarity as a measure of the similarity of two texts. It is assumed that the two texts are similar in meaning if the cosine similarity of their vector representations is greater or equal to the coefficient with a value of 0.8. This ratio can be adjusted. The increase in the coefficient may be due to the desire to find a smaller number of the most similar text documents. At the same time, reducing this ratio will lead to more search results among semantically similar documents.

V. Software tool implementation

The developed tool that implements the proposed automated approach is written in Java 8 and consists of several modules:

- 2 connectors to remote web systems (Jira, Confluence)
- 2 processors (Documentation and Confluence)
- HTML report generator
- Process executor which coordinates the work of the entire system

The architecture of the tool is shown in Fig. 2:

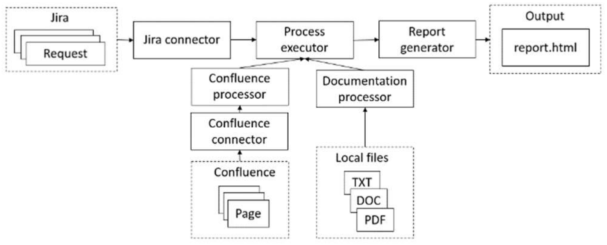


Fig. 2. The tool architecture

The Jira and Confluence connectors were developed and used to load data from Jira and Confluence web resources, respectively. The services are connected to via corresponding REST API interfaces. Anonymous or basic authentication with a username and password is possible.

Jira Connector obtains a textual description of unresolved requests from Jira. The Confluence connector gains textual representation of the wiki pages.

The Jira Rest Java Client library (JRJC)[1] is used in the Jira connector. This Java library allows one to connect to any instance of Jira 4.2+ using the REST API. JRJC currently provides a thin layer of abstraction on top of the REST API, as well as the client-side Jira object model. These objects represent request entities: Issue, Priority, Resolution, Status, User, etc.

The Confluence Rest Java Client library (CRJC)[2] is used in the Confluence connector. As well as JRJC, the CRJC provides a thin layer of abstraction on top of the Confluence REST API.

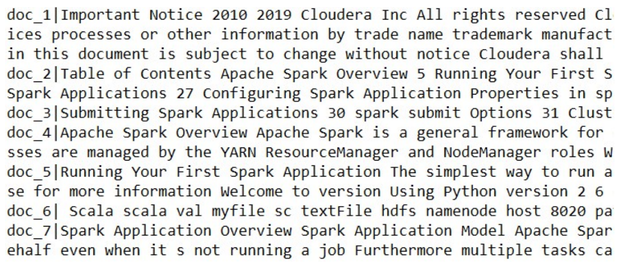
Each processor is responsible for three things:

- creation of a data set from raw sources
- creation of vector models using data sets
- usage of created model to find related documentation for unresolved requests

The Documentation processor uses local text files to create Documentation_DataSet.txt file, and the Confluence processor uses text content from wiki pages to create Confluence_DataSet.txt file.

The structure of documentation data set is as follows: each line of the file begins with a unique identifier that points to a specific page of a particular document, then there is a separating vertical bar character and then all text content from a specific page.

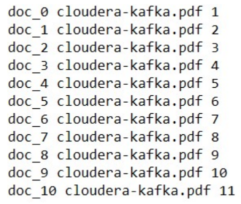
The format of the created data set file can be seen in Fig. 3.



```
doc_1|Important Notice 2010 2019 Cloudera Inc All rights reserved Cl
ices processes or other information by trade name trademark manufact
in this document is subject to change without notice Cloudera shall
doc_2|Table of Contents Apache Spark Overview 5 Running Your First S
Spark Applications 27 Configuring Spark Application Properties in sp
doc_3|Submitting Spark Applications 30 spark submit Options 31 Clust
doc_4|Apache Spark Overview Apache Spark is a general framework for
sses are managed by the YARN ResourceManager and NodeManager roles W
doc_5|Running Your First Spark Application The simplest way to run a
se for more information Welcome to version Using Python version 2 6
doc_6| Scala scala val myfile sc textFile hdfs namenode host 8020 pa
doc_7|Spark Application Overview Spark Application Model Apache Spar
ehalf even when it s not running a job Furthermore multiple tasks ca
```

Fig. 3. Part of the documentation data set file

In addition to the data set file, a metadata file is created. Its structure can be seen in Fig. 4.



```
doc_0 cloudera-kafka.pdf 1
doc_1 cloudera-kafka.pdf 2
doc_2 cloudera-kafka.pdf 3
doc_3 cloudera-kafka.pdf 4
doc_4 cloudera-kafka.pdf 5
doc_5 cloudera-kafka.pdf 6
doc_6 cloudera-kafka.pdf 7
doc_7 cloudera-kafka.pdf 8
doc_8 cloudera-kafka.pdf 9
doc_9 cloudera-kafka.pdf 10
doc_10 cloudera-kafka.pdf 11
```

Fig. 4. Part of the documentation metadata file

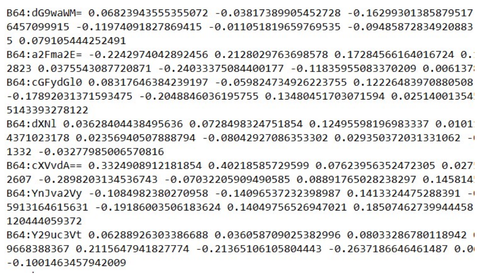
The structure of confluence data set is similar to the one used in the documentation, but the unique identifier of each row points to a particular confluence wiki page ID directly and there is no need to create a metadata file in this case.

The basis of the processors is the Doc2Vec algorithm, which is implemented in the Java library Deeplearning4j[3]. The work with the algorithm is divided into three stages: data preparation, training and use.

Before transferring the contents of the data set *.txt files to the input of the Doc2Vec algorithm, it is necessary to prepare text data to create a vector model. The preparation process consists of tokenization [25], stemming [26] and removing stop words. The software tool uses a popular implementation of Porter's stemmer [27, 28] for the Java language.

[1] https://bitbucket.org/atlassian/jira-rest-java-client
[2] https://docs.atlassian.com/atlassian-confluence/5.9.2/
[3] https://deeplearning4j.org/

The data set files generated by the Documentation and Confluence connectors and containing preprocessed page texts are used for the training. As a result of training, a vector model is obtained, which is subsequently serialized into the Documentation_VectorModel.zip and Confluence_VectorModel.zip files, respectively. Vectors represent the meaning of requests, and using mathematical operations on vectors, you can find similarities between different requests. The example of document vectors is shown in Fig. 5.



```
B64:dG9waWM= 0.06823943555355072 -0.03817389905452728 -0.16299301385879517
6457099915 -0.11974091827869415 -0.011051819659769535 -0.094858728349208835
5 0.079105444252491
B64:a2Fma2E= -0.2242974042892456 0.2128029763698578 0.17284566164016724 0.
2823 0.0375543087720871 -0.24033375084400177 -0.11835955083370209 0.006137
B64:cGFydGl0 0.08317646384239197 -0.05982473492622375 0.12226483970880508
-0.17892031371593475 -0.2048846036195755 0.13480451703071594 0.02514001354
5143393278122
B64:dXNl 0.03628404438495636 0.0728498324751854 0.12495598196983337 0.0101
4371023178 0.02356940507888794 -0.08042927086353302 0.029350372031331062 -
1332 -0.03277985006570816
B64:cXVvdA== 0.3324908912181854 0.40218585729599 0.07623956352472305 0.027
2607 -0.289820313453674З -0.07032205909490585 0.08891765028238297 0.145814
B64:YnJva2Vy -0.1084982380270958 -0.14096537232398987 0.141332447528839l -
5913164615631 -0.19186003506183624 0.14049756526947021 0.18507462739944458
120444059372
B64:Y29uc3Vt 0.06288926303386688 0.036058709025382996 0.08033286780118942
9668388367 0.2115647941827774 -0.21365106105804443 -0.2637186646461487 0.0
-0.1001463457942009
```

Fig. 5. Vector representations of the documents

All settings for the training are contained in the doc2vec.properties configuration file. This file contains the following fields:

- minWordFrequency — defines the minimum word frequency in a training dataset, all words that are less than this threshold will be deleted before learning the model,
- iterations — determines the number of learning iterations performed for each part of the training body,
- epochs — the number of iterations across the entire training body,
- layerSize — number of output vectors,
- learningRate — initial learning speed of the model [6],
- windowSize — context window size [6],
- sampling — determines whether to use a subsample or not, to establish a selection, this field should have a value greater than 0.

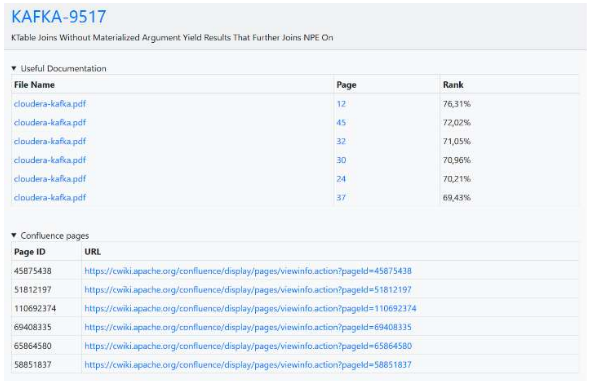
The VectorModel.zip archive contains the following files:

- codes.txt — codes for the Huffman tree [29],
- config.json — settings of the Doc2Vec algorithm,
- frequencies.txt — metrics tf-idf [30] and bag-of-words [23],
- huffman.txt — coordinates of the Huffman tree points,
- labels.txt — list of resolved requests in base64 format,
- syn0.txt — weights of connections between input and hidden neurons of the network,
- syn1.txt — weights of connections between hidden and output neurons of the network.

After learning the Doc2Vec algorithm, it is possible to use it. To do this, a model is loaded into memory from the VectorModel.zip file, in which each request is represented as a numerical vector and is associated with a particular documentation or wiki page identifier. After that, the text content of unresolved requests is sent to the input of the Doc2Vec algorithm.

The search process for semantically related documentation is as follows. First, a numerical vector is formed from the incoming unresolved request. After that, this vector is compared with the vectors of the documentation pages and with the vectors of wiki pages. In this case, the similarity of the text data is determined by the cosine proximity coefficient of their vector representations. The greater the coefficient value is, the more confidently it can be argued that the two texts are similar.

Thus, the result of the algorithm is a list with identifiers of the documentation pages most similar to the input unresolved requests. All this data is then compiled into a report and presented to the user.



KAFKA-9517

KTable Joins Without Materialized Argument Yield Results That Further Joins NPE On

▾ Useful Documentation

File Name	Page	Rank
cloudera-kafka.pdf	12	76,31%
cloudera-kafka.pdf	45	72,02%
cloudera-kafka.pdf	32	71,05%
cloudera-kafka.pdf	30	70,96%
cloudera-kafka.pdf	24	70,21%
cloudera-kafka.pdf	37	69,43%

▾ Confluence pages

Page ID	URL
45875438	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=45875438
51812197	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=51812197
110692374	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=110692374
69408335	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=69408335
65864580	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=65864580
58851837	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=58851837

Fig. 6. Final report with related documentation references

The process executor coordinates the work of all these components. It first starts the Jira connector, retrieves a list of unresolved requests, and then iteratively sends each request to both processors. The results of processing are sent to the report generator, which creates the report.html file. An example of such report is shown in Fig. 6.

VI. Evaluation of Document Embedding Methods

This section compares the results of using various document embedding methods in the proposed approach to find semantically similar software documentation. In addition to Doc2Vec, two known methods, LDA and BERT, were also chosen for evaluation.

To assess the quality of the useful documentation search for one request, the classic measures for the information retrieval task were selected: precision, recall and F1-score. In our case, any search query is represented by all text data from an unresolved request. Relevant documentation pages are those pages that helped in solving the request and retrieved pages are those that are presented in the final report.

Precision (formula 3) is the fraction of the documentation pages retrieved that are relevant to the user's information need.

$$precision = \frac{|\{relevant\ pages\} \cup \{retrieved\ pages\}|}{|\{retrieved\ pages\}|} \quad (3)$$

Recall (formula 4) is the fraction of the documentation pages that are relevant to the query that are successfully retrieved.

$$recall = \frac{|\{relevant\ pages\} \cup \{retrieved\ pages\}|}{|\{relevant\ pages\}|} \quad (4)$$

The traditional F-measure or balanced F-score (formula 5) is the weighted harmonic mean of precision and recall.

$$F_1 = \frac{2*precision*recall}{precision+recal} \qquad (5)$$

The described measures were used to evaluate 100 requests using three document embedding methods: LDA, Doc2Vec and BERT. The results for the first three requests are shown in the Table I.

TABLE I. EVALUATION OF PRECISION, RECALL AND F1-SCORE FOR 3 REQUESTS WITH LDA, DOC2VEC AND BERT ALGORITHMS

Request ID	**Methods**	**Measures**		
		precision	***recall***	***F1***
1	LDA	0.815	0.716	0.762
	Doc2Vec	0.885	0.906	0.895
	BERT	0.693	0.851	0.764
2	LDA	0.844	0.809	0.826
	Doc2Vec	0.950	0.821	0.881
	BERT	0.791	0.738	0.763
3	LDA	0.748	0.816	0.781
	Doc2Vec	0.839	0.891	0.864
	BERT	0.782	0.810	0.796

The table shows that the Doc2Vec algorithm basically has better precision, recall and F1 scores than LDA and BERT algorithms.

Precision, recall and F1 are single-value metrics based on the whole list of documents returned by the system. For systems that return a ranked sequence of documents, it is desirable to also consider the order in which the returned documents are presented. By computing a precision at every position in the ranked sequence of documents, it is possible to calculate the average precision (formula 6) of the algorithm:

$$AP@n = \frac{\sum_{k=1}^{n}(P(k)*rel(k))}{R}, \qquad (6)$$

where *k* is the rank in the sequence of retrieved documents; *n* is the number of retrieved documents; *P(k)* is the precision at cut-off *k* in the list; *R* is number of relevant documents; *rel(k)* is an indicator function equaling 1 if the item at rank *k* is a relevant document, zero otherwise.

Mean average precision (MAP) (formula 7) for a set of queries is simply the mean of all the average precision scores for each query.

$$MAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q}, \qquad (7)$$

where *Q* is the number of queries in the set and *AP(q)* is the average precision for a given query, q.

We calculate the MAP measure depending on how many results for each query are retrieved by the developed system: 5, 10 and 20.

The Table II shows that the MAP score is higher for the Doc2Vec algorithm and that it is most optimal to return 10 the most similar documentation pages for each query. This is due to the fact that with 5 results, the remaining important pages are lost. And with 20 results, extra pages appear that do not help in solving the request.

TABLE II. EVALUATION OF MAP-SCORE WITH 5, 10 AND 20 RETRIEVED DOCUMENTS FOR 100 REQUESTS WITH LDA, DOC2VEC AND BERT ALGORITHMS

Methods	**Measures**		
	MAP@5	***MAP@10***	***MAP@20***
LDA	0.834	0.859	0.848
Doc2Vec	0.876	0.921	0.893
BERT	0.745	0.837	0.812

The diagram (Fig. 7) clearly shows the results of comparing three document embedding methods in the task of finding relevant documentation pages for 100 unresolved requests.

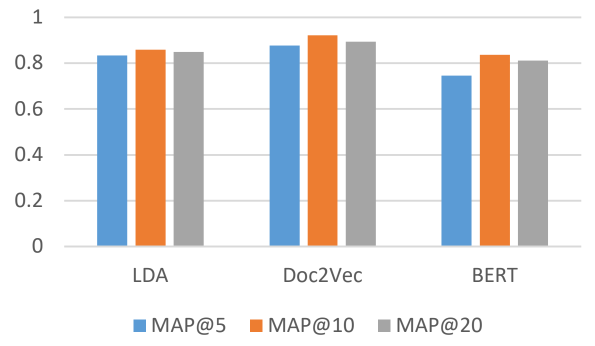


Fig. 7. Graph showing the MAP scores of LDA, Doc2Vec and BERT algorithms in the task of finding proper software documentation for 100 unresolved requests

As a result of evaluation, it can be concluded that the Doc2Vec algorithm shows the best results in the task of finding the most similar documentation pages for unresolved requests. Therefore, this algorithm is used in the developed tool.

VII. EVALUATION OF DOC2VEC MODEL

In order to evaluate the accuracy of the created vector models, it is necessary to apply the Doc2Vec algorithm to the test data.

The test data set is represented by a text file consisting of several lines. In this paper, we consider a test file with the number of lines equal to 100. Each line is a test step and is divided into three columns by a special separating character. The first column contains the text fragment of the original request. The second column contains the text of a request that is close in meaning. In the third column, on the contrary, is the request text, which is completely unrelated to the original.

The task of testing is to determine how many lines out of 100 will be successfully evaluated.

The evaluation is as follows: If the first two columns are similar with the coefficient more than 80%, and the first and third columns are similar to less than 20%, then we assume that this row is evaluated correctly.

After evaluating all the lines, we summarize the number of correct evaluations and divide it by the number of lines. The resulting number shows the accuracy of the vector model.

Typically, model creation is carried out in several epochs. During testing, we found out that 12 epochs are quite enough to create an accurate model (Fig. 8). The accuracy of the model in this case is 90%.

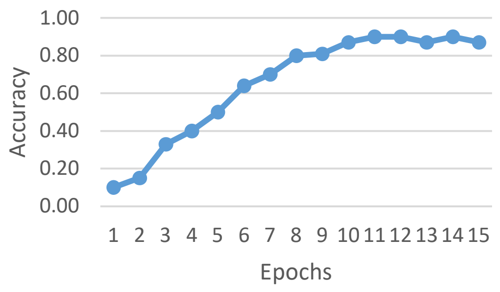


Fig. 8. Graph showing the accuracy of the doc2vec model depending on the number of epochs

During testing and selection of Doc2Vec hyper parameters, it was found out that the following configuration is the most optimal in quality and execution time of the algorithm:

- minWordFrequency = 1
- iterations = 5
- epochs = 12
- layerSize = 100
- learningRate = 0.025
- windowSize = 5
- sampling = 0

VIII. Experiment

The developed tool was used on the open-source project Apache Kafka[4].

Any engineer can find an error in the work of this program and register a request containing a question or description of an error in the corresponding Jira issue tracking system[5]. For the experiment, 100 outstanding requests were retrieved from the Jira.

The Apache Kafka Guide.pdf[6] was used as the local documentation file. The Kafka project space[7] was also used as a remote wiki resource.

The tool is deployed on a platform that serves as a local workstation with the Windows 10 Enterprise x64 operating system, an Intel Core i7-4810MQ processor 2.80 GHz processor and 16 GB RAM.

The Documentation_DataSet.txt file (235 KB) was generated from the documentation pdf file. The number of lines in the data set is equal to the number of lines in the pdf file and accounts for 184 pages.

The Confluence_DataSet.txt data set was generated with a size of 5,21 MB and a number of lines of 765, which corresponds to the number of wiki pages downloaded.

Then from the created data sets the vector models Documentation_VectorModel.zip (3,19 MB) and Confluence_DataSetVectorModel.zip (35,5 MB) were created, respectively.

Creating datasets took 19 minutes, and creating a model took 12 minutes.

During the process of working on the Apache Kafka project, two experiments were conducted. The essence of the first experiment is to apply the tool to unresolved requests and evaluate its effectiveness. It is necessary to understand the percentage of cases, when the tool finds at least one relevant page in the documentation, as well as the number of cases, when this finding was correct and useful for solving the problem. A page is considered found if the semantic proximity between the request text and page text is 80%. At the same time, the page found is considered useful if it can indeed help in solving the problem.

Before experiments, it is necessary to define search process specification for manual and automated approaches.

Manual approach scenario for documentation search:

1. Read an unresolved request
2. Try to find proper information by keywords for this request in the documentation files
3. Try to find proper information by keywords for this request in the wiki system

Automated approach scenario for documentation search:

1. Read an unresolved request
2. Configure the automated tool to get a report for this unresolved request
3. Run the tool and wait until the end of its processing
4. Review the generated report, check proposed documentation pages in local files and wiki
5. If the system does not propose any useful pages, then do the manual search

In the search process, the developer uses only local documentation files and wiki systems on which the Doc2Vec algorithm was trained. The usage of search engines such as Google is not allowed for the purity of the experiment.

As part of the experiment, 100 requests were downloaded from Jira and analyzed. The results of the analysis of these requests are presented in Fig. 9.

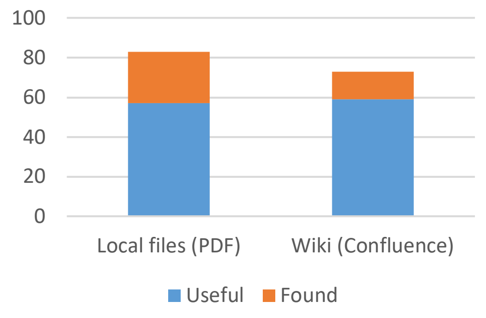


Fig. 9. The results of requests analysis using the automated tool

[4] https://kafka.apache.org
[5] https://issues.apache.org/jira/projects/KAFKA
[6]https://docs.cloudera.com/documentation/enterprise/6/latest/PDF/cloudera-kafka.pdf
[7] https://cwiki.apache.org/confluence/display/KAFKA

The diagram shows the number of found and the number of useful pages in the documentation. The 83 related pages from local files were found, and 57 of them turned out to be useful for solving the problem. It means that the precision of the algorithm for local files is 69%. With regards to the wiki, the 73 related pages were found, and 59 of them are useful to resolution. The precision in this case is 81%. Wiki pages are more helpful because they are often updated with relevant information and they contain a more detailed description of the internal structure of software products.

The second experiment compared the effectiveness of the manual and automated approaches to searching the useful documentation. The second engineer carried out the manual search for the same 100 requests. Furthermore, both engineers participating in the experiment had similar work experience and qualifications. The second participant had not yet seen those requests that were resolved with the automated approach.

To compare the search time for useful documentation pages for each request, a regular timer was used. The timer turned on when the engineer started working on a new request and turned off when the engineer found at least one page of the documentation that proved to be useful for solving the request. The results of the experiment can be seen in Fig. 10.

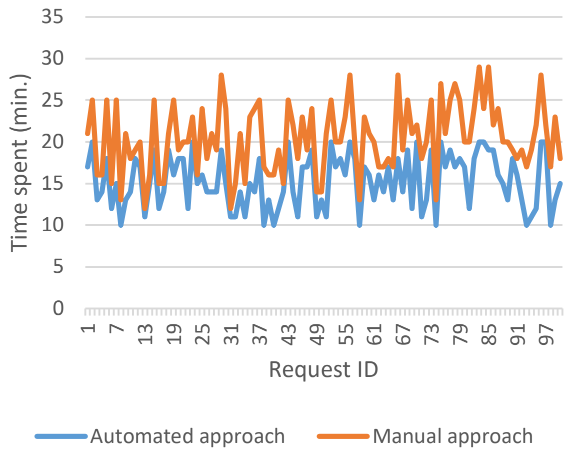


Fig. 10. A comparison of the time spent to search for useful documentation with the manual and automated approach by two developers for 100 requests

The chart shows the time difference between the manual and automated approaches to finding useful software documentation. The average time to find the proper documentation using the manual approach was 12.2 minutes, and the average time spent searching for the relevant documentation with the proposed automated approach was 18.4 minutes. In this case, the decrease in the complexity of the search for documentation amounted to 33.7%.

For more confident results, it was necessary to conduct this experiment with a large number of people. Therefore, another 10 pairs of developers were invited. One of the pair of developers tried to solve 100 requests manually, and the other of the pair tried to solve the same 100 requests in an automated way.

The results of the experiment conducted with the participation of 20 developers on 100 requests are shown in Fig. 11. The chart shows the average time spent by developer searching for the proper documentation for one request.

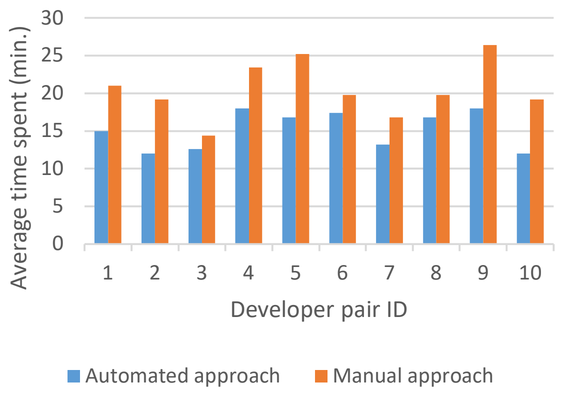


Fig. 11. A comparison of the average time spent to search for useful documentation with the manual and automated approach by 10 pairs of developers for 100 requests

The chart shows that for 100 unresolved requests the automated approach takes significantly less time than the manual approach. The average time spent by 10 developers using the manual approach to find the proper documentation pages was 20.5 minutes, and the average time spent searching for relevant documentation pages by another 10 developers using the proposed automated approach was 15.2 minutes. Thus, we can conclude that the decrease in the complexity of the search for documentation amounted to 25.9%.

IX. Conclusion

The study reviewed the related work in the field of semantic search in the text documents.

An automated approach to reduce the complexity of the customer requests processing is proposed. This approach is based on the use of the Doc2Vec machine learning algorithm, which solves the problem of semantic search in the related documentation.

The created tool was successfully tested on the Apache Kafka project. As a result of using the tool, 100 requests were analyzed. The effectiveness of its use is shown. The results show the benefits of using the software. The average time for analyzing documentation has decreased compared to the traditional manual approach.

References

[1] Y.B. Leau, W.K. Loo, W.Y. Tham, and S.F. Tan, "Software development life cycle AGILE vs traditional approaches," *International Conference on Information and Network Technology*, vol. 37, no. 1, pp. 162-167, 2012.

[2] E.E. Ogheneovo, "On the relationship between software complexity and maintenance costs," *Journal of Computer and Communications*, vol. 2, no. 14, p. 1, 2014.

[3] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams," *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pp. 291-300, February 2010.

[4] J. Fisher, D. Koning, and A.P. Ludwigsen, "Utilizing atlassian jira for large-scale software development management," *14th International Conference on Accelerator & Large Experimental Physics Control Systems (ICALEPCS)*, October 2013.

[5] T. Pellegrini, "Comparing SVM, Softmax, and shallow neural networks for eating condition classification," *16th Annual Conference of the International Speech Communication Association*, pp. 899-903, 2015.

[6] Q. Le, T. Mikolov. “Distributed representations of sentences and documents,” *International conference on machine learning*, pp. 1188-1196, 2014.

[7] A. Kovalev, N. Voinov, and I. Nikiforov. “Using the Doc2Vec Algorithm to Detect Semantically Similar Jira Issues in the Process of Resolving Customer Requests,” *Intelligent Distributed Computing XIII*, pp. 96-101, 2020.

[8] Kassim, J. M., & Rahmany, M. (2009, August). Introduction to semanticsearch engine. In 2009 International Conference on Electrical Engineer-ing and Informatics (Vol. 2, pp. 380-386). IEEE

[9] Blei, D., Ng, A., Jordan, M.: “Latent dirichlet allocation,” *The Journal of Machine Learning Research 3, pp.* 993–1022, 2003.

[10] Wei, Xing & Croft, W. “LDA-based document models for Ad-hoc retrieval,” *Proceedings of the 29th annual international ACM SIGIR*. pp. 178-185, 2006, 10.1145/1148170.1148204

[11] Ai Wang, Yao Dong Li and Wei Wang, “Cross language information retrieval based on LDA,” *IEEE International Conference on Intelligent Computing and Intelligent Systems*, Shanghai, pp. 485-490, 2009, doi: 10.1109/ICICISYS.2009.5358121

[12] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space,” *Proceedings of Workshop at ICLR*, arXiv preprint arXiv:1301.3781, 2013.

[13] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “Glove: Global vectors for word representation,” *EMNLP*, 2014.

[14] Wang S., Koopman R. “Semantic Embedding for Information Retrieval,” *BIR@ ECIR*, pp. 122-132, 2017.

[15] Kurihara K. et al. “Target-Topic Aware Doc2Vec for Short Sentence Retrieval from User Generated Content,” *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services.* pp. 463-467, 2019.

[16] Galke L., Saleh A., Scherp A. “Word embeddings for practical information retrieval,” *INFORMATIK*, 2017.

[17] Hee Seok Cho, Yong Kim. “Development of Doc2Vec-based Question and Answer Search System,” *The 3rd International Conference on Interdisciplinary research on Computer science, Psychology, and Education (ICICPE’ 2019),* December 17-19, 2019.

[18] Devlin, Jacob & Chang, Ming-Wei & Lee, Kenton & Toutanova, Kristina. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” 2018.

[19] Reimers, Nils & Gurevych, Iryna. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” 2019.

[20] Jiang, Zhuolin & El-Jaroudi, Amro & Hartmann, William & Karakos, Damianos & Zhao, Lingjun. “Cross-lingual Information Retrieval with BERT,” 2020.

[21] N. Ur-Rahman and J.A. Harding, “Textual data mining for industrial knowledge management and text classification: A business oriented approach,” *Expert Systems with Applications*, vol. 39, no. 5, pp. 4729-4739, 2012.

[22] L. Wolf, Y. Hanani, K. Bar, and N. Dershowitz, “Joint word2vec Networks for Bilingual Semantic Representations,” *Int. J. Comput. Linguistics Appl.*, vol. 5, no. 1, pp. 27-42, 2014.

[23] Y. Zhang, Y. Jin, and Z.H. Zhou,. “Understanding bag-of-words model: a statistical framework,” *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 43-52, 2010.

[24] G.L Giller, “The Statistical Properties of Random Bitstreams and the Sampling Distribution of Cosine Similarity,” *Giller Investments Research Notes*, no. 20121024/1, 2012.

[25] A.H. Branco and J.R. Silva. “Contractions: breaking the tokenization-tagging circularity,” *Lecture Notes in Computer Science*, vol. 2721, pp. 167-170, 2003.

[26] D. Sharma, “Stemming algorithms: A comparative study and their analysis,” *International Journal of Applied Information Systems*, vol. 4, no. 3, pp. 7-12, 2012.

[27] M.F. Porter, “An algorithm for suffix stripping,” *Program*, vol. 14, no. 3, pp. 130-137, 1980.

[28] P. Willett, “The Porter stemming algorithm: then and now,” *Program: Electronic Library and Information Systems*, vol. 40, no. 3, pp. 219-223, 2006.

[29] M. Sharma, “Compression using Huffman coding,” *International Journal of Computer Science and Network Security*, vol. 10,no. 5, pp. 133-141, 2010.

[30] J. Ramos, “Using tf-idf to determine word relevance in document queries,” *Proceedings of the first instructional conference on machine learning*, vol. 242, pp. 133-142, December 2003.