Recommendation of Microservices Patterns Through Automatic Information Retrieval Using Problems Specified in Natural Language

Álex dos Santos Moura, Mário Alan de Oliveira Lima, Fabio Gomes Rocha, and Michel S. Soares(✉)

Federal University of Sergipe, São Cristòvão, Brazil
michel@dcomp.ufs.br

Abstract. Microservices are becoming increasingly popular for the development of distributed systems. However, in order to adopt microservices in practice, one has to deal especially with decomposition, communication and deployment of services. These issues can be solved by applying design patterns to microservices. Although it seems simple, developers may present difficulties when selecting the correct pattern to solve a given problem. This situation happens both to beginners and experienced professionals, as there are a considerable number of patterns to master. This study proposes an information retrieval-based approach to recommend patterns applied to microservices considering problems expressed by developers in natural language. To evaluate the proposed approach, we created two corpus, the first one containing 10 patterns, and the second 10 problems. When performing the experiments, we were able to achieve a precision rate of 60%. From this study, it is possible to support developers in selecting patterns applied to microservices, which facilitates their adoption and the development of higher quality systems, also benefiting organizations, as well as assisting researchers in future studies, presenting a possible way to recommend these patterns.

Keywords: Automatic recommendation · Design patterns · Microservices · Information retrieval · Machine learning

1 Introduction

The Microservice architecture, a variant of the Service-Oriented Architecture (SOA) [1–3], has been recently proposed with the objective of designing software systems as a collection of loosely-coupled services. Microservices are defined as independent, small, and limited context services, and together they constitute a software system [4].

Microservices have been increasingly adopted in large organisations such as Netflix, SoundCloud, and Amazon. Organisations are adopting microservices as

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13381, pp. 489–501, 2022.
https://doi.org/10.1007/978-3-031-10548-7_36

a solution for software architecture as they allow the development of systems that are easier to scale, deploy and maintain [5]. However, despite these advantages, processes and methods for adopting microservices are not simple, since it is necessary to deal with complex problems, including the decomposition of the system in microservices, how they will communicate, and, finally, the definition of the implementation plan for each microservice [6]. Even though microservices are often discussed, there is little guidance on refactoring processes of legacy applications [7]. Regarding guidance on adopting design patterns in microservices, there is a shortage number of works.

For many years, developers, according to their experiences, have produced proven solutions to common design problems in software projects [8]. These solutions are known as software design patterns. Usually, a design pattern is part of a collection, such as the popular collection presented by Gamma et al., in the book "Design Patterns: Elements of Reusable Object-Oriented Software" [9], which deals with problems related to object-oriented design and development. These patterns are also known in Software Engineering as the Gang of Four (GoF) Patterns, containing 23 patterns that are classified into 3 categories: creational, structural and behavioural.

Design patterns are also proposed for microservices. To solve the aforementioned problems and others present when adopting microservices, developers can apply the patterns pointed out by Richardson in his book named "Microservices Patterns: With Examples in Java" [10], where 42 patterns are presented and classified into 14 groups, such as decomposition, communication style, and deployment.

Although it seems simple, beginners and even experienced developers may present difficulties when choosing the appropriate pattern to solve a given problem. One important reason for these difficulties is the substantial amount of patterns to be mastered [11]. Therefore, it may be difficult to choose the most suitable design pattern for a given problem.

As a matter of fact, to help developers overcome this difficulty, and facilitate the adoption of microservices, this study proposes a recommendation approach that is based on information retrieval, where the developer can express a given problem in natural language and then receive the appropriate pattern to solve it. With this work, it is also possible to assist researchers in the development of future studies, since it presents an open approach to receiving improvements, thus, this study serves as a basis.

This paper contains four additional sections. Section 2, where related works are presented. Section 3 exposes the methodology, that is, the proposed approach, Sect. 4, where experiments and results are shown, and finally Sect. 5, which presents the conclusion and proposal for future work.

2 Related Works

Sanyawong and Nantajeewarawat [11] seek to improve the approach of recommending GoF (Gang of Four) patterns [9] that they developed in a previous work

[12]. Their approach is based on a pattern usage hierarchy, where the developer can identify the pattern to solve a given problem. After subjective evaluation, the authors concluded that it would be necessary to automate the levels of hierarchy to facilitate the selection of a specific pattern. Thus, they presented a way to automate the first level of the hierarchy, through textual classification, since this level associates the problem with a category of patterns. Therefore, they used algorithms such as Naive Bayes, J48, K-NN, and SVM. Also, they used 26 real problems to test the classifiers, and metrics like precision, recall, and F-measure to evaluate them. From the results, the authors concluded that it is possible to apply in practice this approach in projects in the software industry.

Hussain et al. [8], Duyne's HCI [13], Douglass [14] and Security [15] propose a recommended approach for GoF patterns, through information retrieval and the use of "Learning to Rank" to order the patterns according to their relevance for a given problem. The authors used three algorithms, Coordinate Ascent, AdaRank, and LambdaMART. The authors considered 47 real problems to test the approach, and two metrics to evaluate it, MAP (Mean Average Precision), and NDCG (Normalized Discounted Cumulative Gain). The results indicate that in terms of NDCG and MAP, LambdaMART outperforms the other algorithms. Thus, the authors concluded that the proposed approach is promising.

Silva-Rodriguez et al. [16] present a way of recommending interaction patterns to obtain satisfactory user interfaces aligned with user requirements. The authors use text classification, where a pattern is recommended according to a given requirement. Four algorithms were used, they are Logistic Regression, Multinomial Naive Bayes, Linear SVM, and Random Forest. Linear SVM proved to be superior to other algorithms in terms of Accuracy, Precision, Recall, and F1-Score.

Celikkan and Bozoklar [17] present a recommended approach for GoF design patterns that is based on the combination of 3 existing techniques in the information retrieval domain, they are text-based recommendation, case-based recommendation, and question-based recommendation. To evaluate the approach, the authors relied on 120 problems, where the text-based and case-based recommendation phases placed the appropriate pattern in first place for 56 problems, second place for 16, and third place for 8. This means that for 65% of the problems, the correct pattern was ranked among the top three, for 76% the correct pattern was among the top five, and for 86% the correct pattern was placed in the top 7. In the question-based recommendation phase, the answers of the most experienced developers caused a significant improvement in recommending the appropriate pattern.

Rahmati et al. [18] present a new method to select the GoF design patterns using information retrieval methods and cosine similarity. They proposed a more accurate method for automatically selecting GoF design patterns. The proposed method specifies the similarity between user-defined requirements and patterns definition text. This method is implemented based on the extended version of the vector space model (VSM), but the explicit semantic analysis (ESA) was considered a candidate to determine a similarity between the two texts.

The authors used Wikipedia as a broad external knowledge base. Two scenarios are considered to implement the ESA method: scenario 1, determining the semantic similarity between the user input text and the available texts in the repository of patterns; and scenario 2, determining the semantic similarity between the user input text and a set of keywords extracted from each available text in the repository of patterns. For evaluation, the proposed method used 23 design patterns, 29 object-oriented related design problems, and 9 real-world problems. The results of the first scenario were better than the second scenario, and between ESA method and the extended version of VSM, VSM has better performance. In addition, the authors found respectively 8.5%, 1.2%, and 5.2% improvement in terms of precision, recall, and accuracy of the proposed method compared to other methods.

Hamdy and Elsayed [19] proposed a novel approach for the automatic selection of the fit design pattern. Their approach is based on using the Latent Dirichlet Allocation (LDA) topic model. LDA was trained and used to analyze the corpus of pattern descriptions and extract the topics, then transferring each design pattern description and each problem scenario to a VSM of features, building a VSM of unigrams, the VSM is concatenated such that each pattern will be represented by a vector of features for each target design problem scenario, then discover the similarity between the target problem scenario, and the collection of patterns using Improved Sqrt-Cosine similarity measure (ISCS). The selected pattern is the closest to the problem scenario. The proposed approach was evaluated using a design patterns repository and a design problem repository. In the first, there is the textual description of the Gof design patterns and, in the second, there are 29 real design problem scenarios collected from various sources. The authors labelled each design problem manually, and the pre-processing was performed using the natural language toolkit NLTK. The best precision obtained during experiments is about 72% with 10 topics. The results showed that the proposed approach outperforms approaches based on the VSM.

Different from the works previously presented, this study deals with the recommendation of patterns applied to microservices. Thus, we use information retrieval, since the effectiveness of classification-based systems depends on a large and representative dataset [20], which is not easy, as we would have to elaborate this on the dataset, since we did not find any, in addition to having to use several classifiers to obtain good results, according to the works described before.

3 Methodology

In this section, the suggested approach is presented as an answer to the following research question: How to Recommend Microservices Patterns through Automatic Information Retrieval using Problems Specified in Natural Language?

It is worth mentioning that the patterns used were collected from Richardson's popular book called "Microservices Patterns: With Examples in Java" [10] and the design problems were created by the authors, as we could not

find databases with similar problems in real systems based on microservices. In Sect. 4, the data used are commented in detail.

The proposed approach is organized into 3 steps: textual pre-processing, the weighting of terms, and similarity calculation. These steps are depicted in Fig. 1 and are described in the first three subsections as follows.

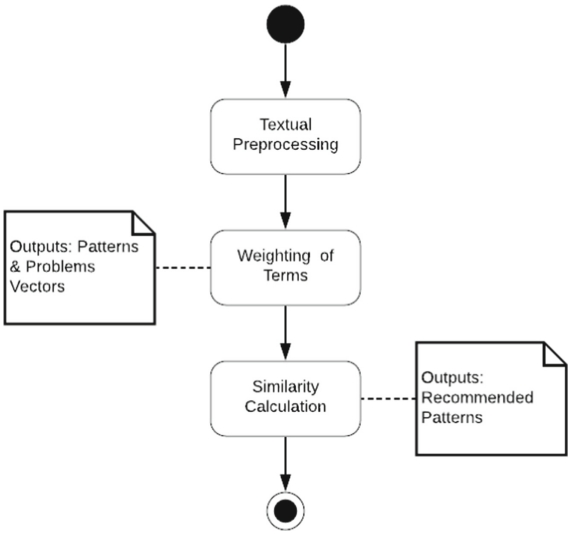

Fig. 1. Steps of the proposed approach

3.1 Textual Pre-processing

This step intends' is to prepare data for use. The three main tasks performed in it are depicted in Fig. 2 and are explained below.

In this work, a token is a word. Tokenization consists of extracting tokens from a text. Stop-word Removal corresponds to the elimination of words that are not relevant, such as articles and pronouns. Finally, stemming is the reduction of an inflected word in its stem. In this task, we apply the algorithm presented by Porter [21].

3.2 Weighting of Terms

In this step, pre-processed patterns and problems are represented by vectors of words in a vector space model. The dimensions of these vectors correspond to different words or terms. To use vector operations, we assign a weight to each term to compare patterns with problems.

$$TF * IDF(t, d, D) = TF(t, d) \cdot IDF(t, D) \quad (1)$$

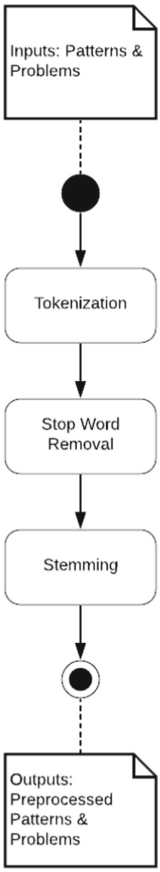

Fig. 2. Main tasks of text pre-processing

We calculate this weight through Eq. 1, which consists of the product of two statistics, named TF (Term Frequency) and IDF (Inverse Document Frequency), where TF(t,d) is the number of times a term (t) appears in a document (d), and IDF(t,D), calculated through Eq. 2, is equivalent to the inverse of DF(t,D), which stands for Document Frequency and indicates the number of documents that have a term (t) in a corpus (D).

$$IDF(t, D) = \frac{1}{DF(t, D)} \tag{2}$$

Thus, the resulting value of Eq. 1 is known as TF*IDF (Term Frequency-Inverse Document Frequency), its purpose is to indicate how important a term is for a document in a corpus. In this work, a document can be a pattern or a problem.

3.3 Similarity Calculation

This is the last step of the presented approach, where we apply a similarity measure that expresses the distance between two vectors. Thus, the appropriate pattern to solve a given problem is the one that has its vector closest to the problem vector.

Because of this, we employ the cosine similarity measure, given that it is popular in the field of information retrieval and is given by Eq. 3, as follows:

$$CS(V,W) = \frac{\sum_{i=1}^{n} V_i W_i}{\sqrt{\sum_{i=1}^{n} V_i^2}\sqrt{\sum_{i=1}^{n} W_i^2}} \tag{3}$$

V is the pattern vector, W is the problem vector, and n is the size of the vectors.

3.4 Evaluation Metric

To evaluate the suggested approach, we use a metric called Precision. It indicates the rate of recommendations that were performed correctly. This metric is obtained from Eq. 4, where TP and FP represent, respectively, the total number of recommendations made correctly and incorrectly.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

4 Implementation and Results

In this section, the implementation of the proposed approach is presented throughout the following Subsect. 4.1, 4.2 and 4.3. Also, in this section, the results of the recommendations made are reported, in Subsect. 4.4, where, in addition, in this same subsection, some ways to improve our approach are described.

4.1 Data About Design Patterns

The design patterns used in this study are aimed at software developed using the microservices architecture, because as seen in Sect. 1, this work intends is to facilitate the selection of these patterns so that developers can apply them properly, which makes these professionals more productive, in addition to favouring the quality of this software.

Thus, we have selected 10 of these patterns from Richardson's popular book called "Microservices Patterns: With examples in Java" [10]. From there, we look for pertinent information about each one of these patterns, this information added together makes up what we call a "description", in this way, we form the description of each pattern taken from the book.

Table 1 presents the information used to create the description of a pattern. In total, there are 11 pieces of information. After elaborating the descriptions

Table 1. Information used to create the description of a pattern

Information	Definition
Name	Term by which the pattern is called
Group	Group in which the pattern is a part
Subgroup	Subgroup in which the pattern is a part
Layer	Layer where the pattern is applied
Context	Context in which the pattern is applied
Problem	Problem the pattern solves
Forces	Issues that must be addressed when solving a problem in a given context
Solution	Way in which the pattern solves the problem
Resulting context	Benefits, drawbacks and issues, arising from the application of the pattern
Issues	Issues that must be addressed to use the pattern
Related patterns	Patterns that have some relationship with the pattern

of the patterns, we generate a corpus with them. This corpus is retrieved when a pattern needs to be recommended to solve a design problem. A detail worth mentioning is that some information that constitutes the description of a pattern was extracted from Richardson's website [22].

4.2 Data About Design Problems

In this work, design problems are difficult scenarios that can occur during the development or maintenance of a system based on the microservices architecture. They are important for the evaluation of our approach, which is given by the metric called "precision", that has already been presented in Subsect. 3.4. In short, the more problems that have adequate recommendations, the greater the precision of our approach. This means something positive, as it indicates how effective the approach is.

Because of this, we created 10 design problems that are part of a corpus, each of these problems is labelled with the correct pattern to solve it. We did this to calculate the precision later, here are 5 of these 10 problems as examples:

Problem #1: A company that develops software systems received a new demand, where it is necessary to develop a clinical management system. Because of this, requirements analysts performed the requirements elicitation, and from the given set of requirements, software architects concluded that the microservices architecture would be the most suitable for the system. Then, systems analysts created a top-level domain model, identified command and query operations, and finally mapped business capabilities with the help of requirements analysts. Now systems analysts want to define system services. These services must be independent, small, and contextually limited.

Problem #2: OnShop is a monolithic software system that will be migrated to microservices. Professionals participating in this migration want to use Domain-Driven Design (DDD) concepts to decompose the system into services, as they are familiar with DDD.

Problem #3: The fictional company XGO is designing a microservices architecture for its main system and it needs to make the microservices communicate. This communication must be request/response based and synchronous.

Problem #4: After making the services communicate through REST, it was observed that when the mobile app makes multiple requests to the order service, through an API gateway, and the order service is down or responding to the requests extremely slowly, the API gateway is unable to serve new requests.

Problem #5: Instances of ordering, payment and other services need to make their network address available in some way, as customers need this information to make requests.

Table 2. Example of problems and the correct pattern to solve each one

Problem #	Correct pattern
1	Decompose by business capability
2	Decompose by subdomain
3	Remote procedure invocation
4	Circuit breaker
5	Self registration

In Table 2, the correct pattern to solve each exemplified problem is presented. An issue worth mentioning is that we did not find in the literature any set of design problems that are aimed at software based on the microservices architecture, for this reason, we had to create a corpus with these problems.

4.3 Tools Used to Develop the Approach

To implement the proposed approach of recommending design patterns used in software based on the microservices architecture, we use Python [23], NLTK (Natural Language Toolkit) [24], Scikit-Learn [25] and Collaboratory.

The Python programming language was chosen for being simple, prioritising code readability with its syntax that is clear and concise, having powerful native resources, libraries and frameworks and, finally, it is familiar to the researchers.

Thus, we implemented our approach in the Collaboratory, which is a free cloud service offered by Google. Collaboratory makes it possible to work with Python in a collaborative environment, in this way, the researchers were able to work together.

NLTK is a leading platform for developing Python programs that need to handle human language data. Given this, we use NLTK in the description of

each pattern and in the design problems, where we extract the tokens, remove the stop words and reduce each inflected word to its stem. These processes have already been detailed in Subsect. 3.1.

To weight the words and calculate how similar each design problem is to each design pattern, we made use of the Scikit-Learn library, as it is easy to use, efficient, and quite popular when it comes to machine learning. It is worth noting that the weighting of words and the calculation of similarity between design problems and design patterns have already been addressed, respectively, in Subsect. 3.2 and 3.3.

4.4 Recommended Patterns for Design Problems

With the corpus ready and our approach implemented, we put it into action and obtained several recommended patterns for each problem, where these patterns were ordered in descending order by the similarity rate given by Eq.3. Thus, we considered only the first 3 recommended patterns for each problem, Table 3 sets out the first 3 recommended patterns for design problem #1.

Table 3. The first 3 recommended patterns for design problem #1

Placing	Recommended pattern	Similarity rate
1st	Decompose by business capability	0.28283
2nd	Decompose by subdomain	0.24633
3rd	Server-side discovery	0.11662

Table 4. Recommended patterns for each problem given as an example in Subsect. 4.2

Problem #	1st recommended pattern	2nd recommended pattern	3rd recommended pattern
1	Decompose by business capability	Decompose by subdomain	Server-side discovery
2	Decompose by subdomain	Decompose by business capability	Remote procedure invocation
3	Remote procedure invocation	Messaging	Decompose by subdomain
4	Remote procedure invocation	Client-side discovery	Server-side discovery
5	Server-side discovery	Remote procedure invocation	Client-side discovery

In view of this, Table 4 shows the recommended patterns for each problem given as an example in Subsect. 4.2. The first recommended pattern is considered the most suitable candidate to solve the problem. In this way, it is possible to confirm this information by checking if it is equal to the correct pattern to solve the problem. Table 2, previously presented, shows the correct pattern to solve each of the 5 problems presented as examples in Subsect. 4.2.

Thus, out of 10 problems, 4 had the correct pattern recommended in first place, 2 in second, and the other 4 below the third. This means that the precision of the proposed approach is equivalent to 40% and that 20% of the problems had the correct pattern recommended in second place, which indicates that the correct pattern for 60% of the problems happens in first and second place.

In search of the reasons that contributed to 4 of 10 problems having their correct pattern recommended below third place, we reviewed these problems. With that, we found that they are problems related to microservices communication, and that the first 3 patterns recommended for each one are patterns that solve problems related to microservices communication, that is, this happened because these patterns share similar terms, and because the problems did not have more specific terms for each pattern, ended up impacting the recommendation.

5 Conclusion and Future Work

This study proposes an approach for recommending design patterns used in microservices-based systems to solve common problems that occur in systems of this type. It is based on information retrieval, where a developer can report a problem in text and then get a pattern that could solve it. To evaluate the proposed approach, we created two corpora, the first one containing 10 patterns, and the second 10 problems. Its precision is equal to 60%.

When developing using the microservices architecture, the adoption of standards is facilitated, providing a higher quality of source code development. Developers who make migrations and new implementations of microservices-based systems can benefit from recommendations that will help them solve problems that may occur during the process of developing these systems. Organizations can also benefit in two ways, by reducing problems in adopting patterns, and by improving the software process. Finally, this work may contribute to future research, presenting a possible way to recommend these patterns.

For future work, the intention is to evolve the two corpora generated in this study, that is, to enrich them with more data, and look for ways to improve the approach, such as replacing the current similarity measure with ISCS (Improved Sqrt-Cosine Similarity) [26], which seems to be better when dealing with smaller documents.

References

1. França, J.M.S., Soares, M.S.: SOAQM: quality model for SOA applications based on ISO 25010. In: Proceedings of the 17th International Conference on Enterprise Information, pp. 60–70 (2015)

2. Soares, M.S., França, J.M.S.: Characterization of the application of service-oriented design principles in practice: a systematic literature review. J. Softw. **11**(4), 403–417 (2016)
3. Niknejad, N., Che Hussin, A.R., Prasetyo, Y.A., Ghani, I., Fajrillah, A.A.N.: Service oriented architecture adoption: a systematic review. Int. J. Integr. Eng. **10**(6) (2018)
4. Dragoni, N., et al.: Microservices: yesterday, today, and tomorrow. In: Present and Ulterior Software Engineering, pp. 195–216. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67425-4_12
5. Chen, R., Li, S., Li, Z.: From monolith to microservices: a dataflow-driven approach. In: 2017 24th Asia-Pacific Software Engineering Conference (APSEC), pp. 466–475 (2017)
6. Balalaie, A., Heydarnoori, A., Jamshidi, P.: Migrating to cloud-native architectures using microservices: an experience report. In: Celesti, A., Leitner, P. (eds.) ESOCC Workshops 2015. CCIS, vol. 567, pp. 201–215. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33313-7_15
7. Bogner, J., Fritzsch, J., Wagner, S., Zimmermann, A.: Industry practices and challenges for the evolvability assurance of microservices. Empir. Softw. Eng. **26**(5), 104 (2021)
8. Hussain, S., et al.: A methodology to rank the design patterns on the base of text relevancy. Soft. Comput. **23**(24), 13433–13448 (2019)
9. Gamma, E., Helm, R., Johnson, R., Johnson, R.E., Vlissides, J.: Design Pattern: Elements of Reusable Object-Oriented Software. Addison-Wesley, NewYork (1995)
10. Richardson, C.: Microservices Patterns: With Examples in Java. Simon and Schuster (2018)
11. Sanyawong, N., Nantajeewarawat, E.: Design Pattern Recommendation: A Text Classification Approach (2015)
12. Sanyawong, N., Nantajeewarawat, E.: Classifying design-level requirements using machine learning for a recommender of interaction design patterns. In: IEEE, pp. 134–139 (2014)
13. Landay, J.A., Hong, J.I.: The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience. Addison-Wesley Professional, Reading (2003)
14. Douglass, B.P.: Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Addison-Wesley Professional, Reading (2003)
15. Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns: Integrating Security and Systems Engineering. John Wiley & Sons, Hoboken (2013)
16. Silva-Rodríguez, V., Nava-Muñoz, S.E., Castro, L.A., Martínez-Pérez, F.E., Pérez-González, H.G., Torres-Reyes, F.: Classifying design-level requirements using machine learning for a recommender of interaction design patterns. IET Softw. **14**(5), 544–552 (2020)
17. Celikkan, U.; Bozoklar, D.: A consolidated approach for design pattern recommendation. In: 2019 4th International Conference on Computer Science and Engineering (UBMK). IEEE, pp. 1–6 (2019)
18. Rahmati, R., Rasoolzadegan, A., Dehkordy, D.T.: An automated method for selecting GoF design patterns. In: 9th International Conference on Computer and Knowledge Engineering (ICCKE 2019), 24–25 October 2019, Ferdowsi University of Mashhad (2019)
19. Hamdy, A., Elsayed, M.: Topic modelling for automatic selection of software design patterns. In: ICGDA 2018, 20–22 April 018, Prague, Czech Republic (2018)

20. Uysal, A.K.: An improved global feature selection scheme for text classification. Expert Syst. Appl. **43**, 82–92 (2016)
21. Jones, K.S., Willett, S. (eds.): Readings in Information Retrieval. Morgan Kaufmann, San Francisco (1997)
22. Microservices Homepage, https://microservices.io. Aaccessed 16 Mar 2022
23. Python Homepage, https://www.python.org. Accessed 16 Mar 2022
24. NLTK Homepage, https://www.nltk.org. Accessed 16 Mar 2022
25. Scikit-Learn Homepage, https://scikit-learn.org. Accessed 16 Mar 2022
26. Sohangir, S., Wang, D.: Improved sqrt-cosine similarity measurement. J. Big Data **4**(1), 1–13 (2017). https://doi.org/10.1186/s40537-017-0083-6