УДК 004.416
doi:10.31799/1684-8853-2021-1-17-27

Научные статьи
Articles

Автоматизированный подход к семантическому поиску по программной документации на основе алгоритма Doc2Vec

А. Д. Ковалев[a], *аспирант, ассистент, orcid.org/0000-0002-4610-5524, kov3000@ya.ru*
И. В. Никифоров[a], *канд. техн. наук, доцент, orcid.org/0000-0002-2330-2197*
П. Д. Дробинцев[a], *канд. техн. наук, доцент, orcid.org/0000-0003-1116-7765*
[a]*Санкт-Петербургский политехнический университет Петра Великого, Политехническая ул., 29, Санкт-Петербург, 195251, РФ*

Введение: *одним из значимых этапов жизненного цикла разработки программного обеспечения является этап его поддержки, когда заказчики могут обращаться в службу поддержки компании-поставщика с вопросами, проблемами и предложениями. Для решения поступившего запроса инженеры пользуются соответствующей документацией. С целью снизить трудоемкость и повысить качество этапа сопровождения можно автоматизировать поиск необходимых страниц, параграфов и предложений документации.* ***Цель:*** *разработка подхода к семантическому поиску по документации с использованием алгоритма машинного обучения Doc2Vec для автоматизации решения запросов заказчиков.* ***Результаты:*** *предложен подход к семантическому поиску по текстовым файлам документации и вики-страницам с использованием алгоритма машинного обучения Doc2Vec. Страницы документации, которые имеют семантическое сходство с текстовым описанием неразрешенного запроса заказчика, помогают разработчику более эффективно обрабатывать входящий запрос. На базе предложенного подхода разработан программный инструмент, предоставляющий инженеру отчет со ссылками на семантически близкие к нерешенному запросу страницы документации. Во время испытаний инструмента установлены оптимальные параметры алгоритма Doc2Vec, которые обеспечивают необходимое качество семантического поиска. Идея эксперимента заключалась в применении инструмента к нерешенным запросам и оценке его эффективности. Предложенный подход и реализующий его инструмент успешно протестированы на проекте с открытым исходным кодом Apache Kafka. В рамках эксперимента загружено и проанализировано 100 запросов из системы отслеживания ошибок Jira. Результаты эксперимента показывают преимущество использования инструмента в процессе поддержки программного продукта. Среднее время анализа документации сократилось по сравнению с традиционным ручным подходом.* ***Практическая значимость:*** *результаты исследований использованы при решении реальных запросов заказчиков. Разработанный подход и реализованное на его основе программное средство позволяют сократить трудоемкость этапа сопровождения.*

Ключевые слова *— поддержка программного обеспечения, автоматизация, Doc2Vec, машинное обучение, семантический поиск, документация.*

Для цитирования: Ковалев А. Д., Никифоров И. В., Дробинцев П. Д. Автоматизированный подход к семантическому поиску по программной документации на основе алгоритма Doc2Vec. *Информационно-управляющие системы*, 2021, № 1, с. 17–27. doi:10.31799/1684-8853-2021-1-17-27

For citation: Kovalev A. D., Nikiforov I. V., Drobintsev P. D. Automated approach to semantic search through software documentation based on Doc2Vec algorithm. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2021, no. 1, pp. 17–27 (In Russian). doi:10.31799/1684-8853-2021-1-17-27

Введение

Этап сопровождения программного обеспечения (ПО) является одним из наиболее значимых и трудоемких этапов жизненного цикла разработки [1, 2]. В процессе сопровождения ПО разработчики компании-поставщика исправляют ошибки и решают проблемы, возникающие в ходе эксплуатации программного продукта на стороне заказчика. Например, причина неправильной работы ПО может быть связана с некорректной конфигурацией системы или с дефектом в коде программы. На данном этапе происходит прием и обработка запросов заказчика инженером технической поддержки или программистом. Запрос, как правило, написан на естественном, неформализованном языке и содержит описание проблемы в программном продукте. Для удобного хранения и управления такими запросами обычно используются системы отслеживания ошибок [3]. В данной работе рассмотрена система отслеживания ошибок Jira [4].

В процессе решения проблемы заказчика, как правило, необходимо обратиться к соответствующей документации или базе знаний. Документация может быть представлена в виде локальных текстовых файлов разных форматов (PDF, DOC, RTF и др.), а также содержаться на удаленных сайтах. Частный случай сайта с документацией — это вики-система Confluence. Именно ее мы и рассмотрим в данной работе.

Документация может содержать очень большой объем информации, и поиск необходимых страниц может занять много времени, что делает

процесс изучения документации трудоемким. Как правило, поиск в текстах документов осуществляется по ключевым словам. Однако данный подход не эффективен, если поисковый запрос становится слишком большим и начинает содержать больше пяти слов. Главным недостатком поиска по ключевым словам является невозможность определить синонимы к словам в поисковом запросе. Также в процессе поиска можно пропустить смысловые части документов, которые совпадают по семантике, но не содержат ключевые слова.

В решении задачи семантического поиска по документации могут помочь алгоритмы машинного обучения и нейронные сети [5], а именно алгоритм Doc2Vec [6]. За счет использования программного средства, основанного на алгоритме Doc2Vec, возможно снизить трудоемкость и повысить эффективность процесса сопровождения. Повышение качества сопровождения помогает выстроить долгосрочные отношения поставщика ПО с заказчиком.

Целью данной работы является сокращение трудоемкости поиска необходимой информации по документации за счет автоматизированного подхода, основанного на применении алгоритма Doc2Vec. Для достижения поставленной цели необходимо разработать программное средство, которое реализует предложенный подход, а также показать эффективность применения предложенного подхода и его реализации в программном средстве на актуальных данных.

Актуальность исследования обусловлена тем, что в процессе развития и усложнения программных продуктов увеличивается количество написанного исходного кода. Это неизбежно приводит к повышению числа дефектов и недоработок в ПО, что является причиной обращения заказчиков в службу поддержки компании-поставщика [7].

Обзор литературы

Существует множество исследований в области повышения эффективности сопровождения программных продуктов за счет внедрения современных методов работы с программной документацией.

Aghajani E. и др. [8] провели крупномасштабное эмпирическое исследование, в котором проанализировали и классифицировали 878 артефактов, имеющих отношение к программной документации. Результатом их работы явился подробный обзор проблем, связанных с документацией, и ряд действенных предложений как для исследователей, так и для практиков. Кроме того, рассмотрены решения, которые применяются при возникновении этих проблем.

На практике документация, как правило, обладает многочисленными проблемами, такими как недостаточное содержание и устаревшая неоднозначная информация. Чтобы противостоять этому, исследователи изучают разработку систем, которые автоматически генерируют высококачественную документацию. Liu M. и др. в статье [9] представляют инструмент OpenAPIDocGen2, который генерирует документацию на основе анализа исходного кода. Данный инструмент создает комбинированную документацию, которая включает в себя описания функций, директивы, концепции предметной области, а также примеры и сценарии использования.

Современная программная документация так же сложна, как и само ПО. В течение жизненного цикла документация накапливает множество «почти повторяющихся» фрагментов, т. е. фрагментов текста, которые были скопированы из одного источника и позже изменены различными способами. Такие дубликаты снижают качество документации и затрудняют ее дальнейшее использование. Luciv D. V. и др. [10] представляют алгоритм обнаружения дубликатов в программных документах, основанный на использовании адаптированного инструмента Clone Miner.

Для того чтобы эффективно пользоваться документацией, недостаточно правильно ее написать и оформить. Также необходимо обеспечить эффективный поиск по ней. Калиниченко А. В. [11] предлагает интерактивный метод поиска похожих текстовых документов, позволяющий повысить пертинентность поиска. Диалог с пользователем, используемый в методе, позволяет уточнить информационную потребность и построить более точный поисковый запрос.

Также существует множество исследований в области технологий семантического поиска по текстовым документам. Семантический поиск является актуальной областью исследований в сфере компьютерных наук [12].

Использование онтологий — один из подходов к семантическому поиску. Например, Kassim J. M. и Rahmany M. [13] предлагают семантическую поисковую систему, которая состоит из сканера онтологий, аннотатора онтологий, веб-сканера, модуля семантического поиска и обработчика запросов. Они используют онтологию для хранения структуры слов и создания информационных структур, связанных с доменом.

Другая группа подходов к поиску семантически похожих текстов в корпусе документов включает представление документов в виде числовых векторов. Существует множество техник численного представления документов, например, bag-of-words, TF-IDF (term frequency-inverse document frequency), латентное распределение

Дирихле (Latent Dirichlet Allocation — LDA) и алгоритмы машинного обучения.

Латентное распределение Дирихле можно в основном рассматривать как модель, которая разбивает набор документов на темы, представляя документ как смесь тем с их распределениями вероятностей.

Wei Xing и Croft W. [14] применяют алгоритм LDA для Ad-hoc поиска. Они предлагают модель документа на основе LDA в рамках фреймворка для языкового моделирования и оценивают ее на нескольких тестовых наборах данных TREC.

Ai Wang, Yao Dong Li и Wei Wang [15] предлагают метод межъязыкового поиска на основе LDA, который не полагается на пословный перевод запроса или документа.

Существуют исследовательские работы, в которых рассматривается алгоритм Doc2Vec. Например, Wang S. и Koopman R. [16] сравнивают подходы к векторному представлению документов Doc2Vec и Ariadne в контексте поиска информации. Эти подходы были протестированы на наборе документов, связанных с доказательной медициной. Однако результаты экспериментов показывают, что Ariadne работает так же хорошо, как и Doc2Vec в специфической задаче поиска информации.

Doc2Vec часто показывает низкую точность, если данные для обучения состоят из коротких предложений. Kurihara K. и др. [17] предлагают новый метод дополнения контекста коротких предложений для этапа обучения Doc2Vec. Этот метод использует идентификаторы целевой темы вместо идентификаторов предложений в качестве контекста. Они провели масштабный эксперимент на основе данных, связанных с обзорами фильмов, и доказали эффективность своего подхода.

Ни в одном из рассмотренных исследований не применяется подход, основанный на алгоритме Doc2Vec для семантического поиска в документации ПО на этапе сопровождения. Таким образом, отличительной чертой настоящего исследования является использование вышеуказанного алгоритма для выявления семантически связанных страниц документации.

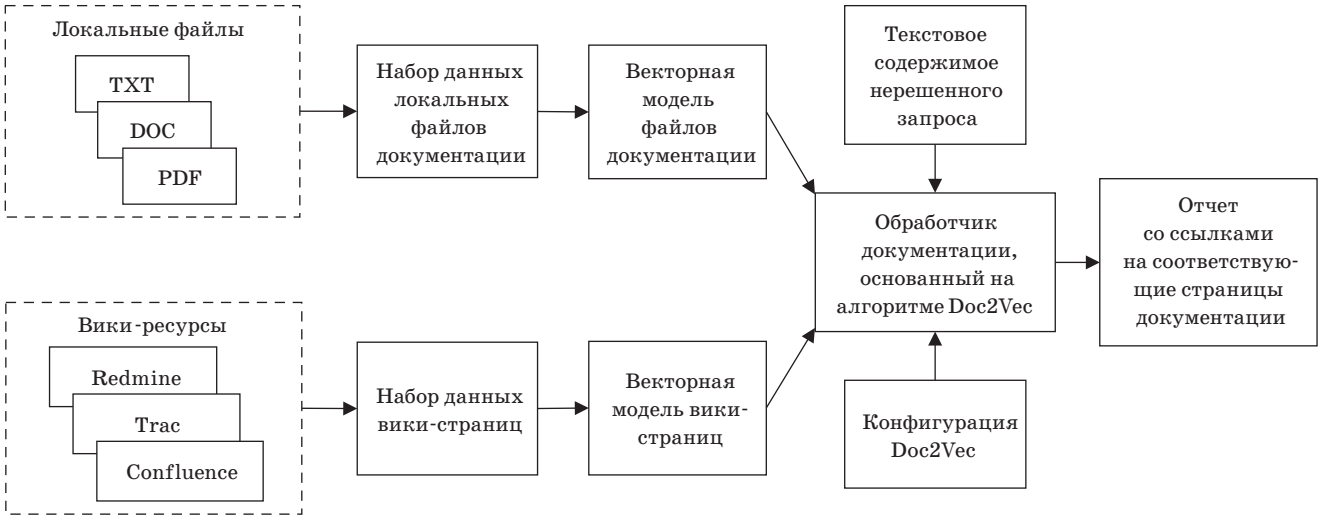
Предлагаемый подход

В данной работе предлагается снизить трудоемкость ручного анализа запросов заказчика за счет семантического поиска по документации. Суть ручного подхода состоит в том, чтобы получить список нерешенных запросов заказчика в системе отслеживания ошибок, а затем итеративно рассматривать и обрабатывать каждый запрос. Во время обработки запроса нередко приходится обращаться к документации программного продукта. Инженер-разработчик должен по ключевым словам искать конкретную информацию среди большого объема текстовых данных. Такой процесс неэффективен с точки зрения затрат времени, и его можно оптимизировать.

Концептуальная схема предлагаемого автоматизированного подхода к семантическому поиску по документации изображена на рис. 1.

На схеме видно, что предлагаемый подход к семантическому поиску по программной документации включает следующие этапы:

— создание наборов данных из локальных и удаленных ресурсов;



■ ***Рис. 1.*** Концептуальная схема предлагаемого подхода

■ ***Fig. 1.*** Conceptual schema of proposed approach

— обучение векторных моделей;

— применение алгоритма Doc2Vec к нерешенным запросам заказчика.

Существует два типа источников для создания наборов данных:

— локальные файлы документации различных форматов, таких как PDF, DOC (DOCX), RTF, TXT, PPT (PPTX) и т. д.;

— удаленные сетевые ресурсы, например вики-порталы (Confluence, Redmine, Trac и т. д.).

Структура набора данных следующая: каждая строка файла начинается с уникального идентификатора, который указывает на определенную страницу конкретного документа, затем идет разделяющий символ (например, «|»), а затем весь текстовый контент с конкретной страницы.

В дополнение к файлу набора данных создается файл метаданных. Он состоит из строк, которые разделены на три столбца: уникальный идентификатор страницы, затем имя документа, затем номер страницы в этом документе.

Файл метаданных необходим для того, чтобы после нахождения уникального идентификатора страницы было понятно, из какого документа и с какой страницы конкретный текст.

Затем для каждого набора данных создается векторная модель. После создания векторные модели используются алгоритмом Doc2Vec для поиска в документах страниц, которые семантически похожи на текстовое содержимое нерешенной проблемы.

В результате инструмент генерирует HTML-отчет, который содержит ссылки на конкретные страницы в локальных документах и на вики-сайтах.

Получив отчет с результатами, пользователь имеет возможность изучить соответствующие части документации, которые помогут быстро решить проблему заказчика. Нет необходимости искать нужную страницу в документе или в вики-системе.

Подводя итог, предлагаемый подход можно выразить формулой

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

где $U(x, y)$ — функция применения векторной модели к текстовым данным, в результате чего получается массив наиболее похожих страниц из документации; $L(x)$ — функция обучения, которая применяется на текстовых наборах данных и результатом работы которой является векторная модель; T — массив текстовых данных из документации; I_i — текстовые данные, описывающие конкретный запрос заказчика; $Rate(x)$ — число, которое выражается в процентах и означает семантическое сходство t_i и I_i; t_i — одна из похожих по смыслу найденных страниц документации; N — количество найденных похожих страниц документации.

Процедура нахождения похожих текстов

Семантическая близость двух текстов определяется косинусным сходством их векторов. Косинусное сходство [18] — это мера, которая измеряет косинус угла между двумя ненулевыми векторами. Данная метрика определяет расположение одного вектора в пространстве относительно другого вектора. Два вектора с одинаковой ориентацией в пространстве имеют косинус-сходство 1, а два вектора, располагающиеся под углом 90° относительно друг друга, имеют сходство 0. Два диаметрально противоположных вектора имеют сходство –1. Косинус-сходство в основном используется в положительном векторном пространстве, в котором результат ограничен в пределах [0, 1]. Единичные векторы максимально «схожи», если они параллельны, и максимально «несхожи», если они ортогональны (перпендикулярны).

Коэффициент схожести *similarity* вычисляется по формуле

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

где A_i и B_i — компоненты векторов **A** и **B** соответственно.

Мера косинусного сходства применима для векторных пространств с любым числом измерений. Данная мера наиболее часто используется в многомерных положительных пространствах.

При векторном представлении текстовых данных каждое слово или документ сопоставляется со своим уникальным вектором. Косинусное сходство между двумя векторами слов или документов показывает вероятность семантической схожести этих двух слов или документов.

В предлагаемом подходе автоматизированного анализа запросов заказчика в качестве меры схожести двух текстов предлагается использовать косинусное сходство. Будем считать, что два текста являются похожими по смыслу, если косинусное сходство их векторных представлений больше или равно коэффициенту *threshold*. Данный коэффициент можно настраивать. Его увеличение может быть обусловлено желанием находить меньшее количество максимально схожих текстовых документов. В то же время уменьшение данного коэффициента приведет к большему количеству результатов поиска среди семантически схожих документов. В ходе оптими-

зации гиперпараметров алгоритма Doc2Vec было вычислено оптимальное значение коэффициента *threshold*, которое составило 0,8.

Реализация программного инструмента

Разработанный инструмент, реализующий предложенный автоматизированный подход, написан на языке Java 8 и включает несколько модулей:

— два коннектора к удаленным веб-ресурсам (Jira, Confluence);

— два обработчика (обработчик локальной документации и Confluence);

— генератор HTML-отчета;

— исполнитель обработчиков, который координирует работу всей системы.

Архитектура инструмента показана на рис. 2.

Коннекторы к Jira и Confluence разработаны и использованы для загрузки данных с веб-ресурсов Jira и Confluence. Подключение к этим сервисам осуществляется с помощью соответствующих интерфейсов REST API. Возможна анонимная или базовая аутентификация с использованием имени пользователя и пароля.

Jira-коннектор получает текстовое описание нерешенных запросов от Jira. Коннектор Confluence получает текстовое представление вики-страниц.

Библиотека Jira Rest Java Client (JRJC) [https://bitbucket.org/atlassian/jira-rest-java-client] используется в коннекторе Jira. Эта Java-библиотека позволяет подключаться к любому экземпляру Jira 4.2+ с помощью REST API. В настоящее время JRJC предоставляет тонкий слой абстракции поверх REST API, а также объектную модель Jira на стороне клиента. Эти объекты представляют сущности запроса: *Issue*, *Priority*, *Resolution*, *Status*, *User* и т. д.

Библиотека Confluence Rest Java Client (CRJC) [https://docs.atlassian.com/atlassian-confluence/5.9.2/] используется в коннекторе Confluence. Как и JRJC, CRJC обеспечивает тонкий слой абстракции поверх REST API.

Каждый обработчик отвечает за три процесса:

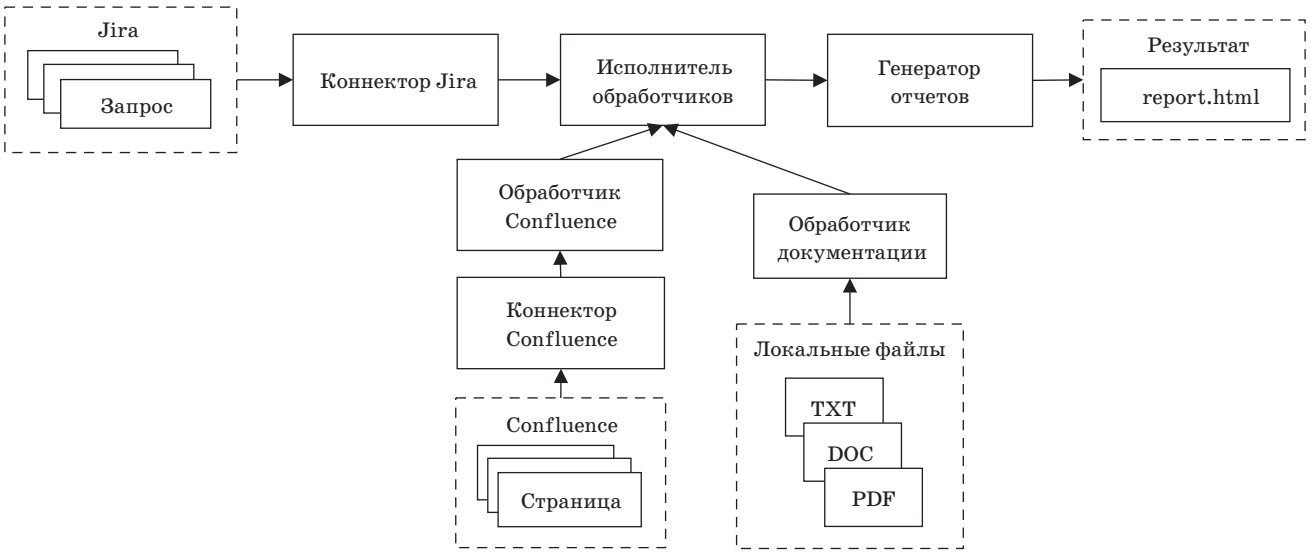
— создает структурированный набор данных из необработанных источников;

— создает векторную модель на основе полученного набора данных;

— использует созданную векторную модель для поиска страниц документации, которые связаны по смыслу с нерешенным запросом.

Обработчик документации использует локальные текстовые файлы для создания файла *Documentation_DataSet.txt*, а обработчик Confluence использует текстовое содержимое с вики-страниц для создания файла *Confluence_DataSet.txt*.

Структура набора данных для обработчика документации следующая: каждая строка файла начинается с уникального идентификатора, который указывает на конкретную страницу конкретного документа, затем идет символ разделительной вертикальной черты, а затем все текстовое содержимое с определенной страницы. Формат созданного файла набора данных можно увидеть на рис. 3, *а*. В дополнение к файлу набора данных создается файл метаданных (рис. 3, *б*).



■ ***Рис. 2.*** Архитектура программного инструмента

■ ***Fig. 2.*** The software tool architecture

а)
```
doc_1|Important Notice 2010 2020 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 Kafka Guide 8 Ideal Publish Subscribe
doc_3|Single Cluster Scenarios 26 Leader Positions 26 In Sync Replic
Using Kafka with Apache Flume 45 Using Kafka with Apache Spark Strea
doc_4|Kafka Administration 60 Kafka Administration Basics 60 Broker
kens 78 Delegation Token Basics 79 Broker Configuration Settings 79
doc_5|Kafka Performance Tuning 86 Tuning Brokers 86 Tuning Producers
doc_6|Appendix Apache License Version 2 0 182
doc_7|Apache Kafka Guide Apache Kafka is a streaming message platfor
it of Unlimited Lookback A new Subscriber A1 can read Publisher A s
igh availability but different semantics and message delivery guaran
doc_8|The next few sections provide an overview of some of the more
messages sent to the topics and serves consumer requests Apache Kafk
```

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

■ ***Рис. 3.*** Часть файла набора данных (*а*) и метаданных (*б*) локальной документации
■ ***Fig. 3.*** Part of the documentation data set (*а*) and metadata (*б*) file

Структура набора данных для Confluence аналогична набору данных для файлов документации, но отличие состоит в том, что уникальный идентификатор каждой строки напрямую указывает на конкретный идентификатор вики-страницы, и в этом случае нет необходимости создавать файл метаданных.

Основой обработчиков является алгоритм Doc2Vec, который реализован в Java-библиотеке Deeplearning4j [https://deeplearning4j.org/]. Работа с алгоритмом делится на три этапа: подготовка данных, обучение и использование.

До передачи содержимого **_DataSet.txt* файлов на вход алгоритма Doc2Vec необходимо подготовить текстовые данные для создания векторной модели. Процесс подготовки состоит из токенизации [19], стемминга [20] и удаления стоп-слов. Программный инструмент использует популярную реализацию стеммера Портера [20, 21] для языка Java.

Файлы наборов данных, сгенерированные обработчиками документации и содержащие предварительно обработанные тексты страниц, используются для обучения. В результате обучения получается векторная модель, которая впоследствии сериализуется в файлы *Documentation_VectorModel.zip* и *Confluence_VectorModel.zip* соответственно. Векторы представляют численное значение «смысла» запросов, и, используя математические операции над векторами, можно найти сходство между различными запросами.

Все настройки для обучения содержатся в конфигурационном файле *doc2vec.properties*. Этот файл содержит следующие поля: 1) *minWordFrequency* — минимальная частота слов в наборе обучающих данных; 2) *iterations* — количество итераций обучения, выполненных для каждой части тренировочного корпуса; 3) *epochs* — количество итераций (эпох) по всему учебному корпусу; 4) *layerSize* — размер выходных векторов; 5) *learningRate* — начальная скорость обучения модели; 6) *windowSize* — размер окна контекста; 7) *sampling* — численная характеристика подвыборки [22]; 8) *threshold* — пороговое значение косинусного сходства.

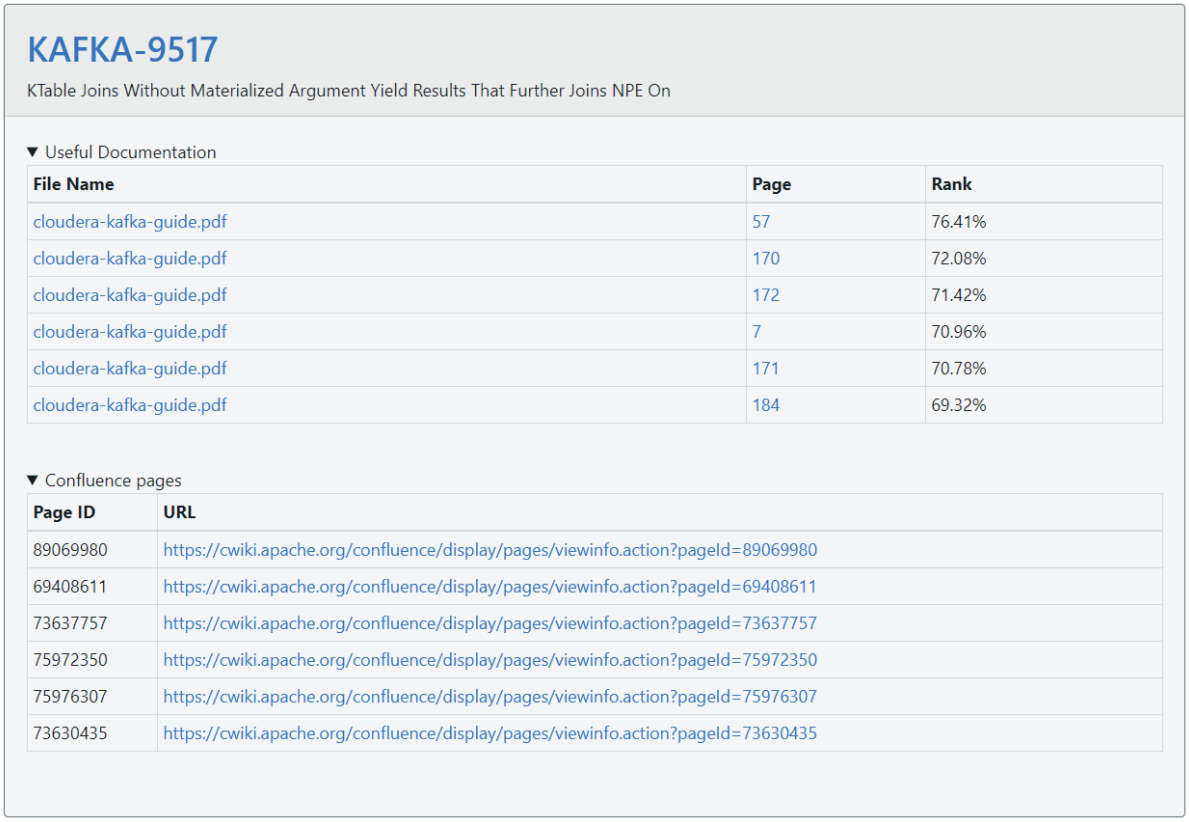
Архив *VectorModel.zip* содержит следующие файлы: *codes.txt* — коды для дерева Хаффмана [23]; *config.json* — настройки алгоритма Doc2Vec; *frequencies.txt* — метрики tf-idf [24] и bag-of-words [25]; *huffman.txt* — координаты точек дерева Хаффмана; *labels.txt* — список идентификаторов страниц документации в формате base64; *syn0.txt* — веса связей между входными и скрытыми слоями нейронной сети; *syn1.txt* — веса связей между скрытыми и выходными слоями нейронной сети.

После обучения алгоритмом Doc2Vec модель можно использовать для нахождения семантически близких векторов документов. Для этого в память загружается модель из файла **_VectorModel.zip*, в котором каждая страница представлена в виде числового вектора и связана с определенным идентификатором страницы в локальном документе или на вики-сайте. Затем текстовое содержимое неразрешенных запросов отправляется на вход алгоритма Doc2Vec.

Процесс поиска семантически связанной документации заключается в следующем. Сначала формируется числовой вектор из поступающего неразрешенного запроса. Затем этот вектор сравнивается с векторами страниц файлов документации или вики-страниц. В этом случае сходство текстовых данных определяется коэффициентом косинусной близости их векторных представлений. Чем больше значение коэффициента, тем более уверенно можно утверждать, что оба текста похожи.

Таким образом, результатом работы алгоритма является список с идентификаторами страниц документации, наиболее похожих на входные неразрешенные запросы. Все эти данные затем собираются в отчет и предоставляются пользователю (рис. 4).

Модуль исполнителя обработчиков координирует работу всех этих компонентов. Сначала он запускает коннектор Jira, получает список нерешенных запросов, а затем итеративно отправляет



KAFKA-9517

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

▼ Useful Documentation

File Name	Page	Rank
cloudera-kafka-guide.pdf	57	76.41%
cloudera-kafka-guide.pdf	170	72.08%
cloudera-kafka-guide.pdf	172	71.42%
cloudera-kafka-guide.pdf	7	70.96%
cloudera-kafka-guide.pdf	171	70.78%
cloudera-kafka-guide.pdf	184	69.32%

▼ Confluence pages

Page ID	URL
89069980	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=89069980
69408611	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=69408611
73637757	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=73637757
75972350	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=75972350
75976307	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=75976307
73630435	https://cwiki.apache.org/confluence/display/pages/viewinfo.action?pageId=73630435

■ ***Рис. 4.*** Отчет, содержащий ссылки на соответствующую входящему запросу документацию
■ ***Fig. 4.*** Final report with related documentation references

каждый запрос обоим обработчикам. Результаты обработки отправляются в генератор отчетов, который создает файл *report.html*.

Оценка точности модели Doc2Vec

Оптимизация гиперпараметров для обучающего алгоритма Doc2Vec проводилась методом случайного поиска (Random Search) с помощью программного пакета Deeplearning4j.

Для оптимизации сложных моделей (с более чем несколькими гиперпараметрами) случайный поиск превосходит по скорости и качеству другие классические методы, такие как поиск по решетке (Grid Search) и байесовскую оптимизацию [26].

Случайный поиск дополнительно сопровождался перекрестной проверкой на тренировочном наборе данных. Использование перекрестной проверки при оптимизации гиперпараметров позволяет произвести оценку эффективности выбранной модели с наиболее равномерным использованием имеющихся данных.

Тестовый набор данных представлен текстовым файлом, состоящим из множества строк. Нами рассматривается тестовый файл с количеством строк, равным 100. Каждая строка является тестовым шагом и разбита на три столбца специальным разделяющим символом. В первом столбце находится текстовый фрагмент исходного запроса. Во втором столбце находится текст близкого по смыслу запроса. А в третьем столбце, наоборот, находится текст запроса, который совершенно не связан с исходным.

Задача тестирования — определить, сколько строчек из 100 будут вычислены удачно. Вычисление заключается в следующем. Если алгоритм определяет, что первые два столбца похожи на более чем 80 %, а первый и третий столбец похожи менее чем на 20 %, то считаем, что данная строчка вычислена правильно. После вычисления всех строк суммируем количество правильных вычислений и делим на количество строк. Полученное число показывает точность векторной модели.

В результате тестирования и выбора гиперпараметров Doc2Vec выяснилось, что следующая конфигурация является оптимальной по качеству и времени выполнения алгоритма: *minWordFrequency* = 1; *iterations* = 5; *epochs* = 12; *layerSize* = 100; *learningRate* = 0,025; *windowSize* = = 5; *sampling* = 0, *threshold* = 0,8. При этом точ-

ность алгоритма на тестовом наборе данных составила 89,96 %.

Описание эксперимента

Разработанный инструмент был применен на проекте с открытым исходным кодом Apache Kafka [https://kafka.apache.org]. Любой инженер может найти ошибку в работе этой программы и зарегистрировать запрос, содержащий вопрос или описание ошибки, в соответствующей системе отслеживания ошибок Jira. Для эксперимента было получено 100 запросов из Jira.

Файл Apache Kafka Guide.pdf [https://docs.cloudera.com/documentation/enterprise/6/latest/PDF/cloudera-kafka.pdf] использовался в качестве локального файла документации. Confluence-пространство проекта Kafka [https://cwiki.apache.org/confluence/display/KAFKA] также использовалось в качестве удаленного вики-ресурса.

Инструмент развернут на платформе, которая является локальной рабочей станцией с операционной системой Windows 10 Enterprise x64, процессором Intel Core i7-4810MQ с тактовой частотой 2,80 ГГц и 16 ГБ оперативной памяти.

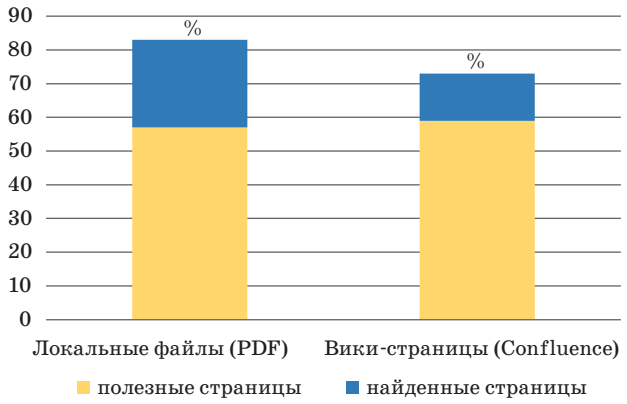
Файл *Documentation_DataSet.txt* (235 КБ) создан из PDF-файла документации. Количество строк в этом наборе данных равно количеству страниц в файле PDF и составляет 184 страницы. Количество строк в наборе данных *Confluence_DataSet.txt* (5,21 МБ) равно 765, что соответствует количеству загруженных вики-страниц. Затем из наборов данных были созданы векторные модели *Documentation_VectorModel.zip* (3,19 МБ) и *Confluence_VectorModel.zip* (35,5 МБ) соответственно. Создание наборов данных заняло 19 мин, а создание моделей — 12 мин.

В процессе работы над проектом Apache Kafka был проведен эксперимент, суть которого заключается в применении инструмента к нерешенным запросам и оценке его эффективности. Необходимо понять, в каком проценте случаев инструмент находит хотя бы одну соответствующую страницу в документации, а также в скольких случаях найденная страница является корректной и полезной для решения проблемы. Страница считается найденной, если семантическая близость между текстом запроса и текстом страницы составляет 80 %. В то же время найденная страница считается полезной, если она действительно может помочь в решении проблемы.

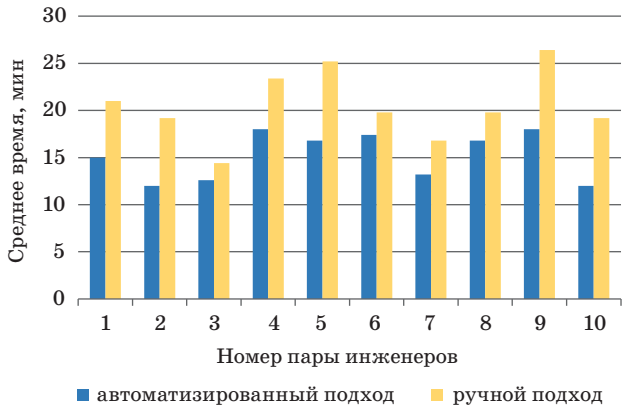
В рамках эксперимента было загружено и проанализировано 100 запросов из системы отслеживания ошибок Jira. Результаты анализа этих запросов представлены на рис. 5.

На диаграмме показано количество найденных и полезных страниц в документации. Были найдены 83 связанные страницы из локальных файлов, 57 из них оказались полезными для решения проблемы. Это означает, что точность алгоритма для локальных файлов составляет 69 %. Что касается вики-ресурсов, были найдены 73 связанные страницы, 59 из них оказались полезными для решения запроса. Точность в данном случае составляет 81 %.

Во второй части эксперимента сравнивалась эффективность ручного и автоматизированного подходов к поиску полезной документации. Для получения более уверенных результатов необходимо было провести этот эксперимент с большим количеством людей. Поэтому были приглашены 10 пар разработчиков для обработки 100 запросов. Во всех парах, участвовавших в эксперименте, оба инженера имели одинаковый опыт работы и квалификацию. Первый разработчик пытался



■ ***Рис. 5.*** Результаты эксперимента
■ ***Fig. 5.*** The experiment results



■ ***Рис. 6.*** Среднее время, затраченное на поиск полезной документации по одному запросу при ручном и автоматизированном подходе
■ ***Fig. 6.*** Average time spent to search for useful documentation for one request with the manual and automated approach

решить задачу вручную, а второй — в автоматизированном режиме.

Для сравнения времени поиска полезных страниц документации по каждому запросу использовался обычный таймер. Таймер включался, когда инженер начинал работу над новым запросом, и выключался, когда инженер находил хотя бы одну страницу документации, которая оказалась полезной для решения запроса. Результаты эксперимента показаны на рис. 6.

На диаграмме видно, что автоматизированный подход требует значительно меньше времени, чем ручной. Среднее время, затраченное на поиск подходящих страниц документации при ручном и автоматизированном подходе, составило 15,2 и 20,5 мин соответственно, т. е. снижение времени составило 25,9 %.

Заключение

В работе рассмотрены существующие исследования в области семантического поиска в текстовых документах.

Предложен автоматизированный подход для снижения трудоемкости обработки запросов заказчиков. Этот подход основан на использовании алгоритма машинного обучения Doc2Vec, который решает проблему семантического поиска в документации к программному продукту.

Созданный инструмент был успешно протестирован на проекте Apache Kafka — было проанализировано 100 запросов. Показаны эффективность и преимущества его использования. Среднее время анализа документации сократилось по сравнению с традиционным ручным подходом на 25,9 %.

Литература

1. **Shylesh S.** A study of software development life cycle process models. *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences*, 2017, pp. 534–541.
2. **Ogheneovo E. E.** On the relationship between software complexity and maintenance costs. *Journal of Computer and Communications*, 2014, vol. 2, no. 14, p. 1.
3. **Noei E., Zhang F., Wang S., Zou Y.** Towards prioritizing user-related issue reports of mobile applications. *Empirical Software Engineering*, 2019, vol. 24, no. 4, pp. 1964–1996.
4. **Filion L., Daviot N., Le Bel J., Gagnon M.** Using Atlassian tools for efficient requirements management: An industrial case study. *IEEE International Systems Conference*, April 2017, pp. 1–6.
5. **Pellegrini T.** Comparing SVM, Softmax, and shallow neural networks for eating condition classification. *16th Annual Conference of the International Speech Communication Association*, 2015, pp. 899–903.
6. **Maslova N., Potapov V.** Neural network Doc2Vec in automated sentiment analysis for short informal texts. *Lecture Notes in Computer Science*, 2017, vol. 10458, pp. 546–554.
7. **Kovalev A., Voinov N., Nikiforov I.** Using the Doc2Vec algorithm to detect semantically similar jira issues in the process of resolving customer requests. *Intelligent Distributed Computing XIII*, 2020, pp. 96–101. doi:10.1007/978-3-030-32258-8_11
8. **Aghajani E., Nagy C., Vega-Marquez O., Linares-Vasquez M., Moreno L., Bavota G., Lanza M.** Software documentation issues unveiled. *IEEE/ACM 41st International Conference on Software Engineering*, Montreal, QC, Canada, 2019, pp. 1199–1210.
9. **Liu M., Peng X., Meng X., Xu H., Xing S., Wang X., Liu Y., Lv G.** Source code based on-demand class documentation generation. *IEEE International Conference on Software Maintenance and Evolution*, Adelaide, Australia, 2020, pp. 864–865.
10. **Luciv D. V., Koznov D. V., Chernishev G. A., Terekhov A. N., Romanovsky K. Yu., Grigoriev D. A.** Detecting near duplicates in software documentation. *Program Comput Soft 44*, 2018, pp. 335–343.
11. **Калиниченко А. В.** Диалоговый метод автоматизации поиска семантически похожих документов. *Вестник ВГТУ*, Воронеж, 2012, т. 8, № 8, с. 15–17.
12. **Ensan F., Bagheri E.** Document retrieval model through semantic linking. *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, February 2017, pp. 181–190.
13. **Kassim J. M., Rahmany M.** Introduction to semantic-search engine. *International Conference on Electrical Engineering and Informatics,* August 2009, vol. 2, pp. 380–386.
14. **Wei Xing, Croft W.** LDA-based document models for Ad-hoc retrieval. *Proceedings of the 29th Annual International ACM SIGIR*, 2006, pp. 178–185. doi:10.1145/1148170.1148204
15. **Ai Wang, Yao Dong Li, Wei Wang.** Cross language information retrieval based on LDA. *IEEE International Conference on Intelligent Computing and Intelligent Systems*, Shanghai, 2009, pp. 485–490. doi:10.1109/ICICISYS.2009.5358121
16. **Wang S., Koopman R.** Semantic embedding for information retrieval. *BIR@ECIR*, 2017, pp. 122–132.
17. **Kurihara K., Shoji Y., Fujita S., Durst M. J.** 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*, 2019, pp. 463–467.
18. **Gunawan D., Sembiring C. A., Budiman M. A.** The implementation of cosine similarity to calculate text relevance between two documents. *Journal of Physics:*

Conference Series, March 2018, vol. 978, no. 1, article id. 012120.
19. **Ковалев А. Д., Никифоров И. В., Дробинцев П. Д.** Интеллектуальная обработка запросов заказчика на этапе сопровождения программного продукта. *Неделя науки СПбПУ: материалы научной конференции с международным участием*, Санкт-Петербург, 19–24 ноября 2018 г. СПб., 2019, с. 165–168.
20. **Farrar D., Hayes J. H.** A comparison of stemming techniques in tracing. *IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST)*, May 2019, pp. 37–44.
21. **Литвинов М. Б., Воинов Н. В.** Система интеллектуального анализа заявок пользователей телекоммуникационных услуг. *Современные технологии в теории и практике программирования: сборник материалов конференции*, Санкт-Петербург, 23 апреля 2020 г. СПб., 2020, с. 159.
22. **Ji S., Satish N., Li S., Dubey P. K.** Parallelizing word2vec in shared and distributed memory. *Transactions on Parallel and Distributed Systems*, 2019, vol. 30, no. 9, pp. 2090–2100.
23. **Rahman M., Hamada M.** Burrows – Wheeler transform based lossless text compression using keys and huffman coding. *Symmetry*, 2020, vol. 12, no. 10, p. 1654.
24. **Qaiser S., Ali R.** Text mining: use of TF-IDF to examine the relevance of words to documents. *International Journal of Computer Applications*, 2018, vol. 181, no. 1, pp. 25–29.
25. **Ayadi W., Elhamzi W., Charfi I., Atri M.** A hybrid feature extraction approach for brain MRI classification based on Bag-of-words. *Biomedical Signal Processing and Control*, 2019, vol. 48, pp. 144–152.
26. **Bergstra J., Bengio Y.** Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012, vol. 13, no. 10, pp. 281–305.

UDC 004.416
doi:10.31799/1684-8853-2021-1-17-27

Automated approach to semantic search through software documentation based on Doc2Vec algorithm

A. D. Kovalev[a], Post-Graduate Student, Assistant Professor, orcid.org/0000-0002-4610-5524, kov3000@ya.ru
I. V. Nikiforov[a], PhD, Tech., Associate Professor, orcid.org/0000-0002-2330-2197
P. D. Drobintsev[a], PhD, Tech., Associate Professor, orcid.org/0000-0003-1116-7765
[a]Peter the Great St.Petersburg Polytechnic University, 29, Polytechnicheskaya St., 195251, Saint-Petersburg, Russian Federation

Introduction: An important stage in a software development life cycle is the support phase, when customers can contact the support service of the supplier company and request a solution to an issue encountered in the software. To solve the request, engineers often have to refer to the relevant documentation. In order to reduce the complexity of the maintenance phase, the search for the necessary documentation pages can be automated. **Purpose:** Development of an approach to semantic search through documentation using Doc2Vec machine learning algorithm in order to automate the solution of customer requests. **Results:** An approach is proposed to semantic search through text documentation files and wiki pages using Doc2Vec machine learning algorithm. The documentation pages with semantic similarities to the textual description of an unresolved customer request help the engineer to process the request more efficiently and rapidly. Based on the proposed approach, a software tool has been developed which provides the engineer with a report containing links to documentation pages semantically related to the unresolved request. During the configuration of this tool, the optimal parameters of the Doc2Vec algorithm were found, providing the necessary quality of the semantic search. The idea of the experiment was to apply the tool to unresolved requests and evaluate its effectiveness. The developed approach and software tool were successfully tested in an open source Apache Kafka project. In the course of the experiment, 100 requests from Jira bug tracking system were downloaded and analyzed. The experimental results show the advantage of using the tool in software product support. The average documentation analysis time has been reduced as compared to the traditional manual approach. **Practical relevance:** The research results were used to solve real customer requests. The developed approach and the software implemented on its basis can reduce the complexity of the maintenance phase.

Keywords — software maintenance, automation, Doc2Vec, machine learning, semantic search, documentation.

For citation: Kovalev A. D., Nikiforov I. V., Drobintsev P. D. Automated approach to semantic search through software documentation based on Doc2Vec algorithm. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2021, no. 1, pp. 17–27 (In Russian). doi:10.31799/1684-8853-2021-1-17-27

References

1. Shylesh S. A study of software development life cycle process models. *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences*, 2017, pp. 534–541.
2. Ogheneovo E. E. On the relationship between software complexity and maintenance costs. *Journal of Computer and Communications*, 2014, vol. 2, no. 14, p. 1.
3. Noei E., Zhang F., Wang S., Zou Y. Towards prioritizing user-related issue reports of mobile applications. *Empirical Software Engineering*, 2019, vol. 24, no. 4, pp. 1964–1996.
4. Filion L., Daviot N., Le Bel J., Gagnon M. Using Atlassian tools for efficient requirements management: An industrial case study. *IEEE International Systems Conference*, April 2017, pp. 1–6.
5. Pellegrini T. Comparing SVM, Softmax, and shallow neural networks for eating condition classification. *16th Annual Conference of the International Speech Communication Association*, 2015, pp. 899–903.
6. Maslova N., Potapov V. Neural network Doc2Vec in automated sentiment analysis for short informal texts.

Lecture Notes in Computer Science, 2017, vol. 10458, pp. 546–554.
7. Kovalev A., Voinov N., Nikiforov I. Using the Doc2Vec algorithm to detect semantically similar jira issues in the process of resolving customer requests. *Intelligent Distributed Computing XIII*, 2020, pp. 96–101. doi:10.1007/978-3-030-32258-8_11
8. Aghajani E., Nagy C., Vega-Marquez O., Linares-Vasquez M., Moreno L., Bavota G., Lanza M. Software documentation issues unveiled. *IEEE/ACM 41st International Conference on Software Engineering*, Montreal, QC, Canada, 2019, pp. 1199–1210.
9. Liu M., Peng X., Meng X., Xu H., Xing S., Wang X., Liu Y., Lv G. Source code based on-demand class documentation generation. *IEEE International Conference on Software Maintenance and Evolution*, Adelaide, Australia, 2020, pp. 864–865.
10. Luciv D. V., Koznov D. V., Chernishev G. A., Terekhov A. N., Romanovsky K. Yu., Grigoriev D. A. Detecting near duplicates in software documentation. *Program Comput Soft 44*, 2018, pp. 335–343.
11. Kalinichenko A. V. A dialogue technique for automation semantically similar documents search. *Vestnik VGTU*, Voronezh, 2012, vol. 8, no. 8, pp. 15–17 (In Russian).
12. Ensan F., Bagheri E. Document retrieval model through semantic linking. *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, February 2017, pp. 181–190.
13. Kassim J. M., Rahmany M. Introduction to semanticsearch engine. *International Conference on Electrical Engineering and Informatics,* August 2009, vol. 2, pp. 380–386.
14. Wei Xing, Croft W. LDA-based document models for Ad-hoc retrieval. *Proceedings of the 29th Annual International ACM SIGIR*, 2006, pp. 178–185. doi:10.1145/1148170.1148204
15. Ai Wang, Yao Dong Li, Wei Wang. Cross language information retrieval based on LDA. *IEEE International Conference on Intelligent Computing and Intelligent Systems*, Shanghai, 2009, pp. 485–490. doi:10.1109/ICICISYS.2009.5358121
16. Wang S., Koopman R. Semantic embedding for information retrieval. *BIR@ECIR*, 2017, pp. 122–132.
17. Kurihara K., Shoji Y., Fujita S., Durst M. J. 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*, 2019, pp. 463–467.
18. Gunawan D., Sembiring C. A., Budiman M. A. The implementation of cosine similarity to calculate text relevance between two documents. *Journal of Physics: Conference Series*, March 2018, vol. 978, no. 1, article id. 012120.
19. Kovalev A. D., Nikiforov I. V., Drobintsev P. D. Intelligent processing of customer requests at the stage of software product maintenance. *Materialy nauchnoy konferentsii s mezhdunarodnym uchastiyem "Nedelya nauki SPbPU"* [Proc. Int. Conf. "SPbPU Science Week"], Saint-Petersburg, 2018, pp. 165–168 (In Russian).
20. Farrar D., Hayes J. H. A comparison of stemming techniques in tracing. *IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST)*, May 2019, pp. 37–44.
21. Litvinov M. B., Voinov N. V. System for intellectual analysis of requests from telecommunication services users. *Sbornik materialov konferentsii "Sovremennyye tekhnologii v teorii i praktike programmirovaniya"* [Proc. Conf. "Modern technologies in the theory and practice of programming"], Saint-Petersburg, 2020, p. 159 (In Russian).
22. Ji S., Satish N., Li S., Dubey P. K. Parallelizing Word2Vec in shared and distributed memory. *Transactions on Parallel and Distributed Systems*, 2019, vol. 30, no. 9, pp. 2090–2100.
23. Rahman M., Hamada M. Burrows – Wheeler transform based lossless text compression using keys and huffman coding. *Symmetry*, 2020, vol. 12, no. 10, p. 1654.
24. Qaiser S., Ali R. Text mining: use of TF-IDF to examine the relevance of words to documents. *International Journal of Computer Applications*, 2018, vol. 181, no. 1, pp. 25–29.
25. Ayadi W., Elhamzi W., Charfi I., Atri M. A hybrid feature extraction approach for brain MRI classification based on Bag-of-words. *Biomedical Signal Processing and Control*, 2019, vol. 48, pp. 144–152.
26. Bergstra J., Bengio Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012, vol. 13, no. 10, pp. 281–305.