Self Admitted Technical Debt Detection and Classification Using Text Mining and Machine Learning Approach

[1]K.Ravikumar, [2]Dr.G.Gunasekaran,
[1]Research Scholar, Dept. of CSE, Manonmaniam Sundaranar University,Tirunelveli Tamilnadu
[2]Principal, JNN Institute of Engineering, Chennai,Tamilnadu
[1]ravi_kumar2k3@yahoo.com
[2] gunaguru@yahoo.com

ABSTRACT

The efficiency in software development is mandate as bug fixes is the major task to be performed. But the deadlines makes the developers to adopt faster methodologies to meet the requirements in the appointed time. So better solution are compromised due to time factor. The Technical Debts are the quick choice but redesign is the penalty they pay. Source code comments are inserted by developers known as Self-Admitted Technical Debt (SATD) to indicate the future task. SATDs detected from source code comments is a challenging task and in literature several methods are performed. In this paper we present the detail investigation of the methods to detect SATD and a new approach is proposed to detect the SATD through text mining of source code comments. The training set are used from the available database and example codes. The analysis shows that the proposed method improvise the performance of detecting and classifying the SATD using text mining and machine learning approach.

Keywords: SATD, text mining, machine learning, Technical debt, comments, source code, SVM, Neural network

I INTRODUCTION

Software development in recent years took less time and more efficient. But the software faces bug issues [Lim et al 2012]. Managing the bugs is a important module when there is an issue at the customer end [Allman, 2012]. The cost increase as the redesign and reconfiguration of software happens in future due to technical debt[Zazworka et al, 2013]. The dept is fixed by Self-Admitted Technical Debt (SATD). Combining text mining and machine learning approach will improve the performance. To speed up-development of software the temporary solution adopyed was the Technical debt which leads to higher cost later on. Recently, In literature methods were designed to detect the technical debt from text mining of the source code comments. The classification by different methods of self-admitted technical debt was used which also removes the same. But there is no empirical evidence on software quality degradation due to SATD. Therefore, self-admitted technical debt and software quality has to be investigated and removed. The analysis of files to detect self-admitted technical debt is a long process though several methods were investigated in literature [Poldar, 2014]. The entropy of the defect on files has to be calculated[Kruchten, 2013] .The debts are detected using several methods like text mining [Xia et al, 2014], word embedding [Flisar and V. Podgorelec, 2019], Vocabulary Models[Farias et al,2015] and others [9-13]. Once the detection is completed the next will be the classification and removal. For classification several models were used from past [McCallum A, Nigam K et al (1998) to recent years [4].

2. BACKGROUND METHODOLOGY

Examining the Impact of Self-admitted Technical Debt is been done by several methodologies. Evaluation of the debt improves the software quality. The source code comments are been utilized in the analysis of debts through text mining, embedded analysis and pattern matching. Among this, the paper focuses on the text mining. The comments are organized, detected and classified using machine learning approaches.

2.1. Text Mining

Text mining is a simple baseline approach where unstructured comments data is made structured pattern. The text mining techniques convert the code comments into patterns. choose a target text/code/project for prediction. The comments are further integrated into one single dataset. In the method four open source software projects - Eclipse, Chromium OS, Apache HTTP Server, and ArgoUML are used to identify self-admitted technical debt. Some of the patterns arrived from the comments used in the work is given in table 1.

Table 1 Patterns integrated from the code comments

Patterns that indicates SATD	
Hack	Ugly
Nuke	Barf
Yuck	Crap
Hacky	Silly
Fixme	Stupid
Kludge	Kaboom
give up	toss it
Retarded	bail out
at a loss	take care
this is bs	causes issue
prolly a bug	this is wrong
fix this crap	Inconsistency
is problematic	don’t use this

The text mining based SATD detection and classification is shown in figure 1.

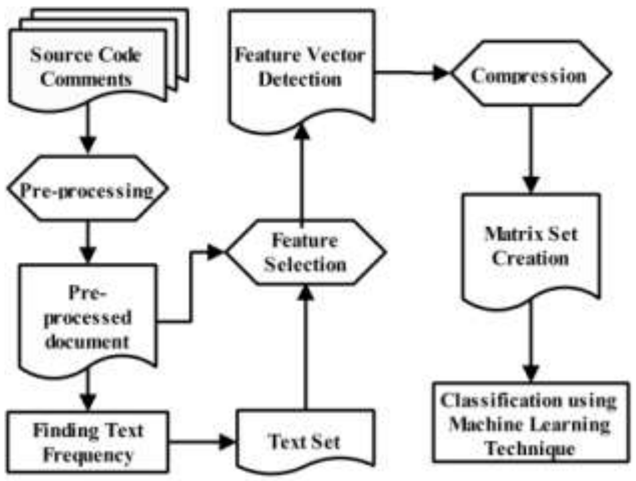

Figure 1. Block Diagram of text mining based SATD detection and classification

The code is processed in the text preprocessing stage which contains the stages like Tokenization, Stop-word Removal and Stemming. The process converts the stream of text into words and symbol, removal of stop words like “the” , “to” ,“I”, “should” etc and finally convert the word to its root form. The next stage is feature extraction and selection which enhances the efficiency of the classifier. The features are differentiated to categorize into SATD relavant and non SATD. In literature Information Gain is used to select the features which represent the number of bits for each pattern label.

In existing methods SVM, KNN and NBM are used.

The comments are labeled as

$C = \{(C1, L1), (C2, L2), ..., (CN, LN)\}$,

where Ci represents the *ith* comment and Li is a label denotes "SATD" or "notSATD" ($\bar{t}$), and the word vector of Ci is denoted as $Ci = \{w1, w2, ..., wn\}$, where n represents the number of different words appeared in Ci and wi represents the *ith* word.

For a feature (i.e., word) w and a comment Ci , there would be 4 possible relationships:

1. (w, t): comment Ci contains the feature w, and it is a SATD comment (i.e., t).
2. $(w, \bar{t})$: comment Ci contains the feature w, but it is not a SATD comment (i.e., $\bar{t}$).
3. $(\bar{w}, t)$: comment Ci does not contains the feature w, but it is a SATD comment (i.e., t).
4. $(\bar{w}, \bar{t})$: comment Ci does not contains the feature w, and it is not a SATD comment (i.e., $\bar{t}$).

The inputs to the classifier is the integrated comments pattern feature sets. In existing method Naive Bayes Multinomial (NBM), Support Vector Machine (SVM) and k-Nearest Neighbor (kNN) are used for classification techniques

The labels are predicted in the classifier stage some of the labels are given in table.2

Table 2 : An example of classifiers set

Classifier set	Predicted Label
set 1	SATD comment
set 2	Without SATD
Composite classifier	SATD may be

2.2.Types of Machine learning approaches

Since most of SATD detection and classification need manual orautomated classifiers, designing a method for the same is required. The different classifiers investigated is given below.

a. SVM CLASSIFIER

The SVM method was familiar binary classification method used for prediction and classification. In this Being a binary classifier, dealing with too many classes will reduce the predictive power of SVM. Support vector machines map the training SATD and non SATD comment data into a higher-dimensional feature space. A hyperplane (decision surface) is then constructed in this feature space that bisects the two categories and maximises the margin of separation between itself and those points lying nearest to it (called the support vectors). This decision surface can then be used as a basis for classifying vectors of unknown classification.

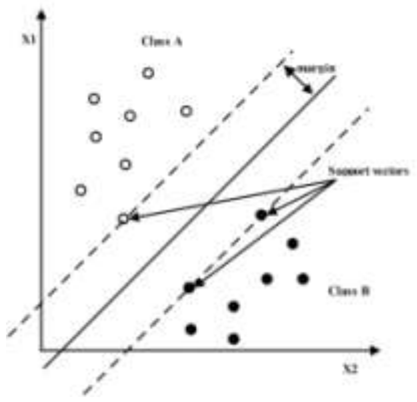

Figure 2: Maximum-margin hyperplane for a SVM classifier. Samples on the margin are called the support vectors.

b. KNN CLASSIFIER

K nearest neighbor(KNN) is a simple algorithm, which stores all cases and classify new cases based on similarity measure.KNN algorithm also called as 1) case based reasoning 2) k nearest neighbor 3)example based reasoning 4) instance based learning 5) memory based reasoning 6) lazy learning .KNN algorithms have been used since 1970 in many applications like statistical estimation and pattern recognition etc.KNN is a non parametric classification method which is broadly classified into two types1) structure less NN techniques 2) structure based NN techniques. In structure less NN techniques whole data is classified into training and test sample data. From training point to sample point distance is evaluated, and the point with lowest distance is called nearest neighbor. Structure based NN techniques are based on structures of data like orthogonal structure tree (OST), ball tree, k-d tree, axis tree, nearest future line and central line .Nearest neighbor classification is used mainly when all the attributes are continuous.

NBM CLASSIFIER

By default, we train each sub-classifier using the Naive Bayes Multinomial (NBM) technique, which is widely used in text mining (McCallum et al. 1998). Before introducing NBM, we first introduce Naive Bayes (NB) (McCallum et al. 1998). The major advantage of NB classification is its short computational training time, since it assumes that given a label (i.e., with or without SATD), features (i.e., words) are conditionally independent.

Notice that in NB, we only consider the presence or absence of a feature in a comment. NBM is similar to NB, but the label is determined by the number of times each feature appears in the comment. In general, when the total number of unique features in the comment collection is large, NBM may perform better than NB (Xia et al. 2014).

2.2. open source project dataset: The projects used for dataset are given in table 3. The data was contributed by several authors and the domain are mentioned in the table.

Table 3 Summary of projects in our dataset

Project	Domain	Contributors	LoC	Comment Ratio
ArgoUML	UML Modeling Tool	87	926K	High
Columba	Email Client	10	155K	High
Hibernate	ORM Framework	314	703K	Low
JEdit	Text Editor	57	310K	Average
JFreeChart	Char Library	19	317K	High
JMeter	Performance Tester	41	354K	Average
JRuby	Ruby Interpreter	374	841K	Low
SQuirrel	SQL Client	40	708K	Average

3. PROPOSED METHODOLOGY

The block diagram of the proposed method is presented in Figure.3. The text mining consists of test preprocessing, feature selection and classification. In classification Binary Tree SVM architecture is used . The structure comprices of 4 SVM in parallel to indicate several combinations like 00,01,10,11- The combination 00 and 11 are doubtful situation where the feature is considered relevant to SATD doubt and SATD not. In existing SVM, being a binary classifier cant able to classify the additional category. Binary Tree SVM classifier SVM1, SVM2, SVM3, SVM4, classifies Comment set 1, set 2 , set 3 and set 4 respectively. The performance is given in Table 4 for classification of the features. The binary tree Support Vector Machine (SVM) has self-learning features, dimensionally and space independent. SVMs are feed forward network with a single layer of non-linear units show accurate results with high efficiency. In addition the SVM is proved to have minimized the structural risk for implementation. SVM minimizes the bound on the errors made by the learning machine over the data used for testing. The objective function of the training datasets can't be minimized. The SVM can able to classify perfectly the text comment features that do not belong to training data. Classifications of data in training are done using support vectors. For difficult data set the SVM places the hyper plane which separates the difficult data into two classes using support vectors. The features obtained from the previous stage are classified using the proposed classifier.

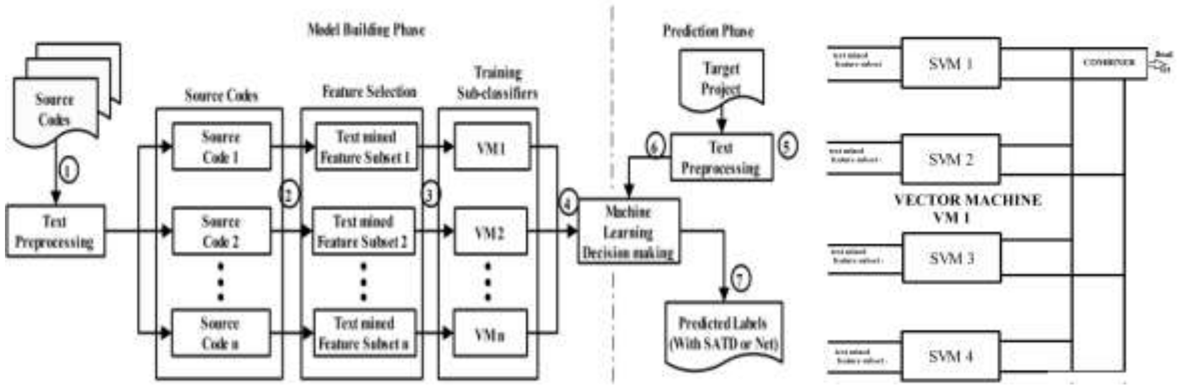

Figure 3. block diagram of the proposed method (a) and the structure of VM using Tree SVM.

Table 4 : Model prediction time (in seconds) for different data set

App	Argo.	Col.	Hib.	JE.	JF.	JM.	JR.	SQ.
Tree SVM	0.048	0.022	0.063	0.032	0.020	0.015	0.015	0.008
Pattern	0.064	0.047	0.094	0.001	0.001	0.016	0.015	0.016
NBM	0.142	0.078	0.036	0.048	0.015	0.016	0.016	0.016
SVM	5.727	5.243	3.201	6.053	3.121	5.414	4.383	6.120
kNN	35.195	28.098	25.589	28.539	18.134	32.161	23.543	27.880

4. CONCLUSION AND FUTURE WORK

The paper presents the investigation of existing methods on the SATD detection and classification. A new method based on Machine learning approach and text mining is used to detect and classify the SATD. The analysis shows that the prediction time is better in the proposed design. The testing of the methods are carried out using open source projects. The implementation shows the effectiveness of the proposed method when compared to the existing method.

In Future, the work will be extended towards implementing the software in a stand-alone hardware. The tree SVM method will be combined with optimization technique to enhance the performance. Now hybrid methods will be proposed to minimize the software development and estimation cost.

REFERENCES

[1]. McCallum A, Nigam K et al (1998), “A comparison of event models for naive bayes text classification”, In: AAAI-98 Workshop on learning for text categorization. Citeseer, vol 752, pp 41–48.

[2]. Xia X, Lo D, Qiu W, Wang X, Zhou B (2014), “Automated configuration bug report prediction using text mining”, In: 2014 IEEE 38th annual computer software and applications conference (COMPSAC). IEEE, pp 107–116.

[3]. Potdar, Aniket & Shihab, Emad. (2014). “An Exploratory Study on Self-Admitted Technical Debt”, Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014. 91-100.

[4]. J. Flisar and V. Podgorelec, "Identification of Self-Admitted Technical Debt Using Enhanced Feature Selection Based on Word Embedding," in IEEE Access, vol. 7, pp. 106475-106494, 2019, doi: 10.1109/ACCESS.2019.2933318.

[5]. Kruchten, Philippe & Nord, Robert & Ozkaya, Ipek & Falessi, Davide. (2013). Technical debt: Towards a crisper definition report on the 4th international workshop on managing technical debt. ACM SIGSOFT Software Engineering Notes. 38. 51-54.

[6]. Lim, Erin & Taksande, Nitin & Seaman, Carolyn. (2012). A Balancing Act: What Software Practitioners Have to Say about Technical Debt. Software, IEEE. 29. 22-27.

[7]. Zazworka, Nico & Spínola, Rodrigo & Vetro, Antonio & Shull, Forrest & Seaman, Carolyn. (2013). A Case Study on Effectively Identifying Technical Debt. ACM International Conference Proceeding Series. 42-47.

[8]. Allman, Eric. (2012). Managing Technical Debt. Communications of the ACM. 55. 50-55.

[9]. Guo, Yuepu & Seaman, Carolyn & Gomes, Rebeka & Cavalcanti, Antonio & Tonin, Graziela & Silva, Fabio & Santos, André & Siebra, Clauirton. (2011). Tracking technical debt — An exploratory case study. 528-531.

[10]. ST. Klinger, P. Tarr, P. Wagstrom, and C. Williams, "An enterprise perspective on technical debt," in Proceedings of the 2nd Workshop on Managing Technical Debt, MTD '11. New York, NY, USA: ACM, 2011, pp. 35--38.

[11]. Siebra, Clauirton & Tonin, Graziela & Silva, Fabio & Oliveira, Rebeka & Junior, Antonio & Miranda, Regina & Santos, André. (2012). Managing technical debt in practice: An industrial report. 247-250.

[12]. Spínola, Rodrigo & Zazworka, Nico & Vetro, Antonio & Seaman, Carolyn & Shull, Forrest. (2013). Investigating technical debt folklore: Shedding some light on technical debt opinion. 1-7.

[13]. Maldonado, Everton & Abdalkareem, Rabe & Shihab, Emad & Serebrenik, Alexander. (2017). An Empirical Study on the Removal of Self-Admitted Technical Debt. 238-248. 1

[14]. Da S. Maldonado, Everton & Shihab, Emad. (2015). Detecting and quantifying different types of self-admitted technical Debt. 9-15.

[15]. Farias, Mário & Mendonça, Manoel & Batista da Silva, André & Spínola, Rodrigo. (2015). A Contextualized Vocabulary Model for identifying technical debt on code comments. 25-32.