UDC 004.415.2
DOI https://doi.org/10.32838/2663-5941/2020.4/15

Oleshchenko L.M.
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

Lysenko O.O.
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

SOFTWARE METHOD FOR CLUSTERING SOFTWARE TESTING REPORTS USING KNN ALGORITHM

The article is devoted to the development and software implementation of a method for analyzing the results of software testing using the KNN algorithm. Software developers are trying to avoid manual product testing in order to reduce the risk of human factors and reduce the cost of testing. The average error analysis time after testing for 500 "failed" tests can be up to 24 hours. IT companies need automation of error analysis after software testing to provide the customer with a complete understanding of what's going on in the project with a detailed description of what is already working and needs refinement. This is very difficult to do manually, especially when the project is very large and runs more than a thousand tests. There is a need to develop a software system for automated analysis of software test results, which will reduce the time of error analysis and will allow view the status of tests in real time. Existing machine learning methods that can be used to analyze software test results are analyzed. The choice of the most appropriate method of machine learning for this task, namely k-nearest neighbors (KNN), is grounded. On the basis of KNN a web application is created for automatic analysis of software testing results. The architecture of the software system is implemented in the form of microservices. The basic idea of the proposed method is that the stack path is used as the data to be clustered, this part contains information about the cause of the "drop" of the test. The proposed method makes it possible to automate the error analysis process, reducing the time and amount of human resources required to analyze software test results. The article compares the results of clustering methods of KNN, Support Vector, and Naive Bayes to automatically analyze software test results for 100, 200, 500, and 1000 failed tests. The automatic analysis of results is compared with the manual one. When studying the results of comparing automatic error analysis of software with 1000 failed tests, KNN method shows an accuracy of 0.982, which is the most accurate result among the clustering methods considered. When analyzing 150 failed tests, the algorithm showed a result 12 times faster than manual analysis. Thus, it is shown, that the proposed software method is effective.

Key words: *software testing, machine learning methods, test case, stack trace, clustering, KNN algorithm, Elasticsearch, TF, IDF.*

Problem statement. Testing is one of the key stages of software development. Analyzing developed software requires first-class testers, their constant training and motivation, which is a big problem for small and medium-sized companies. Software developers are trying to avoid manual product testing in order to reduce the risk of human factors and reduce the cost of testing. The average error analysis time after auto-testing for 500 "failed" tests can be up to 24 hours. IT companies need automation of error analysis after software testing to provide the customer with a complete picture of what's going on in the project with a detailed description of what is already working and needs refinement. This is very difficult to do manually, especially when the project is very large and runs more than a thousand tests. There is a need to develop a software system for automated analysis of software test results, which will reduce the time of error analysis and will allow view the status of tests in real time.

Related research. In previous works clustering methods: Naive Bayes method, Support Vector Ma-chines method and k-nearest neighbors (KNN) algorithm are analyzed [1]. This study shows that Naive Bayes and Support Vector Machines methods are complex to implement, have less accuracy for large datasets than kNN, but have a higher clustering rate [2; 3].

The main goal of the article is to automate the process of analyzing the results of software testing, which means reducing human intervention in this process, the main requirement for the selection of the algorithm selected accuracy. The running time of the algorithm was not considered to be the main metric for selecting the algorithm, since using any of these algorithms in general will greatly reduce the time for testing analysis. For this reason, the KNN algorithm was chosen as the basis for its simplest implementation and highest accuracy on large volumes of data. The first step of the algorithm is to specify the number *k* of nearest neighbors.

Presentation of the main research material. In the second step, there are k entries with a minimum distance to the feature vector of the new object (neighbor search). The distance calculation function must comply with the following rules: $d(x, y) \geq 0$, $d(x, y) = 0$ if and only if $x = y$; $d(x, y) = d(y, x)$; $d(x, z) \leq d(x, y) + d(y, z)$, provided that the points x, y, z do not lie on a straight line, where x, y, z are the vectors of the signs of the objects being compared.

For ordered attribute values, the Euclidean distance is used [4]:

$$D_E = \sqrt{\sum_i^n (x_i - y_i)^2}\,, \tag{1}$$

where n is the number of attributes.

For row variables that cannot be ordered, the difference function can be applied [4]:

$$dd(x,y) = \begin{cases} 0, x = y; \\ 1, x \neq y. \end{cases} \tag{2}$$

When finding the distance, they take into account the importance of attributes, which is determined subjectively by the expert or analyst, relying on their own experience. In this case, when finding the distance, each *i*-th square of the difference in the sum is multiplied by a factor Z_i.

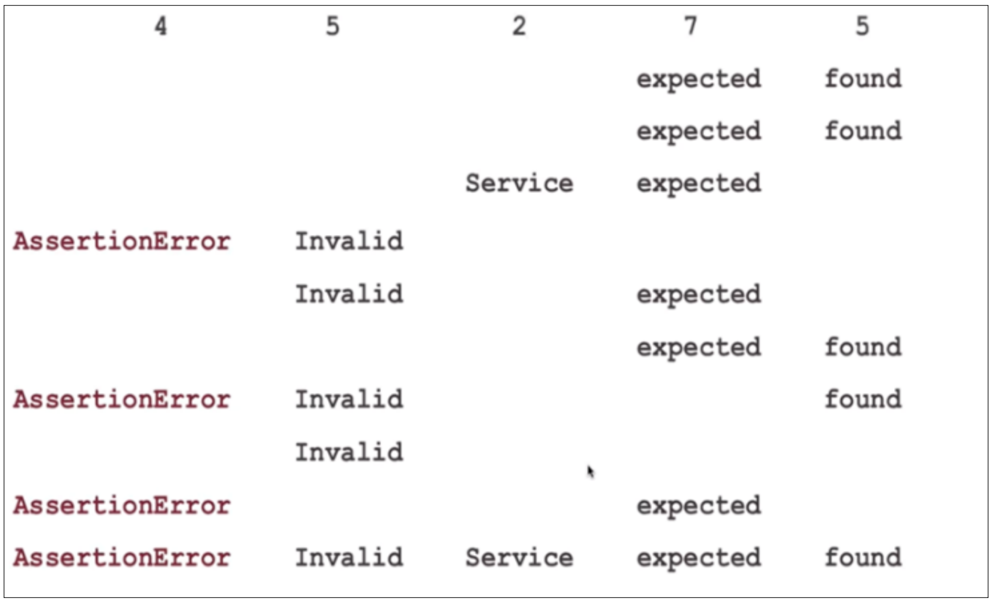
Using KNN algorithm for clustering reports of software testing

As the results of automated tests will be analyzed, the result of performing these tests will be a stack trace, which will contain messages about what is wrong in the test [5]. In this research the results of the Junit library for Java are used.

The first line of the stack track reports is the reason for the "drop" of the test. Lines of stack trace show the files in which the error occurred, followed by errors that pop up at all levels of the abstraction library used, and are usually identical in all automated tests. Since the bulk of this test is identical for each test, it is pointless to submit the entire stack path to the algorithm input, so this will not bring the desired result, so the first step would be to delete this duplicate text. Thereafter, basic information will remain that is unique to each test result. However, this will also not be enough, since there are still "noises" that need to be removed. The next step is to delete the date, which will also prevent clustering, then lowering the entire text to lowercase and removing punctuation marks. Then we get a text that can be worked with.

After a lot of repetitive code is gathered, we need to collect frequency pointers, that is, how often specific words are repeated in a certain category (Fig. 1). The more tests, the more text that can be worked with will be collected and thus the accuracy of the algorithm will increase.

Once we have all these metrics, we move on to the frequency response, this is the part where machine learning happens and is called the TF-IDF metric. This metric shows how often this word appears in



4	5	2	7	5
			expected	found
			expected	found
		Service	expected	
AssertionError	Invalid			
	Invalid		expected	
			expected	found
AssertionError	Invalid			found
	Invalid			
AssertionError			expected	
AssertionError	Invalid	Service	expected	found

Fig. 1. Frequency error indicators

a document and how important it is to all documents in the library.

TF (term frequency) is the ratio of the number of occurrences of the selected word to the total number of words in the document. Thus, the importance of the word within the selected document is evaluated.

$$TF = \frac{n_i}{\sum_k n_k}, \qquad (3)$$

where n_i is the number of occurrences of the word in the document and the denominator is the total number of words in the document.

IDF (inverse document frequency) is the inversion of the frequency with which a word occurs in a collection document. Using IDF reduces the weight of commonly used words.

$$IDF = \log \frac{|D|}{|d_i \supset t_i|}, \qquad (4)$$

where |D| is number of collection documents; $|d_i \supset t_i|$ is the number of documents in which the word occurs t_i (when $n_i \uparrow 0$).

The choice of the basis of the logarithm in the formula (4) is irrelevant, since changing the basis will change the weight of each word by a constant factor, the weight ratio will remain unchanged. TF-IDF is the product of two factors: TF and IDF. This frequency response will help determine how important a word is for a particular

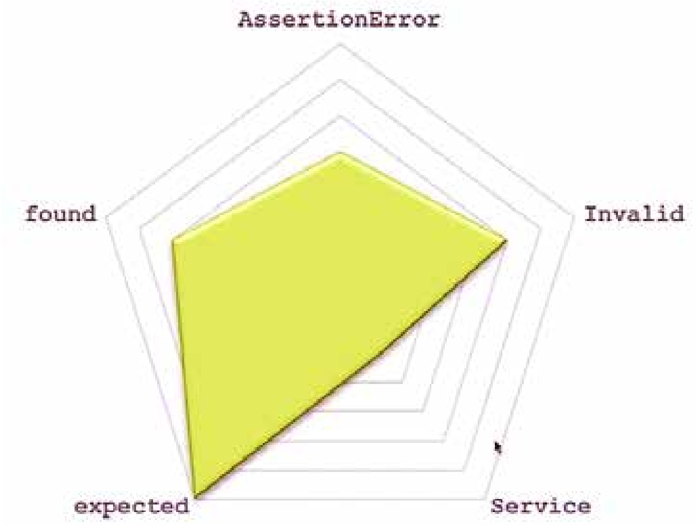


Fig. 2. Graphical representation of the TF-IDF metric

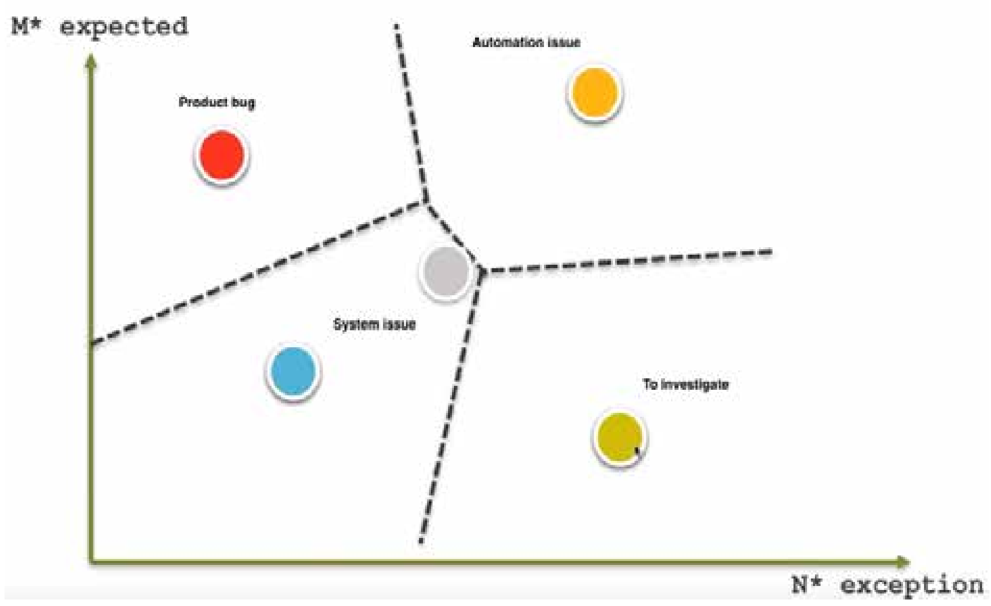


Fig. 3. Two-dimensional clustering of test results

document and how important it is in categorizing a particular type of document.

We have all of these words that belong to different categories of errors, such as Product Bug, System Issue, or Automation Issue, and this categorization will be done by a person at the beginning so that machine learning understands this document, that is, this stature, refers to specific reason for the "failure" of the test.

This information will be presented in vector form. Analyzing text with machine learning algorithms means converting all this large text into a mathematical vector and this vector will be located in a multidimensional space where each space will refer to a specific word.

Software implementation of the proposed method

The developed software system enables:

– GitHub sign-up and sign-in;

– view real-time test execution;

– automatic clustering of test results;

– specify categories and subcategories for failed tests;

– save up to 10 test runs;

– merge multiple launches into one;

– attach images, pieces of code to a specific test result;

– edit failed test categories and subcategories;

– filter test results by categories and subcategories;

– add, modify, delete a description of the test results;

– adjust parameters for auto-analysis;

– display the results graphically.

The architecture of the software system is implemented in the form of microservices. The whole system is divided into client and server parts. The client part includes services such as Logger, Agent, Client. Client are API integrations. HTTP clients that process HTTP request sending. Agent is a framework integration. Special reporters / listeners who monitor test events and cause events to be transmitted through the client. Logger is an integration of logs that helps to collect logs, associate it with a test code through an agent, and send it to a server through a client.

Gateway is the main point of entry for application services. This service is responsible for routing requests for proper maintenance and load balancing. The gateway contacts the service registry to get a list of the actual services that are allowed to route traffic.

Registry is a tool that stores the actual list of running services with added meta information. It checks the status of each running service to ensure availability.

The API is responsible for handling agent inbound requests and the user interface.

Authorization is a module that authorizes users and creates / revokes user tokens. It supports various types of authentication mechanisms:

– Basic Auth;

– GitHub Auth (OAuth2);

– LDAP Auth.

LDAP Authis an OAuth2 server that authorizes the user using the mechanisms mentioned below and creates an OAuth2 internal token that is used by the user interface and the agent agent. There are two types of tokens:

– UI (ending token);

– API is an endless token designed for use on the agent side.

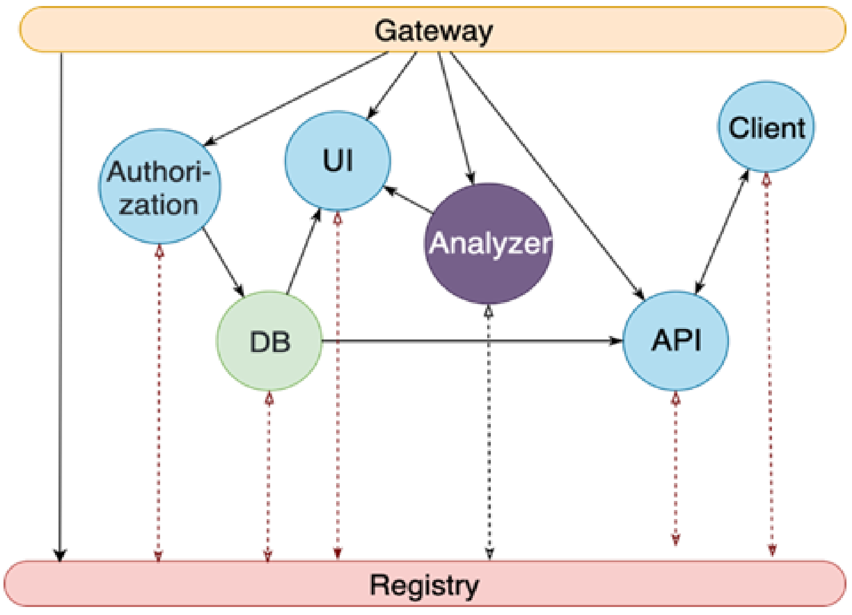


Fig. 4. Architecture of software

Table 1

Accuracy comparison of clustering methods

The number of "failed" tests	KNN	Support Vector Machines method	Naive Bayes method
100	0.7483	0.7948	0.8126
200	0.8347	0.8589	0.7650
500	0.9078	0.8451	0.7607
1000	0.9820	0.9241	0.7520

The Analyzer stores an index of the project's user logs and provides the ability to search for that index and used by the automatic analysis function.

UI is the service responsible for the client-side system. DB is the service responsible for the system database. The developed software system works with Java programming language and framework for automation of JUnit testing. Developed system allows using integration with bug-tracking systems such as Jira, Trello, Bugzilla, Airbrake, ZenHub. The system has the following structure of test organization levels: Launch – Test Suite – Test Case – Test step. Launch contains all test kits that were run on startup. Test Suite is a set of test cases that are combined in that they relate to one test module, functionality, priority, or one type of test. Each test suite consists of more than one test case and is often performed with a whole "bundle" in the process of testing. Test Case formally describes algorithm for testing a program, specially designed to determine the occurrence of a specific situation in a program, certain source data. Test Step describe the steps to play the bug. The steps are recommended to minimize, find the shortest way to reproduce the error and describe in the steps, it is important that they remain as clear as possible to developers. The system allows to combine multiple launches into one. If a project has a large number of test kits, they are split into parts because they cannot be in one particular startup. Once completed, they can be combined into a single startup to represent this data in dashboards and generate reports. Two types of mergers are implemented: Linear and Deep. If the user selects Line Merger, a new start is created. The new startup contains elements of the startup merge. The element levels remain the same as in the beginner startup. Statistics is calculated as the sum of the statistics of all merged launches. Initial launches are removed from the system.

Conclusions. The main feature of researched method comparison was accuracy and to reduce time and human resources for analyzing the results of software testing. The results of the comparison of these methods can be seen in Table 1. As can be seen from the table, the KNN method slightly "loses" in the accuracy of the method of Support Vector Machines on small on-fringes by an average of 3–6%, but on the sets the 500-1000 KNN algorithm shows 6–10% better results than the reference vector method and 23–26% better results than Naive Bayes method. A comparison is made with manual analysis of "failed" tests and using machine learning methods on a real project. The analysis of 150 tests was started, the algorithm coped with this task in about 20 minutes, and failed to automatically recognize only 9 tests, while it took the person about 4 hours to read all the errors and attribute them to categories. Thus, KNN algorithm coped with this task 12 times faster than humans.

The article compares the results of clustering methods of KNN, Support Vector, and Naive Bayes to automatically analyze software test results for 100, 200, 500, and 1000 failed tests. The automatic analysis of results is compared with the manual one. When studying the results of comparing automatic error analysis of software with 1000 failed tests, KNN method shows an accuracy of 0.982, which is the most accurate result among the clustering methods considered. When analyzing 150 failed tests, the algorithm showed a result 12 times faster than manual analysis. Thus, it is shown that the proposed software method is effective.

References:

1. Jain, A., Dubes, R. Algorithms for Clustering Data. Prentice Hall, IGI Global, 2012. Pp. 43–62.
2. Kaufman, L., Rousseeuw, P. Finding Groups in Data: An Introduction to Cluster Analysis. MA: MIT Press, 2007. Pp. 215–266.
3. Bradley, P., Mangasarian, O., Street, W. Clustering via Concave Minimization. Advances in Neural Information Processing Systems, vol. 9. MA: MIT Press, 1997. Pp. 368–374.
4. Jain, A., Dubes, R. Algorithms for Clustering Data. Prentice Hall, IGI Global, 2012, pp. 43–62.
5. Java stack trace, understanding and using for debug. URL: https://www.scalyr.com/blog/java-stack-trace--understanding/

Олещенко Л.М., Лисенко О.О. ПРОГРАМНИЙ МЕТОД КЛАСТЕРИЗАЦІЇ ЗВІТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ АЛГОРИТМУ KNN

Стаття присвячена розробці та впровадженню програмного методу для аналізу результатів тестування програмного забезпечення з використанням алгоритму KNN. Розробники програмного забезпечення намагаються уникати ручного тестування програмного забезпечення, щоб зменшити ризик людських факторів та витрати на тестування. Середній час аналізу помилок після тестування для 500 «невдалих» тестів може становити до 24 годин. IT-компанії потребують автоматизації аналізу помилок після тестування програмного забезпечення, щоб забезпечити замовнику повну картину того, що відбувається в проєкті, з детальним описом того, що вже працює і потребує доопрацювання. Це зробити дуже важко вручну, особливо коли проєкт дуже великий і проходить більше тисячі тестів. Є необхідність розробити програмну систему для автоматизованого аналізу результатів тестування програмного забезпечення, що скоротить час аналізу помилок і дасть змогу переглянути статус тестів у режимі реального часу. Проаналізовано наявні методи машинного навчання, які можна використовувати для аналізу результатів тестування програмного забезпечення. Обґрунтовано вибір найбільш відповідного методу машинного навчання для цього завдання, а саме k-nearest neighbors (KNN). На базі KNN створено вебдодаток для автоматичного аналізу результатів тестування програмного забезпечення. Архітектура програмної системи реалізована у вигляді мікросервісів. Основна ідея запропонованого методу полягає в тому, що stack trace використовується як даних для кластеризації, ця частина містить інформацію про причину «падіння» тесту. Запропонований метод дає змогу автоматизувати процес аналізу помилок, скорочуючи час і кількість людських ресурсів, необхідних для аналізу результатів тестування програмного забезпечення. У статті зіставлені результати методів кластеризації KNN, Support Vector та Naive Bayes для автоматичного аналізу результатів тестування програмного забезпечення для 100, 200, 500 та 1000 невдалих тестів. Автоматичний аналіз результатів порівнюється з ручним. У процесі вивчення результатів порівняння автоматичного аналізу помилок програмного забезпечення з 1000 невдалих тестів метод KNN показує точність 0,982, що є найбільш точним результатом серед розглянутих методів кластеризації. У процесі аналізу 150 невдалих тестів із використанням методу KNN отримано результат у 12 разів швидше, ніж із використанням ручного аналізу.

Ключові слова: *тестування програмного забезпечення, методи машинного навчання, test case, stack trace, кластеризація, алгоритм KNN, Elasticsearch, TF, IDF.*