Enhanced Model for Mining Software Repositories

P.C. Nwosu
Dept. of Physical Sciences
Rhema University Aba,
Abia State, Nigeria

F.E. Onuodu
Dept. of Computer Science
University of Port Harcourt, Rivers State, Nigeria

U.A. Okengwu
Dept. of Computer Science
University of Port Harcourt, Rivers State, Nigeria

ABSTRACT

This study presented Enhanced Model for Mining Software Repositories using Supervised Machine Learning technique. The work adopted Object Oriented Analysis and Design (OOAD) methodology for the system design and was implemented using PHP Hypertext Pre-processor scripting language for the purpose of enabling flexibility and user-friendliness in mining source codes from repositories. The database for the model was created using MySQL. The enhanced model utilized K-Nearest Neighbor (k-NN), a supervised machine learning algorithm for data classification to eliminate voluminous comments from source codes in order to reduce bulkiness. The results and performance evaluation of the existing and enhanced models were illustrated. The pre-defined parameters for both models comprised of the number of iterations for mining, the time taken to generate the codes in seconds and number of generated lines of codes. Seven iterations were carried out for both models in which the existing model generated a total of 56 lines of codes in 2.322 seconds, while the developed model generated a total of 93 lines of well-defined lines of codes in 0.017 seconds. Therefore, the results obtained clearly showed that the model performed much better than the existing model in terms of speed, accuracy and extraction of well defined codes. The model could be beneficial to data miners, programmers, software engineers, project managers of large industrial environments as well as researchers because relevant information from the study can be applied to problem-solving.

Keywords

Software Repositories, Source Code, Data Mining, Machine Learning, KNN

1. INTRODUCTION

Poor code readability is identified as a major issue in mining software repositories because readable codes are easier to understand, modify, maintain and reuse. Based on the ideas of developers, source code readability is directly related with code maintenance which is a very important feature with which software quality is measured [16]. A software repository which is as well referred to as Code Repository is a collective storage location for software resources and their artifacts which provides access to source codes to potential users for either personal use to develop a new project, contribute to other people's project or collaborate in a large project as a team. Among many other benefits, it supports code reuse and collaboration among programmers which are highly promoted in software engineering by providing remote access to source code modules. In addition, a code repository is also an archive where large amounts of source code for software applications are accessed. These repositories are used by developers of open-source projects and other collaborative projects in managing different changes by different individuals in assigned tasks referred to as version control.

[16] described Mining Software Repository as one among the remarkable and main expanding area in software engineering. In all engineering professions, precise measurement is a basic requirement; software engineering inclusive. Engineers as well as researchers attempt to numerically demonstrate the efficiency of software using different tools just for the purpose of software quality assessment. In order to measure software quality, several evaluation metrics have been presented and evaluated. A lot of applications are equally provided for gathering metrics from program descriptions.

The essential collection of tools allows users the opportunity to select the most suitable tool that meets their requirements. Nevertheless, it has created the assumption that most of these tools carry out the task of measuring, analyzing and executing the same metrics using the same approach. At the inception of mining software repository, scholars barely had access to software repositories because the few that were available then were hosted privately by enterprises but nowadays users can boast of unlimited access to code repositories as a result of availability of several public repositories with open access like SourceForge, Bitbucket, GitHub, RhodeCode, GitLab, SCM-Manager etc. Those in the field of research presently have a wealth of data to mine from and reach significant conclusions. Software engineering collaborative documents or artifacts are gathered and stored in software repositories. Similarly, communications among programmers and their teams are archived in mailing lists, newsgroups and personal archives, while different versions of modifications made to source codes are recorded in version controls like Concurrent Versions System [9].

Research in the field of mining software repositories is primarily focused on determining as well as creating new tools and approaches for discovering methods in which software repository mining can contribute to understanding software development process and progress, to promote predictions concerning the development of software and to effectively use the discovered information to make plans for prospective development [2].

Software developers should have the ability to explore code repository, determine essential libraries and discover who created them and ways of using them. Library authors have the necessity of monitoring the usage of their Application Programming Interface (API). These entail a considerable effort in providing efficient code exploration and browsing tools, nevertheless, Google Corporation has immensely benefitted from this effort, enhancing the efficiency of software developers. Unlimited access to software repositories promotes wide-range of code sharing and code reuse. A number of people may disagree about this model that depends so much on the extreme scalability of the Google build system, makes it very simple to add dependency and lowers the motivation for

software developers to create reliable and well considered APIs [12].

In software engineering, data mining approaches like classification, clustering and association rule are all concerned with finding valuable traits and relationships in data. Thus, it is used for solving software engineering issues like defective or susceptible systems, pattern reused, or modifications in source code modules. Extracting and getting important information from this type of data eliminates error and allows software developers to carry out timely project delivery. Mining structures software repositories has recorded huge success with different things including source codes, execution traces and version control but software repositories are also made up of unorganized data like specifications documents, plain text in bug reports, archived mailing list, comments in source code, identifiers etc. Mining software repository researches have been able to estimate that a very large percentage of all the data archived in software repositories is unorganized, which poses various difficulties due to the fact that such data are usually noisy, incomplete, unlabeled and unclear [17].

Approximately every sector of life such as education, security, transportation, engineering, science, healthcare, energy, entertainment etc depends on the smooth operation of superior quality software. Regrettably, software development is a rigorous, time-consuming and expensive process because software developers have to deal with the intrinsic complexity of software still keeping bugs off as well as timely delivery of exceptionally efficient software products to the users. So, there exists continuous need for improvements in software tools that enables highly functional and maintainable software. Novel approaches are regularly required so as to minimize software complexity to enable developers to create enhanced software products [1].

2. REVIEW OF RELATED WORKS

[16] presented Mining Software Repositories for Software Metrics. The authors presented a theoretical model for extracting information from software repositories for software metrics in order to produce the measure of software performance, productivity and readability (i.e. software metrics) for Industrial Environments. The proposed model was divided into three phases - the beginning phase, which describe in-depth organization of software repositories and decide the repository selection; the implementation phase, which deals with the retrieval of source code from repositories and presented standard measure coupled with its authentication with retrieved data and finally the reporting phase, which integrates the final outcome obtained from the presented metrics and its evaluation with industry benchmark by carrying out a review from software developers handling on open source projects. However, the developed model could not automatically assign a single comment per hundred lines of codes.

[18] presented the SmartSHARK, an ecosystem that allows research that can be replicated and reproduced with focus on mining software repository. It was opined that mining software repository serves as the basis for various practical software engineering practices. Gathering and analyzing intricate data could be very difficult, particularly when the involved data needs to be shared in order to facilitate repeatable research and open science practices. A major setback on the study was that the proposed Smart-Shark framework data was not extended to a large number of software projects.

[11] worked on Mining Patterns in Source Code Using Tree-Mining Algorithm. From the researcher's perspective, finding consistency in source codes is highly important to software developers, in both academic and industrial environments since it could offer essential information to provide assistance in diverse activities like understanding of source codes, code refactoring and fault localization. Furthermore, a novel model for extracting attributes in source code that was built on FREQT tree extraction algorithm codenamed FREQTALS was presented. Basically, some control measures that effectively facilitates the discovery of more important traits were put in place; which was followed by showing the way to effectively incorporate them into the FREQT algorithm. Illustrating the effectiveness of the control measures, the researchers collaborated with software engineers and engaged in a case study that enabled the identification of several useful attributes in a Java code repository. Nevertheless, the surveyed FREQT algorithm approach was not applied using any machine learning prototype that could provide significant improvement to the proposed study.

[5] presented Deploying Smart Program Understanding on a Large Code Base. The researchers posited that software engineers are confronted with the duty of categorizing large files and functions with no assistance. The proposed technique for tackling the challenge referred to as FEAT does automatic mining of source code artifacts with hierarchical agglomerative clustering. The clustering technique employed in the model applied a novel hybridized distance that incorporates written and structural elements mechanically mined from software codes as well as comments. The model was applied on Software Heritage, a huge open source repository composed of around sixty-five million free software projects. Incorporation of FEAT and Software Heritage program was aimed at offering very theoretical method to engage in search for software artifacts according to program topoi, thereby permitting the utilization of normal text in queries. Although the testing revealed that the approach was appropriate for gaining knowledge about large free software projects of about two thousand (2, 000) functions and one hundred and fifty (150) files that made it suitable for applying it in open-source projects. However, semantic categorization could not be utilized on every source codes to automatically discover very extensive duplicates and mining of text documents to discover unknown relationships among test cases.

[4] presented Discovering Program Topoi via Hierarchical Agglomerative Clustering. The researchers stressed that managing source codes in a more theoretical manner, more rapidly like human beings has gotten attention from software engineers; although regrettably, no any standard technique or application that accurately offer much assistance in handling huge source codes archives. Consequently an efficient remedy to the problem was presented for automatic mining program topoi that provides well organized listing of function names related with a directory of significant words. The study was extended to the area of program understanding based primarily on transforming raw input data into numerical form suitable for machine learning algorithm and automatic discovery of key features software components through analysis of source code and other artifacts. The presented model employed unsupervised machine learning algorithm to simplify the deployment and utilization. It exploited text mining and analysis of the structure of codes to direct the classification task. The proposed system was implementation a generic software analysis program and empirical analysis were conducted using several free software projects. During the evaluation, the proposed model was analyzed within the

context of a number of viewpoints including the grouping process, extensibility of the method as well the efficiency of the model. Nevertheless, a serious setback in the work points to the fact that the built system had issue with cost-benefit as well as evaluation standard.

[3] carried out research on Emerging Topics in Mining Software Repositories. The researchers presented an overview and a research argument this domain over these past period and additionally ascertained utilized machine learning approaches, present operational subjects and finally the potential challenges for the enhancement of decision-making process of organizations. Moreover, other emerging issues in software repository were pointed out including project connected with novel technology such as energy use in portable and wearable gadgets and security issues due to weakness; and software developmental process like constant integration, and integrated project repositories counterfeiting. Ways for applying data mining in mining software repositories were analyzed and more light was thrown on data samples, the models and the experimental test methods in practice. It was discovered that the software archives record huge amounts of data, even though there exist no typical tools for gathering data or mine particular information from repositories are available.

[10] worked on Exploring the Applicability of Low-Shot Learning in Mining Software Repositories. The researchers explored the applications of low-shot learning in extracting software artifact. The researchers suggest that if issues of categorization are considerately matched with the suitable deep learning model, it definitely becomes likely to realize significant outcome with considerably little training samples than what are normally exploited for deep learning but it does not suggest that deep learning model is the appropriate application for all task. Nonetheless, for tasks involving deep learning models, exploiting a low-shot approach could support the employment of them whenever little training samples would ordinarily not permit their use.

[20] worked on Overview of Different Approaches to Solving Problems of Data Mining. The paper dealt with important activities in the evaluation of huge volume of data coupled with comparing different approached to ascertain the outcome. The researcher opined that data miming can be applied to the transformation of raw data and solving pre-processing problem. The k-Nearest Neighborhood algorithm and the Decision Tree algorithm were utilized for data classification and regression in specified domain. The results analyses indicates that the solution tree developed for given subject areas was highly effective but fails with increase in the volume of data. The model developed for k-nearest neighbour method is also effective but the operating time slightly increased with increase in the volumes of data but it is still very fast. Therefore, it can be deduces that both approaches are appropriate for employment on specific topics, with extended correctness and speed. The k-nearest neighbor classifier moderately maintains reduce extensibility, while the decision tree method can be extended with no significant rise in training time, hence, the Decision Trees algorithm provides more satisfactory result than the k-Nearest Neighborhood algorithm.

[19] presented Accelerating Source Code Analysis at Massive Scale. An approach that decreases the total execution time for source code extraction function carried out in extremely huge repositories was presented, particularly the ones associated with analysis that involves control and data flow. The main purpose was to study code extraction activities in order to determine and eliminate unimportant data from source codes before the mining task is executed. The system was evaluated with 16 classical control and data-flow analyses that are usual components of mining tasks and seven million (7 million) Control Flow Graph (CFG). The observation was that greater number of Reduced Control Flow Graph (RCFG) depictions of programs in the data samples showed similarities in relation with nodes and edges. This causes a remarkable survey to ascertain if the resemblance could be applied to executing extraction procedure just on distinctive RCFG and recycle the results. The outcome revealed that the method could realize an average of forty percent decrease execution time and the viability of the proposed method. Even though the similarity determined in the RCFG could not be investigated to ascertain if it could be used execute the procedure just on distinctive CFG to produce superior quality outcomes.

[8] worked on Mining Software Repositories for Adaptive Change Commits Using Machine Learning Techniques. The researchers examined the version information of free software in order to automate the categorization of version merges into one of two different groups, which were the adaptive commits and non-adaptive commits. The commits gathered the version information of three free software and eight different codes edits measure associated with different things such as the number of edited statement of codes, functions, modules and files. On the basis of these edits measures, a machine learning method to determine if a merge conformed to the changes or not was developed. Based on the result, it was deduced that code edits measure could be suggestive of tailored maintenance tasks. Additionally, the categorization result indicated that the developed machine learning algorithm showed about 75% forecasting accuracy in marked modification information. This showed that the technique was capable of automatic processing the evaluation of version information of software products and determined the particular code commits that were connected to tailored maintenance activities. It was observed that the algorithm could provided a better platform for building improved algorithm with forecasting ability on tailored commits devoid of the necessity of physical efforts. Although, the adopted algorithm could not predict the efficiency of the tailored commits from modification logs that were not classified.

[15] worked on A Novel Scheme for Improving Accuracy of KNN Classification Algorithm Based on the New Weighting Technique and Stepwise Feature Selection. The researchers proposed a new system to enhance the correctness of k-Nearest Neighbor categorization tool using the latest weighting method and stepwise feature selection. First, a stepwise selection technique was applied to remove inappropriate attributes and choose very interrelated attributes with the class group. Subsequently, the weighting technique was applied in order to grant weight value to each of the data in the training dataset according to neighbor groups and the Euclidean distances. The weighting method offers top priority to data samples with neighbors having close Euclidean distance in the same category, which could efficiently enhance the accuracy of categorization of the system. The correctness level of the presented model was evaluated and examined using the conventional K-Nearest Neighbor technique and other related study employing of five actual UCI data sets. The outcome of the evaluation determined that the presented model efficiency was far better than the conventional K-Nearest Neighbor method with a record of about 10% improvement on accuracy. This indicated that the algorithm significantly improved the classification performance of the traditional K-NN method. But the experiments were not conducted on various real-world applications.

3. METHODOLOGY

The Object Oriented Analysis and Design (OOAD) methodology was employed in this research because it makes implementation simple; it is also very efficient and supports reusability. The OOAD software development methodology employs a repetitive and object-oriented building approach which breaks down a whole system into smaller segment or modules. It is a widely used scientific approach for analysis and design from the implementation of an object-oriented model all through the entire system development procedure. OOAD offers a modern software engineering practice that is excellently carried out to meet the system's goals.

3.1 Object Oriented Analysis

The system was analyzed using the architecture in Figure 6. K-Nearest Neighbour algorithm was introduced in the system to handle data classification in order to appropriately identify comments in source code for possible removal; this was to ensure improvement in readability of source code.

3.1.1 K-Nearest Neighbour Algorithm

K-Nearest Neighbour algorithm is the basic and easiest classification method. With little or no previous information concerning the distribution of data, the algorithm classifies every sample together based on closeness of data with similar attributes. One of the most crucial elements in the KNN method is the k value. There is no exact value for k, and the appropriate value depends on the problem's space and data distribution [7]. In the application of k-Nearest Neighbour algorithm, an item is categorized using majority vote of its neighbors, every item is allotted to the most common group amongst the nearest neighbors (k is typically a small positive integer). For example, If k = 1, the item is just allocated to the class of its nearest neighbours [13].

3.1.2 The Traditional K-Nearest Neighbour

Step-1: Start.

Step-2: Choose the number *k* of the neighbours.

Step-3: Compute the euclidean distance of *k* number of neighbors.

Step-4: Take the *k* nearest neighbors based on the calculated euclidean distance.

Step-5: Among these *k* neighbours, count the number of the data samples in each group.

Step-6: Assign a new data points to that group with the maximum of number of neighbours.

Step-7: End.

3.1.3 The K-Nearest Neigbhour Algorithm for the developed system

Step1: Start
Step2: Initialize the system
Step3: Activate the proposed model Step4: Input User Request (R)
Step5: Read R using K-NN function
Step6: Increment R
Step7: R =R + 1
Step8: input set of classes in the program to mine software repository
Step9: input set of 4 datasets type (e.g. null method)
Step10: refactoring_count = 0
Step11: repeat
Step12: classes for code files = set of classes in program
Step13: while!empty (classes) do
Step14: class = classes.pick()
Step15: if fitness_function_improves () then
Step16: refactoring_count++
Step17: update system output
Step18: else
Step19: refactoring.undo()
Step 20: end

3. 2 System Design

The various steps taken in the design of the model for mining source codes in software repositories was discussed in this section. The designs discussed in this section include the interface design, the functional design, input/output specifications and the use-case design.

3.2.1 Interface Design

The interface design involves the specification of programming language and other tools that were employed in the development of the user interface of the system.

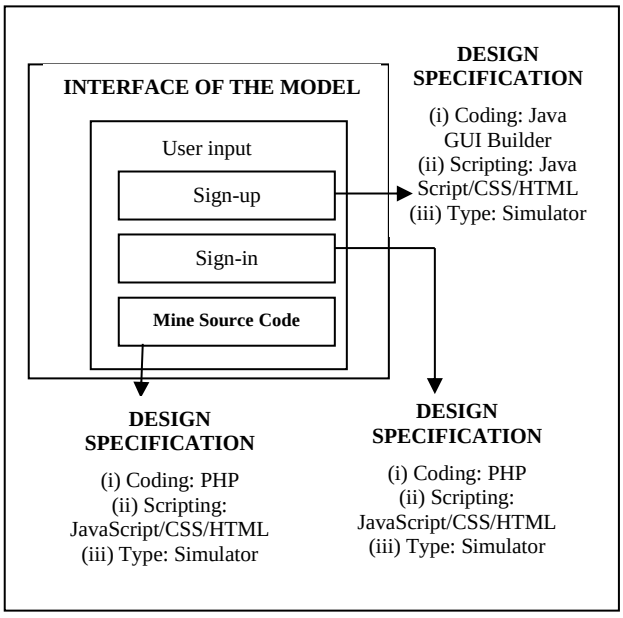

Figure 1: Interface Design of the System

3.2.2 Functional Design

The functional design of the system indicates the flow of information within the system. It shows the internal mechanism of the system; that is, how the system works. It was used to simplify the design of the system and further assure that every segment of the system is assigned a single task and that the task is carried out without any negative impact on other modules. Interaction could exist between software components, hardware components, peripherals, users or any of them combined.

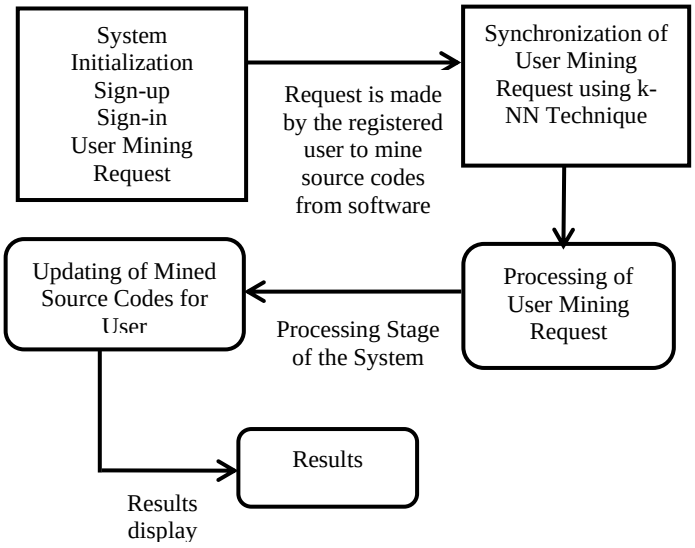

Figure 2: Functional Design of the System

3.2.3 The Use-Case Design

The Use-Case design process involves a dynamic context which describes interactions of the system with its users and environment. Figure 3.5 shows the use-case design of the system.

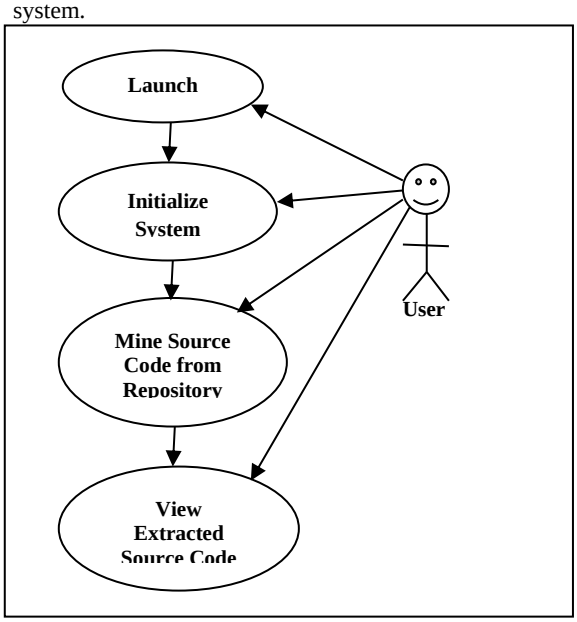

Figure 3: Use-case Design

3.3 The Architecture of the System

The architecture of the model for mining source codes repositories is illustrated in Figure 4. The red dotted line shows the introduction of the k-nearest neighbour algorithm which served the purpose of identifying comments for possible removal from source codes in order to reduce bulkiness and improve readability, thereby enhancing the model.

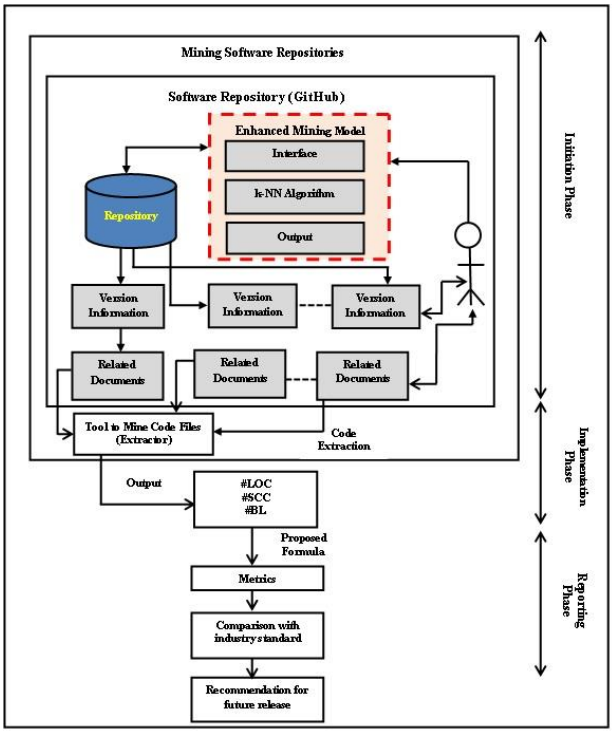

Figure 4: Architecture of the System

4. CONCLUSION

Enhanced model for mining repositories was developed and implemented for the purpose of enabling flexibility and user-friendliness in mining source codes. The developed model utilized the K-Nearest Neighbor (k-NN) algorithm, a supervised learning technique and a flexible interface to reduce bulk comment lines in mined source codes from repositories. The system demonstrated a very good performance in mining source codes from code repositories.

The research could further be extended by integrating a different classification algorithm with the K-NN algorithm to have a hybridized model. This could provide more excellent approach for indexing and classifying data in order to place each document on a proper footing based on a considerable standard. This could significantly provide better classification and offer greater improvement on the system since better predictions can be made.

5. RECOMMENDATION

The developed model could be beneficial to data miners, software developers, technical personnel, project managers of large industrial environments and researcher in the areas of data mining and software engineering. This is because; the beneficiaries can utilize relevant information from the study in solving complex issues relating to data mining.

6. REFERENCES

[1] Allamanis, M. (2019). The Adverse Effects of Code Duplication in Machine Learning Models of Code. Proceedings of the ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, 143-153.

[2] Chaturvedi K., Singh V., and Singh P. (2013). Tools in Mining Software Repositories. Proceedings of the 13th

International Conference on Computational Science and Its Applications, IEEE Press, 1, 89 – 98.

[3] Güemes-Peña, D., López-Nozal, C., Marticorena-Sánchez, R., and Maudes-Raedo, J. (2018). Emerging Topics in Mining Software Repositories. Progress in Artificial Intelligence, 7(3), 237–247.

[4] Ieva, C., Gotlieb, A., Kaci, S. and Lazaar, L. (2018). Discovering Program Topoi via Agglomerative Clustering. Proceedings of the Thirty-Second IAAI/AAAI Conference on Innovative Applications of Artificial Intelligence, IEEE Transactions on Reliability, 69(3), 758-770.

[5] Ieva C, Gotlieb A., Kaci S. and Lazaar L. (2019). Deploying Smart Program Understanding on a Large Code Base. Proceeding of the 1st IEEE International Conference on Artificial Intelligence Testing (AITest), San Francisco East Bay, CA, USA. 73 - 80.

[6] Kim, K. (2021). Normalized Class Coherence Change-Based KNN for Classification of Imbalanced Data. Pattern Recognition, 120, 108126.

[7] Kuhkan M. (2016). A Method to Improve the Accuracy of K-Nearest Neighbor Algorithm. International Journal of Computer Engineering and Information Technology, 8(6), 90-95.

[8] Meqdadi, O. and Alhindawi, N. (2019). Mining Software Repositories for Adaptive Change Commits Using Machine Learning Techniques. Information and Software Technology, 109, 80-91.

[9] Olatunji S. O., Idrees S. U., Al-Ghamdi Y. S. and Al-Ghamdi J. S. A. (2010). Mining Software Repositories: A Comparative Analysis. International Journal of Computer Science and Security (IJCSNS), 10(8), 161–174.

[10] Ott, J., Atchison, A. and Linstead, E. J. (2019). Exploring the Applicability of Low-Shot Learning in Mining Software Repositories. Journal of Big Data 6(35), 1-10.

[11] Pham H. S., Nijssen S. and Mens K. (2019). Mining Patterns in Source Code Using Tree Mining Algorithms. Proceedings of the 22nd International Conference on Discovery Science, Split, Croatia. Lecture Notes in Artificial Intelligence, 11828, 471–480.

[12] Potvin, R. and Levenberg, J. (2016). Why Google Stores Billions of Lines of Code in a Single Repository. Communications of the ACM, 59(7), 78-87.

[13] Raikwal, J. S. and Saxena, K. (2012). Performance Evaluation of SVM and K-Nearest Neighbor Algorithm over Medical Dataset. International Journal of Computer Applications, 50(14), 35-39.

[14] Ram-Kumar, R. P., Polepaka, S., Lazarus S. F. and Krishna, D. V. (2019). An Insight on Machine Learning Algorithms and Its Applications. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 8(11S2), 432-436.

[15] Siddiqui and Ahmad (2019). Mining Software Repositories for Software Metrics (MSR-SM): Conceptual Framework. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 8(10), 4173-4177.

[16] Sheikhi, S. and Kheirabadi, M. T. (2020). A Novel Scheme for Improving Accuracy of KNN Classification Algorithm Based on the New Weighting Technique and Stepwise Feature Selection. Journal of Information Technology Management, 12(4), 90-103.

[17] Thomas S. W., Hassan A. E. and Blostein D. (2014). Mining Unstructured Software Repositories. Evolving Software Systems, 139-162.

[18] Trautsch, A., Trautsch, F., Herbold, S., Ledel, B., and Grabowski, J. (2020). The Smart-Shark Ecosystem for Software Repository Mining. Proceeding of the 42nd International Conference on Software Engineering (ICSE). ACM, New York, USA, 24-28.

[19] Upadhyaya, G. and Rajan, H. (2018). On Accelerating Source Code Analysis at Massive Scale, IEEE Transactions on Software Engineering, 44(7), 669-688.

[20] Vadim K. (2018). Overview of Different Approaches to Solving Problems of Data Mining. Procedia Computer Science, 123, 234–239.