Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 70 (2015) 632 – 639

Procedia Computer Science

4th International Conference on Eco-friendly Computing and Communication Systems, ICECCS 2015

A Novel Way of Assessing Software Bug Severity Using Dictionary of Critical Terms

Gitika Sharma[a,], Sumit Sharma[a], Shruti Gujral[a]

[a] *Department of Computer Science, Chandigarh University, Gharuan Mohali and 140413 INDIA*

Abstract

Due to increase in demands of software and decreased delivery span of software, assuring the quality of software is becoming a challenge. However, no software can claim to be error free due to the complexity of software and inadequate testing. There is a well-known principle of testing, which states that exhaustive testing is impossible. Hence, maintenance activities are required to ensure smooth functioning of the software. Many open source software provides bug tracking systems to aid corrective maintenance task. These bug tracking systems allow users to report the bugs that are encountered while operating the software. However, in software maintenance, severity prediction has gained much attention recently. Bugs having higher severity should be fixed prior to the bugs having lesser severity. Triager analyzes the bug reports and assesses the severity based upon his/her knowledge and experience. But due to the presence of a large number of bug reports, it becomes a tedious job to manually assign severity. Thus, there is growing need for making the whole process of severity prediction automatic. The paper presents an approach of creating a dictionary of critical terms specifying severity using two different feature selection methods, namely- info gain and Chi square and classification of bug reports are performed using Naïve Bayes Multinomial (NBM) and K-nearest neighbor (KNN) algorithms.

Keywords: Bug, Bug Tracking System, Bug Triaging System, Feature selection methods, Machine Learning Algorithms.

1. Introduction

Software is influencing a large sphere of human activities and its usage is increasing at a tremendous rate. Due to increase in demand of decreased delivery time ensuring quality while decreasing delivery time is becoming critical. Therefore, to ensure quality in software, different testing techniques are used [1]. However, in spite of vigorous testing, the probability of latent bug existence in the software can- not be discarded. It may be encountered while operating the product either under test or while in use. The software bugs that are detected after the deployment of software affect reliability and quality of the software. Bug tracking systems (BTS) allow users as well as developers

doi:10.1016/j.procs.2015.10.059

to report these bugs of software. It may allow identification and solution of more bugs and hence improving the overall quality of the software produced [2]. Example of such BTS are Bugzilla, Fog, ikiwiki and jira etc[3].

The reported bugs in BTS are analyzed by Triager to calculate their validity, correctness, importance, severity and also to verify its duplicity and hence are assigned to the relevant developer to resolve it. Triager is the person who uses his knowledge and experience to analyze and refine the bugs that are reported. This process is called bug triaging process and is shown in Fig.1.

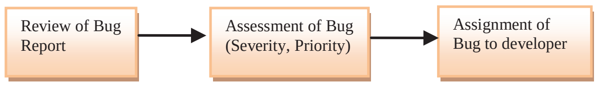
Review of Bug Report
Assessment of Bug (Severity, Priority)
Assignment of Bug to developer

Fig.1. Bug Triaging Process

As reported in literature [4] [5] [6], the bugs filed in these bug tracking systems are large in number. Therefore, developer has to make a choice amongst all reported bugs to resolve. Bug severity is one of the important factors which can be used to identify bugs that need immediate attention to resolve. Severity of a bug is defined as impact of the bug on functionality of software [7]. But, it is a very tedious job and time consuming to manually assign severity in case of a large number of bug reports. Also, accuracy of classification depends on knowledge and experience of triager who analyzed the bug.

Thus, there has been a growing need of automating the whole process of severity prediction of bugs to make bug triaging process more efficient and less time consuming. Therefore, we need to automate the task of classifying bugs based on severity. In past studies, text mining and machine learning algorithms have been applied by many researchers for automation of the bug triaging process [8] and for detection of duplicity in reported bugs [9]. But till now it is still far from reliable accuracy and there is still scope for improvement. Thus, an attempt is made in this paper to automatically predict the bug severity classification with increased reliability and accuracy. The paper is based on the hypothesis that some critical terms used for specifying the bug severity in the summary of bug. report could serve as severity indicator terms. These terms could help in making a dictionary of terms for automatic classification of bug reports based on bug severity.

The rest of the paper is organized as follows: Section II provides related research work. The proposed approach for severity classification is described in Section III. Section IV provides results obtained in experiment and discussion of it. Section V discusses the conclusion of the followed approach and the future scope.

2. Related Research Work

Bug triaging helps in deciding what to do with the reported bugs. Due to the extremely large number of bugs being reported in BTS, their classification during triaging is a tedious and time consuming process. Researchers have been for a long time trying to automate the bug triaging task. Decent success has also been achieved. A list of such works is thoroughly discussed in this section as follows:

First attempt to automate bug triaging process was made by Davor Cubanic, Gail C. Murphy [10], they proposed to use supervised machine learning (ML) algorithm for the prediction of developer to whom bug should be assigned. John Anvik et al. [11] extended the same work of Davor Cubanic et al. and used some different algorithms for supervised learning such as Support Vector Machine (SVM), Naïve Bayes and C4.5. John Anvik [12] further extended his previous work [11] and created a recommender for assigning the bug reports. Tim Menzies et al. [13] proposed a new automated method called SEVERityISsue assessment (SEVERIS) to assign a severity level to bug reports. It was based on the text mining approach and machine learning techniques.

Ahmed Lamkanfi et al. [6] proposed a new method for classifying bugs based on severity. Bug reports of Eclipse, GNOME and Mozilla were pre-processed using text mining algorithms and naïve Bayes classifier was applied. K. K. Chaturvedi et al. [14] presented machine learning approaches to determine severity level. Different machine learning approaches, namely SVM, RIPPER, naïve Bayes, J48 and naïve Bayes Multinomial and K-nearest neighbor were applied on bug reports of IV & V project of NASA.

Jifeng Xuan et al. [15] suggested a model for prioritizing developers. This model ranks the developers, according to their contribution. Support vector machine and naïve Bayes were used for the evaluation of performance of the model. Neelofar et al. [16] proposed an approach that helps to decide which bug should be assigned to which developer. Bugs were classified with different labels using summary of bug reports. Two different approaches called Chi square and Term frequency-Inverse document frequency (TF-IDF) algorithms for feature selection were used. Yuan Tian et al. [4] proposed a new approach to recommend the priority level of bugs using machine learning algorithms. N. K. Nagwani et al. [17] presented an approach for creating taxonomic terms that classify bugs. In this approach Latent Dirichlet Allocation (LDA) technique was used to find taxonomy terms.

From the conducted literature survey, it is concluded that early detection and classification of bugs is critical for maintaining the quality of software. Although, many attempts has been made at solving problem and several machine learning and text mining algorithms have been applied for the same, a novel idea of using dictionary of critical terms (terms specifying severity level) for the prediction of severity level of reports is proposed in this paper.

3. Proposed Approach

The whole process of severity classification which is proposed in this paper can be summarized as follows: The detailed methodology of the experiment is shown below in Fig.2.

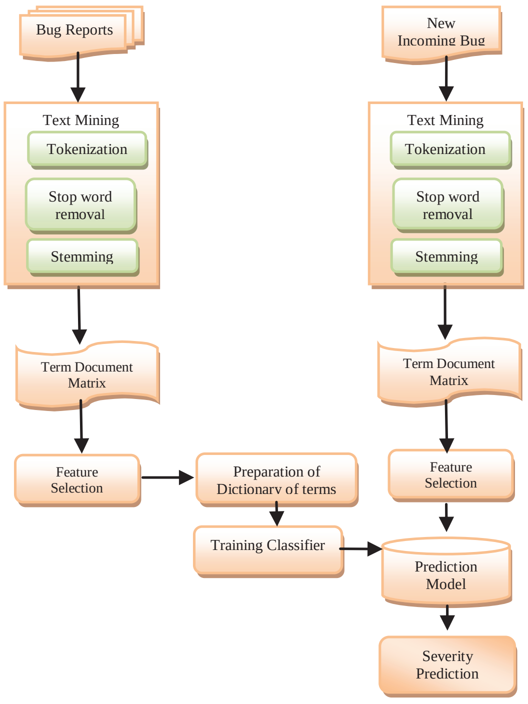

Fig.2.Bug Severity Prediction Methodology

The description of steps followed in methodology is provided below:

3.1. Dataset Acquisition

The experiment considers bug report instances of Eclipse to perform classification of bugs based on severity. Eclipse is an open source integrated development environment (IDE). It is used worldwide by different developers to develop software. It is expected that bug reports of Eclipse should be of good quality [18] as users of Eclipse are developer themselves and they will use technical terms for defining bug report. This will help in this study to create a more accurate dictionary of terms for specifying the bug severity level. Bug report instances of Eclipse are downloaded from Bugzilla [19] bug repository.

The types of severity in instances are blocker, critical, Enhancement, major, minor, normal, trivial. The report instances that have normal and enhancement severity type are not considered in the experiment. Enhancement bug reports do not correspond to the real bug as these are requests for new features. Normal bug reports require manual inspection to assess the severity as these bug reports represents a gray zone to classify bugs report into severe or non severe categories [6].

Severity of bug reports is classified into binary levels as severe and non severe because as it was found that increase in class level of severity, the performance of classification degrades [14]. Therefore the severity level of critical, blocker and major are considered as severe while the severity level of minor and trivial are considered as non severe. This experiment considers bug report instances of four components of Eclipse .These components are UI, SWT, debug and core. The datasets considered in this experiment are given in Table 1.

UI component of Eclipse represents the interface of the IDE and debug component is the component that relates to all debugging activities of the program. SWT component of Eclipse is abbreviated for standard widget tool that relates to all widgets used in developing software in Eclipse and core component of Eclipse represents the main infrastructure of IDE which includes compiler, model for API, code select and assess support etc.

Table 1: Bug report instances of components of Eclipse

Component	Severe Bug report	Non Severe Bug report
Core	1514	1487
Debug	1514	1400
SWT	1618	1393
UI	1764	1432

As the components taken for study represents Eclipse at an abstracted level. Hence, the model generalised for classification of reports of these components may be used for classifying reports of any another component of Eclipse.

3.2. Pre-processing

Pre-processing steps on textual summary of bug reports are performed. The reason for including summary as a prediction of severity level is that in [20], the authors concluded that summary of bug report gives better result than a detail description of bug report. It includes tokenization, stop word removal and stemming [21]. These are explained as below:

3.3. Term -Document matrix

The Term- Document Matrix is created after performing pre-processing steps. Each column in the matrix represents the terms occurring in documents and row represents each bug report. The cells of the matrix are filled with TF-IDF score. If a term is not present in the particular bug reports then that cell is filled with zero value. Term Frequency-Inverse Document Frequency [22] score is generally being used to give weight to each term. TF-IDF is calculated by multiplying term frequency with inverse document frequency and is given as:

$$\mathrm{TF-IDF} = n_w^d \, log_2 \left(\frac{N}{N_w}\right) \tag{1}$$

Where n_w^d = frequency of word w in document d, N is total document and N_w are documents containing word w.

3.4. Feature Selection

Feature selection methods are used to retrieve the most informative terms from a corpus of the matrix. In our research, we have used two feature selection methods info gain [23] and Chi square [24]. These methods are applied on TDM matrix to reduce the size of matrix.

3.5. Creating a dictionary of terms

The terms obtained after applying feature selection are sorted in descending order according to their weights. The top k- terms are used for creating a dictionary. The dictionary contains the terms that help in specifying the severity levels of each bug report. In this paper top 125 terms are used for the creation of the dictionary. The reason for selecting only 125 terms is that as in literature [14], it was found that top 125 terms are appropriate for model building.

3.6. Training and Testing using ML Algorithms

Rapid Miner is used to train KNN and NBM with TDM matrix of dictionary of terms as an input. Training and testing is done by using 5-fold cross validation approach. The input dataset is partitioned into 5 subsets, a single subset is considered as testing dataset and remaining four subsets are taken as training datasets. The cross validation process is repeated 5 times considering each of the subsets as training dataset. Then five results obtained after five iterations are averaged and single result is calculated. Performance of classifier is analyzed and compared on the basis of two performance metrics named accuracy and precision.

4. Result and Discussion

In this research, component specific dictionaries are created of four components of Eclipse. These dictionaries are created using major contributory top 125 terms using two feature selection methods; namely-info-gain and Chi square. The set of dictionary terms is then fed to two widely used ML algorithms named Naïve Bayes and KNN for classification task and performance is analyzed in terms of precision and accuracy. The results are documented below.

4.1. Probability based classification

The mostly used classification techniques in early text classification systems were probabilistic ones. These techniques are based on the assumption of conditional independence of term occurrence. For probability based classification NBM classifier is used in this approach. The performance of this classifier using two approaches of feature selection is evaluated and listed below:

4.1.1 Preparation of dictionary using info- gain– The results obtained after using classifier and info gain is shown in table 2. The accuracy varies from 69 % to 75 % while the precision of severe and non-severe category is found to be varying from 65-69 % and 77-84% respectively. The reason of less precision of the severe category than the non-severe ones could be attributed to the fact that the terms extracted by info gain have more bits of information than the severe class level in the datasets used in our experiments. In this approach of creating dictionary using info- gain and NBM classifier, the performance of classification of severity of bugs is found to be best.

Table 2: Results of info-gain and Naïve Bayes Multinomial

Component	Accuracy	Precision (Severe)	Precision (Non Severe)
SWT	69.67%	66.77%	78.03%
Debug	71.66 %	69.18%	76.40%
Core	72.33%	68.69%	78.03%
UI	**75.38%**	**71.21%**	**84.52%**

4.1.2 Preparation of dictionary using Chi square - The performance of classification of bug reports of four datasets after using the Chi square method and the naïve Bayes Multinomial classifier is shown below in table 3. The accuracy of this experiment lies between 68 % and 74 %. While the precision of severe class ranges from 65 % to 70 %, the precision of the non-severe class is found to be between 77 % to 84 %.The similar pattern of performance is achieved in these datasets using Chi square as achieved in info- gain component. The performance of bug severity classification of SWT component has least accuracy while bug severity classification of UI component has highest accuracy.

Table 3: Results of Chi square and Naïve Bayes Multinomial

Component	Accuracy	Precision (Severe)	Precision (Non-Severe)
SWT	68.04%	65.19%	77.33%
Debug	70.73%	68.00%	76.37%
Core	71.23%	68.26%	75.60%
UI	**74.32%**	**69.92%**	**84.81%**

4.2. Similarity based classification

The other classification technique used in text classification is based on similarity between the two documents. If the two documents have high similarity than these two have high possibility to be of one class. For similarity based classification KNN is used to retrieve its nearest neighbor for classification. It assigns majority class of K nearest neighbor to unlabeled document. The performance of this classifier using two approaches of feature selection is explained below:

4.2.1 Preparation of dictionary using Info- gain- The result is obtained using dictionary formed by info-gain and KNN classifier and shown below in Table 4. The accuracy ranges from 87 % to 91 %. The precision of severe class and non-severe class is obtained as lying from 91 % to 96 % and 79 % to 90 % respectively. UI achieves maximum accuracy of 91 % using the nearest neighbor classifier.

Table 4: Results of info-gain and KNN

Component	Accuracy	Precision (Severe)	Precision (Non Severe)
SWT	87.51%	96.97%	79.62%
Core	85.13%	84.81%	85.47%
Debug	88.32%	91.07%	85.45%
UI	**91.16%**	**91.83%**	**90.39%**

4.2.2 Preparation of dictionary using Chi square- The performance of classification task after using Chi square method and KNN classifier is shown below in Table 5.

Table 5: Results of Chi square and KNN

Component	Accuracy	Precision (Severe)	Precision (Non-Severe)
SWT	87.92%	97.39%	80.01%
Core	86.97%	86.39%	87.58%
Debug	88.91%	**92.32%**	85.49%
UI	**91.59%**	88.86%	**95.60%**

The accuracy of this experiment lies between 87 % and 91 %, precision of severe class ranges from 86 % to 92 %, whereas precision of non-severe class has range from 85% to 95 %. The similar pattern of performance is achieved in these datasets using Chi square as achieved in info- gain. SWT component has least accuracy and UI component achieve more accuracy.

Results show that four combinations of feature selection methods and classifier gave same pattern of accuracy. SWT achieves least and UI component achieves maximum performance in terms of accuracy and precision. However, the performance of the core and debug component show different pattern in both approaches. In probability based classification the debug component has higher accuracy than core component, whereas core component has higher accuracy than debug component in similarity based approach.

4.3 .Comparison of Probability and Similarity based classification

Fig.3 shows the comparison of accuracy level obtained using NBM and KNN using info- gain as a feature selection method. The maximum accuracy obtained with KNN and NBM is 91 % and 75 % respectively. The comparison of the accuracy of NBM and KNN using Chi square for dictionary formation of different component is also shown in Fig.4. It is again found that KNN performs better than NBM in terms of accuracy. The minimum accuracy of NBM is 68 % and maximum accuracy is 74 % whereas the minimum accuracy of KNN classifier is 84 % and maximum is 91 % which are much greater than those achieved using NBM classifier

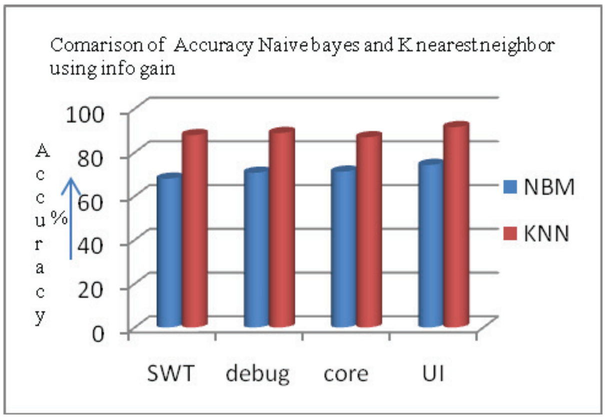

Fig.3. Comparison of accuracy using info- gain

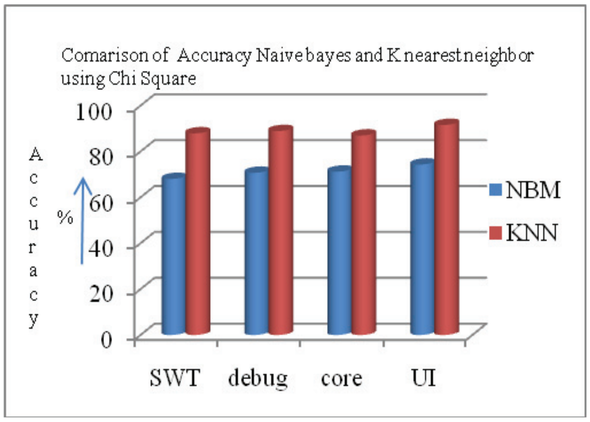

Fig.4. Comparison of accuracy using Chi square

Hence, it can be concluded from the results that KNN performs better for bug classification severity under the given experimental set up. The probable reason behind the better performance of KNN can be attributed to the fact that useful pre-processing steps were applied to the raw dataset extracted from the Bugzilla repository. These pre-processing steps and the process of dictionary formation further refined the datasets. This enables KNN to efficiently classify bugs as it is a well known fact that KNN performs best when noise free data is fed to it.

5. Conclusion and Future Work

Automation of predicting the severity level of bug reports is an emerging issue. Prediction requires historical data for finding out critical terms. The classification task is done by two ML algorithms named as NBM and KNN and on the basis of accuracy and precision metrics their performance is compared. It has been concluded that under used experimental conditions KNN performs better for bug severity classification. The KNN algorithm works on the principle of Euclidean distance while the NBM works on the principle of conditional initial probabilities. As the probabilities are calculated during training in NBM, the classifier classifies each data points on the basis of the same initial probabilities while the KNN classifies each data point on the basis of its current distance from its neighbors. Since, this problem requires scores calculated by text mining, each data point may have common values of attributes which is actually not utilized in the NBM.

The work in this paper only proposes to implement the proposed approach on an offline database downloaded from the Bugzilla repository and is not designing any such automated system for online classification. Therefore, a study can be conducted to make the system online for real time classification of bug reports.

References

1. Beizer, Boris. Software testing techniques. *Dreamtech Press*, 2003.
2. Raymond, Eric.The cathedral and the bazaar. *Knowledge, Technology & Policy 12*; 1999. no. 3. 23-49.
3. “15 Most Popular Bug Tracking Software to Ease Your Defect Management Process”, http://www.softwaretestinghelp.com/popular-bug-tracking-software/”, Feb 12, 2015.
4. Tian, Yuan, Daniel Lo, and Chengnian Sun. Drone. Predicting priority of reported bugs by multi-factor analysis. *In Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. *IEEE ;*2013 , pp. 200-209.
5. Ahsan, Syed Nadeem, Javed Ferzund, and Franz Wotawa. Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine..*In Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on,* . *IEEE*; 2009.pp. 216-221.
6. Lamkanfi, Ahmed, Serge Demeyer, Emanuel Giger, and Bart Goethals. Predicting the severity of a reported bug. *In Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pp. IEEE; 2010. pp.1-10.
7. “Defect severity classification in software testing (with an example)”, *http://www.zyxware.com/articles/3559/defect-severity-classification-in-software-testing-with-an-example,* May 24, 2013.
8. Alenezi, Mamdouh, Kenneth Magel, and Shadi Banitaan. Efficient bug triaging using text mining. *Journal of Software 8;2003. no. 9*.
9. Runeson, Per, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. *In Software Engineering, 2007. ICSE 2007. 29th International Conference on., IEEE ;* 2007.pp. 499-510.
10. Murphy, Gail C., and D. Cubranic. Automatic bug triage using text categorization. *In Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*; 2004.
11. Anvik, John, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug?. *InProceedings of the 28th international conference on Software engineering,. ACM* ;2006. pp. 361-370 .
12. Anvik, John Karsten. Assisting bug report triage through recommendation. *PhD diss., University of British Columbia* ; 2007.
13. Menzies, Tim, and Andrian Marcus. Automated severity assessment of software defect reports.*In Software Maintenance, 2008. ICSM* 2008. *IEEE International Conference on, IEEE*;2008.pp. 346-355.
14. Chaturvedi, K. K., and V. B. Singh. Determining bug severity using machine learning techniques. *In Software Engineering (CONSEG), 2012 CSI Sixth International Conference on*, *IEEE*; 2012, pp. 1-6.
15. Xuan, Jifeng, He Jiang, Zhilei Ren, and Weiqin Zou. Developer prioritization in bug repositories. *In Software Engineering (ICSE), 2012 34th International Conference on. IEEE;* 2012 *pp. 25-35*.
16. Javed, Muhammad Younus, and Hufsa Mohsin. An Automated Approach for Software Bug Classification. *In Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on ,IEEE;* 2012. pp. 414-419.
17. Nagwani, N. K., Shalini Verma, and Krunal K. Mehta. Generating taxonomic terms for software bug classification by utilizing topic models based on Latent Dirichlet Allocation. *In ICT and Knowledge Engineering (ICT&KE), 2013 11th International Conference on, IEEE* ; 2013.
18. Bettenburg, Nicolas, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann.What makes a good bug report?. *InProceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, ACM* ; 2008 pp. 308-318.
19. Bugzilla: https://bugs.eclipse.org/bugs/ [Last Access-6/21/2015].
20. Lamkanfi, Ahmed, Serge Demeyer, Quinten David Soetens, and Tim Verdonck. Comparing mining algorithms for predicting the severity of a reported bug. *InSoftware Maintenance and Reengineering (CSMR), 2011 15th European Conference on, IEEE;* 2011pp. 249-258.
21. Porter, Martin F. An algorithm for suffix stripping. *Program 14 ;* 1980. no. 3 pp. 130-137.
22.Aizawa, Akiko. An information-theoretic perspective of tf–idf measures. *Information Processing & Management* ; 2003. pp 45-65.
23. Yang, Yiming, and Jan O. Pedersen. A comparative study on feature selection in text categorization. *In ICML* ; 1997.vol. 97, pp. 412-420.
24. Dixon, Wilfrid Joseph, and Frank Jones Massey. Introduction to statistical analysis. *New York: McGraw-Hill* ; 1969 .vol. 344.
25. Rapid miner tool: https://rapidminer.com/ [Last Access- 621/2015].