Development Operations for Continuous Delivery

Supritha K[1], Badari Nath K[2]

[1]Student, Dept. of Computer Science and Engineering, RV College of Engineering, Karnataka, India
[2]Assistant Professor, Dept. of Computer Science and Engineering, RV College of Engineering, Karnataka, India

---***---

Abstract - *Development Operations (DevOps) is a methodology that promotes development, testing and deployment to be done in parallel so that applications or services can be continuously delivered to the customers. The major change that DevOps brings about is the automation of manual tasks. This paper discusses the various automation procedures carried out in a DevOps environment. The paper demonstrates the use of various continuous integration tools like Git version control system, Gerrit code review tool, Jenkins Continuous integration server and SonarQube static code analyzer. Any piece of code written must be built before testing and it will be helpful to automate the build procedure to make it faster and error free. The analysis of the build log files will give the status of the build and a machine learning approach has been followed for this purpose. During this process the accuracies obtained by different supervised machine learning algorithms has been compared. The paper also talks about static code analysis with the help of SonarQube to help maintain the code quality. The methodologies followed in the paper demonstrate how automation in DevOps saves human time and eliminates errors. It throws light on the performance of different machine learning algorithms and provides a method for code quality maintenance.*

Key Words*: *Development Operations, Continuous Delivery, SonarQube, Machine Learning, Test Packaging.

1. INTRODUCTION

There is a significant increase in competition in the software market and hence organizations are focused on dedicating resources to develop and deliver higher quality products in an accelerated pace. Development Operations (DevOps) and Continuous Delivery (CD) play a significant role in this endeavor. These methods help in delivering products in an accelerated pace all the while maintaining the quality of the products. Continuous practices provide several benefits such as (1) Quick feedback (2) Frequent and reliable releases (3) Elimination of manual tasks through automation. Many industrial cases show that continuous practices are making a significant impact in software development industrial practices across various sizes and domains of the organization. The migration to continuous practices might not be an easy task because many of the tools may not support the highly complex and challenging nature of these practices. An organization may have several customers with a wide variety of requirements. Software's will be written for each requirement. Once the software is in use, it will be put under maintenance. New versions of the software will be released regularly, and they must be integrated with the older versions. The changes made in the new version might be very minute when compared with the older version, hence only the new changes must be integrated, and this is where continuous practices comes into picture. Many organizations will be manufacturing hardware devices for which the same organization will schedule periodic releases. For example, in communication networks the base transceiver stations will have hardware devices for which the software must be written. The written source code will have to be continuously monitored if it is meeting the necessary quality requirements and SonarQube is one such static code analysis tool. It provides several details about the code quality like the number of bugs, vulnerabilities, code smells, technical debt etcetera. Test packaging is one feature in which all the source code written will have to be build depending on the various build specifications and based on if the status of the build (success/ failure) the packaging activity will be carried out. These packages will have to be tested and after all the test conditions have been passed the software will be ready for release. Supervised machine learning algorithms play an important role in classification problems. The classification problem in our case is the result of the build. After the build is completed a build log will be created and analysis of this log file will indicate if the build is a success or failure. The parameters that define the status are considered as attributes and the status as the label. Several build logs are labeled, and a training and test set is formed. Machine learning algorithms like decision trees and logistic regression can be trained with the training data and tested on the test data. The accuracy thus obtained will tell the performance of the supervised learning algorithm.

2. Background Study

A brief literature survey was carried out to gain understanding about various concepts like continuous practices, machine learning algorithms, technical debt and SonarQube. Excerpts from a few of the papers/ journal referred are as follows:

Mojtaba Shahina et. al. give an overall description and relationship between various continuous practices like continuous integration, continuous delivery and continuous deployment. The source code written by the developer will be committed to a repository. Continuous integration will clone these repositories in the continuous integration servers to build and test the code. Continuous delivery ensures that an application is always in a production ready state after passing all automated tests and quality tests. This practice has several advantages like reduced development risks, lower costs and faster feedback system. Continuous delivery consists of acceptance test and manual movement to production. Continuous deployment automatically and continuously deploys the application to production or customer environment. It consists of acceptance test with automatic movement to production [1].

Carmine Vassallo et. al. have done a study on continuous refactoring in continuous integration. Continuous refactoring is the process of simplifying and clarifying the design of an existing source code without changing its behavior. Some of the key findings of the study were (1) Continuous integration is the right context to apply refactoring. (2) Preventing quality issues is better than fixing. (3) Developers like to consider the output of static analysis tools while deciding whether to refactor or not, however the results generated by these tools might not be accurate which results in the developer not trusting the tool. The tools must be improved such that proper warnings are provided as to when refactoring must be carried out. (4) Developers need to perform refactoring continuously, but this operation is very time consuming and dependent on surrounding development team allocation. Thus, refactoring recommenders and prioritization approaches should exploit effort and community related factors while suggesting which refactoring operations are suitable in a given development context. (5) The refactoring opportunities must be summarized by recommenders because some of the developers were not aware of the refactoring tactics [2].

Qimin Cao et. al. demonstrate a method to analyze web http log files and predict attacks using Decision trees. Web servers are prone to attacks because of their high value. Anomaly detection plays an important role in web security. The log files can be manually analyzed but the length of the files is too long, and attack methods are various. Hence the machine learning method is proposed. The paper proposes a two-level machine-learning algorithm. The data processing phase will involve data extraction and data labeling. The decision tree model is trained to classify normal and anomalous data sets. The normal data set is manually checked for establishing multiple Hidden Markov Models (HMM). The experimental dataset is obtained from real industrial environment. The experimental analysis shows higher detection rate accuracy when compared to other models [3].

Makrina Viola Kosti et. al. discuss the methods for technical debt assessment. Technical debt is somewhat like financial debt. While development is in progress some code might be written that is easy to implement at this point of time, but it will have to be changed later to maintain efficiency. The efficient code can also be implemented but it might take a longer duration. The debt that is incurred while implementing the easy solution is called as technical debt. The most common ways to assess technical debt is (1) structural proxies through quality metrics (2) monetized proxies through the use of Software Quality Assessment based on Lifecycle Expectations (SQALE). In this paper they analyze the relationship between the two methods based on data obtained from 20 open source software projects. A regression model is built that shows the relationship between the two methods [4].

3. METHODS

3.1 Experimental Setup

The various tools and programming languages used while carrying out the experiment is as follows:

- Git Version Control System
- Gerrit Code Review Tool
- SonarQube 7.6 static code analyzer
- MobaXterm Terminal Simulator
- Jenkins Continuous Integration Server
- Python Integrated Development Environment
- Shell Scripting

3.2 Design Modules

The modules that make up the entire system are depicted in Figure 1.

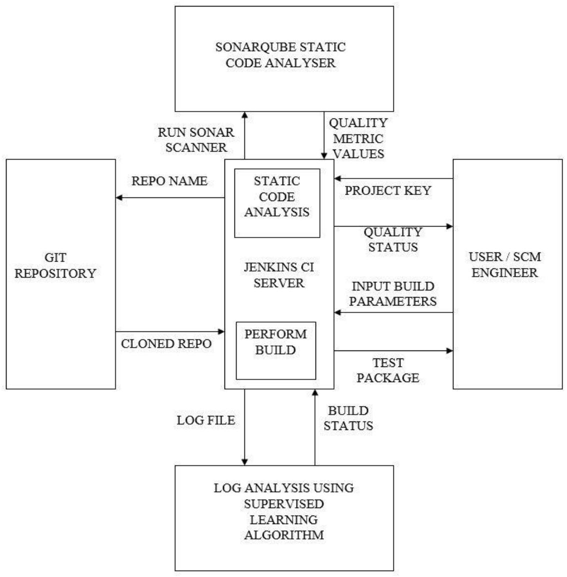

Fig -1: Design Modules
Chart -1: Name of the chart

Test Packaging

For the test package creation the user or Software configuration Management engineer will give inputs to the Jenkins CI server. Jenkins is an open source automation server written in java programming language. It is used to run automated scripts without human intervention. Each job created in Jenkins when run, is called as a build. Jenkins can be configured to provide users with an interface to enter build parameters. Jenkins "Execute Shell" configuration can be used to specify all the scripts that have to be called for a job based on the required order. It can also be configured to send email's to specified recipients regarding the status of the build like success or failure. The input for test packaging will consist of the various build parameters specific to the type of build. The code that has to be built must be cloned from the Git repository. Git is a version control system that is used to hold all the code changes made by developers. All updated codes are available in the Git repo pertaining to the organization. If any changes have to be made then the code must be checked out from the repository and that cloned workspace has to be used. All changes made can be merged to existing code with the help of various Git commands, for example: "git commit" command will commit the changes made in current workspace to main Git branch in the repository. Once user input is obtained the "perform build" component will carry out all the necessary build steps and generate a log file as output. This log file is analyzed and the result will indicate the status of the build success/failure. If the build is successful then the packaging procedure can be carried out to get packages as an output.

Log File Analysis

The log file that is obtained as output after the build procedure has to be manually analysed to check if the build is a failure or a success. Since this procedure can be time comsuming a machine learning approach is followed. The log files of few older builds are labelled to form training and test instances. Once these algorithms are trained, new log files can be given as input to determine the status of the build in lesser time. A comaprision of the accuracy of both the learning models is carried out to determine the best model.

Static Code Analysis

Any code written must be analyzed to determine if it meets the quality standards. SonarQube is a static code analyzer that will recursively scan the code file present in each folder and return the values for various metrics like bugs, code smells, vulnerabilities etcetera. Based on the results obtained it can be concluded if the code meets the quality cutoff or not.

3.3 Flow Chart

Figure 2. gives the flow chart of the entire system. Packaging and static code analysis is two events that happen in parallel. For packaging the build environment is first set and build is performed. The log file that is obtained is analyzed to check the build status. On successful build packaging operation is performed and if failure then code must be analyzed to fix errors. In static code analysis the rules are generated in SonarQube. Sonar scanner is run for each of the jobs and the metric values are analyzed. If the values meet the cutoff then the static code is approved for merged, if not then code changes have to be made.

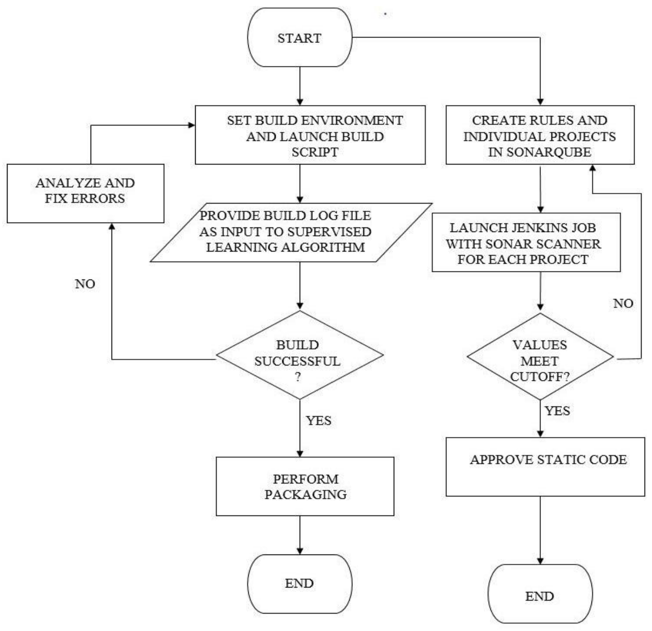
START
SET BUILD ENVIRONMENT AND LAUNCH BUILD SCRIPT
CREATE RULES AND INDIVIDUAL PROJECTS IN SONARQUBE
ANALYZE AND FIX ERRORS
PROVIDE BUILD LOG FILE AS INPUT TO SUPERVISED LEARNING ALGORITHM
LAUNCH JENKINS JOB WITH SONAR SCANNER FOR EACH PROJECT
NO
NO
BUILD SUCCESSFUL ?
VALUES MEET CUTOFF?
YES
YES
PERFORM PACKAGING
APPROVE STATIC CODE
END
END

Fig – 2: Flow Chart

3.4 Implementation

Gerrit is a code review tool used to see side-by-side difference viewing and commenting. It is used to make code review simple and quick and it is used along with Git version control system. Gerrit allows authorized contributors to merge changes to the Git repository after review is completed. The URL's required for repository clone is available in Gerrit according to the project. It also displays a tree structure of the file paths involved in the project structure. The automation script that has been written to perform builds will refer Gerrit for cloning repositories. Jenkins is a continuous integration tool that is used to trigger jobs periodically or as and when needed. A Jenkins job is created for this build. Jenkins job can be configured to accept inputs from the user. The inputs provided for the build are

- Git branch type
- Release number
- Build type
- Build number
- Email id

The user email id is taken as an input parameter to send the results of the build to the user. The input provided is first validated and if any parameter does not meet the requirement then the build will fail. Based on the input provided by the user the Git repository is cloned. Next the build environment is set, and the build command is executed. The output of build completion is obtained in the form of a build log file. Now this build log file must be analyzed to determine if the build is a success or not. For this process a machine learning approach has been followed. The log file is first subjected to a label data procedure where the file is searched for the occurrence of three patterns and the size of the log file is noted. The three patterns are:

- EPSILON build finished without Errors
- Done with: common
- Done with: src

These three patterns and the size of the file form a total of four attributes and the status of the build forms the label. The build is a success only if all these three patterns are found and the size of the file is less than 6MB. If the pattern is present, then the attribute will be given a binary value 1 and if not then 0. If the build is successful then the label is 1, else 0. The build log file is given as an input and the output obtained is a comma-separated file. This procedure is repeated several times to obtain several rows that can be classified as training and test dataset. Two supervised learning algorithms are considered for analyzing the log file

- Decision trees
- Logistic regression

Both the training and test dataset are given, as an input to both these algorithms and the output is the accuracy of the algorithm. Accuracy is defined as (total correct classification / total number of samples) *100. Decision trees provided better accuracy than logistic regression. Next the input CSV file provided to the algorithm will return the build status. If the build is successful, then packaging will be performed else a failure email will be sent to the user. The packages thus created will have to be tested and validated before the final merge.

SonarQube is an open source static code analysis tool. It depicts the overall health of an application in terms of quality and hence helps a developer to maintain the quality of code. Quality profiles service of SonarQube plays and important role, as this is where all the rules are defined. For example, a rule can specify that a method should not have cognitive complexity greater than 15. Sonar-way is the default rule specified for all projects, but it is best practice to define own rules depending on the project requirements. The rules must be formatted in the form of an xml document and they can be restored in quality profiles. The tags considered in the xml are:

- Rule Name
- Language
- Repository Key
- Key
- Priority

The rules thus set can be used by a few projects or it can be set as default so that all projects can use it. Once rules are defined the projects must be cloned from Git repository to get a working directory. In the working directory a few files need not be considered for quality testing and hence these file folders that must be eliminated forms the exclusions list. A sonar-project.properties file is created that contains the file folders in which cppcheck has to be run recursively. The property file also contains details pertaining to the project like:

- Project Key
- Project Name
- Project Version
- Project Modules

Cppcheck is an analysis tool for C/C++ code, unlike other compilers it does not check for syntax errors. It detects errors and bugs that compilers normally fail to detect. The goal is to have no false positives. If cppcheck is not run properly in all modules the errors displayed might be wrong and hence an automated script will make the job error proof. SonarScanner is a recommended scanner to analyze projects with SonarQube. After exclusions are removed and cppcheck is run, sonar-scanner is launched, and it will perform the analysis and return the results on SonarQube. The URL returned by the scanner will display the quality metrics of the project like:

- Code Smells
- Bugs
- Technical Debt
- Vulnerabilities

A Jenkins job is created with the above-mentioned steps for each of the projects that need sonar analysis. Figure 3 shows the overall working of sonar static code analyzer.

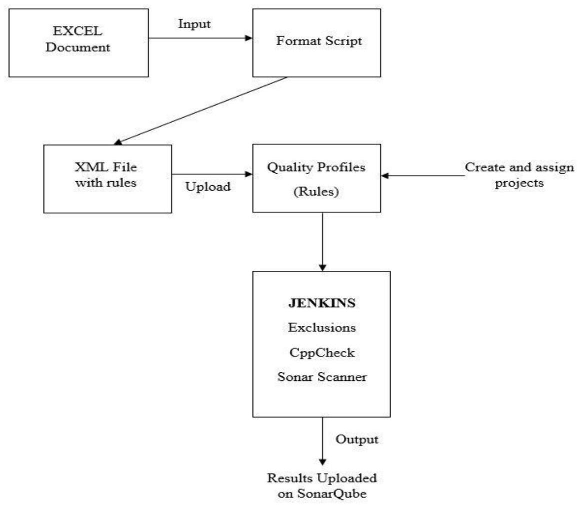

Fig – 3: Sonar Analysis

3.4 Implementation

The build log files that are analyzed using supervised learning algorithms return two different accuracies. The accuracies obtained were:

- Decision Trees – 100%
- Logistic regression – 60%

The accuracy obtained in case of decision trees in 100% because of the limited number of samples. However, decision trees are giving a better performance when compared to logistic regression.

SonarScanner is run periodically every time a code commit happens. The main goal is to be notified if there are any changes in the quality metrics when a new commit happens. A job is designed such that the quality metrics of the current run and previous run is compared and if the values are greater than the previous run then the job will notify all required personnel about the increased values and the developer responsible for introducing the changes. The goal is to maintain zero values for bugs and vulnerabilities and lower values for code smells and technical debt. If the new run gives lower than previous values, then these new values will become the cutoff. Figure 4 shows the sonar scanner results on SonarQube. Figure 5 sows the history of previous scans. Figure 6 shows a graph of code coverage versus technical debt. It depicts the reliability rating and security rating.

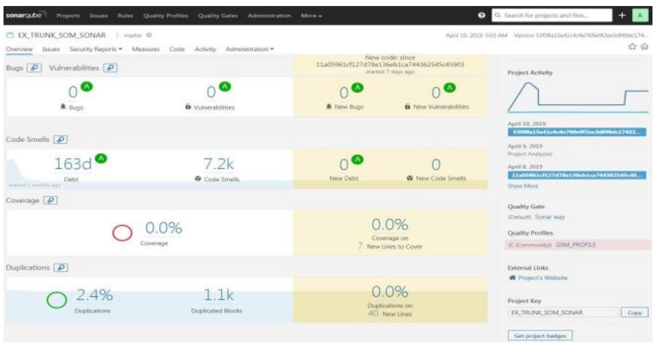

Fig – 4: SonarQube Results

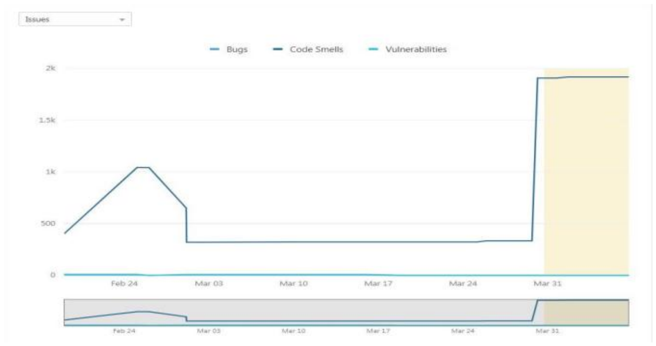

Fig – 4: SonarQube Results

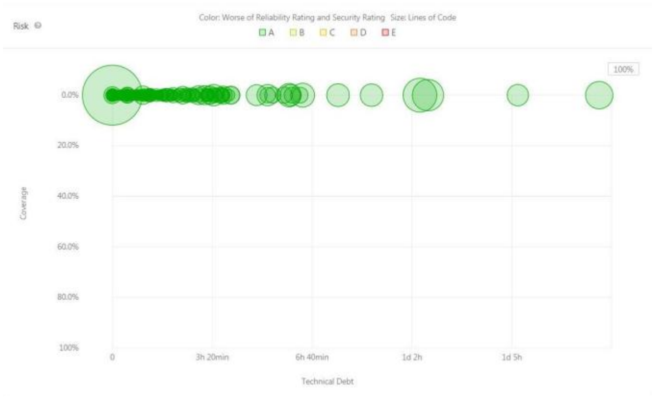

Fig – 4: SonarQube Results

4. CONCLUSION

This paper throws light on how automation can help reduce manual tasks in continuous practices. The packaging task takes lesser time for completion. The various methods and methodologies that are used and carried out in a day to day continuous integration practice is demonstrated in this paper. The machine learning algorithms can be used in other applications as well if the labeling criteria are changed. Better accuracy can be obtained if the number of samples is increased. SonarQube has proven to be an excellent quality analysis tool and it can be used for projects in other languages as well if error detection tools other than cppcheck are used.

REFERENCES

[1] M. Shahin, M. Ali Babar and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," in IEEE Access, vol. 5, pp. 3909-3943, 2017.

[2] C. Vassallo, F. Palomba and H. C. Gall, "Continuous Refactoring in CI: A Preliminary Study on the Perceived Advantages and Barriers," 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, 2018, pp. 564-568.

[3] Q. Cao, Y. Qiao and Z. Lyu, "Machine learning to detect anomalies in web log analysis", 2017 3rd IEEE International Conference on Computer and Communications (ICCC), Chengdu, 2017, pp. 519-523.

[4] M. V. Kosti, A. Ampatzoglou, A. Chatzigeorgiou, G. Pallas, I. Stamelos and L. Angelis, "Technical Debt Principal Assessment Through Structural Metrics," 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, 2017, pp. 329-333.

[5] M. Brandtner, E. Giger and H. Gall, "Supporting continuous integration by mashing-up software quality information," 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering,

and Reverse Engineering (CSMR-WCRE), Antwerp, 2014, pp. 184-193.

[6] T. Amanatidis, N. Mittas, A. Chatzigeorgiou, A. Ampatzoglou and L. Angelis, "The Developer's Dilemma: Factors Affecting the Decision to Repay Code Debt," 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), Gothenburg, 2018, pp. 62-66.

[7] N. A. Ernst, S. Bellomo, I. Ozkaya and R. L. Nord, "What to Fix? Distinguishing between Design and Non-design Rules in Automated Tools," 2017 IEEE International Conference on Software Architecture (ICSA), Gothenburg, 2017, pp. 165-168.

[8] Y. Lu, X. Mao, T. Wang, G. Yin, Z. Li and H. Wang, "Poster: Continuous Inspection in the Classroom: Improving Students' Programming Quality with Social Coding Methods," 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), Gothenburg, 2018, pp. 141-142.

[9] D. Guamán, P. A. Quezada-Sarmiento, L. Barba-Guaman and L. Enciso, "Use of SQALE and tools for analysis and identification of code technical debt through static analysis," 2017 12th Iberian Conference on Information Systems and Technologies (CISTI), Lisbon, 2017, pp. 1-7.

[10] M. Martignano, "Bounded model checking and abstract interpretation of large C codebases," 2017 IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace), Padua, 2017, pp. 16-20.

[11] B. Barta, G. Manz, I. Siket and R. Ferenc, "Challenges of SonarQube Plug-In Maintenance," 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), Hangzhou, China, 2019, pp. 574-578.

[12] J. Holvitie and V. Leppänen, "DebtFlag: Technical debt management with a development environment integrated tool," 2013 4th International Workshop on Managing Technical Debt (MTD), San Francisco, CA, 2013, pp. 20-27.

[13] A. Martini, "AnaConDebt: A Tool to Assess and Track Technical Debt," 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), Gothenburg, 2018, pp. 55-56.

[14] A. Shapochka and B. Omelayenko, "Practical Technical Debt Discovery by Matching Patterns in Assessment Graph," 2016 IEEE 8th International Workshop on Managing Technical Debt (MTD), Raleigh, NC, 2016, pp. 32-35.

[15] P. Sutheebanjard and W. Premchaiswadi, "Fast convert OR-decision table to decision tree," 2010 Eighth International Conference on ICT and Knowledge Engineering, Bangkok, 2010, pp. 37-40.

[16] D. V. Patil and R. S. Bichkar, "A Hybrid Evolutionary Approach To Construct Optimal Decision Trees With Large Data Sets," 2006 IEEE International Conference on Industrial Technology, Mumbai, 2006, pp. 429-433.

[17] Y. Tao, D. Dong and P. Ren, "Notice of Violation of IEEE Publication Principles Decision Trees Generation Based on Fault Trees Analysis," 2009 International Forum on Information Technology and Applications, Chengdu, 2009, pp. 178-180.

[18] Juan Sun and Xi-Zhao Wang, "An initial comparison on noise resisting between crisp and fuzzy decision trees," 2005 International Conference on Machine Learning and Cybernetics, Guangzhou, China, 2005, pp. 2545-2550 Vol. 4.

[19] H. Xie and F. Shang, "The study of methods for post-pruning decision trees based on comprehensive evaluation standard," 2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Xiamen, 2014, pp. 903-908.

[20] H. Yang, "Research on Cost Decision of Specialized-Automobile Manufacturing Enterprise Based on the Theory of Decision Tree," 2010 International Conference on Digital Manufacturing & Automation, Changsha, 2010, pp. 198-203.

[21] X. Chen and R. Ye, "Identification Model of Logistic Regression Analysis on Listed Firms' Frauds in China," 2009 Second International Workshop on Knowledge Discovery and Data Mining, Moscow, 2009, pp. 385-388.

[22] C. C. M. Chen, H. Schwender, J. Keith, R. Nunkesser, K. Mengersen and P. Macrossan, "Methods for Identifying SNP Interactions: A Review on Variations of Logic Regression, Random Forest and Bayesian Logistic Regression," in IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 8, no. 6, pp. 1580-1591, Nov.-Dec. 2011.

[23] R. Serban, A. Kupraszewicz and G. Hu, "Predicting the characteristics of people living in the South USA using logistic regression and decision tree," 2011 9th IEEE International Conference on Industrial Informatics, Caparica, Lisbon, 2011, pp. 688-693.

[24] R. Ksantini, D. Ziou, B. Colin and F. Dubeau, "Weighted Pseudometric Discriminatory Power Improvement Using a Bayesian Logistic Regression Model Based on a Variational Method," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 2, pp. 253-266, Feb. 2008.

[25] L. Wu and M. Li, "Applying the CG-logistic Regression Method to Predict the Customer Churn Problem," 2018 5th International Conference on Industrial Economics System and Industrial Security Engineering (IEIS), Toronto, ON, 2018, pp. 1-5.