BFSp: A Feature Selection Method for Bug Severity Classification

Sadia Sharmin
and Farzana Aktar
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
Email: bit0426@iit.du.ac.bd
Email: bsse0627@iit.du.ac.bd

Amin Ahsan Ali
and Muhammad Asif Hossain Khan
Dept. of Computer Science and Engineering
University of Dhaka
Dhaka, Bangladesh
Email: aminali@du.ac.bd
Email: asif@du.ac.bd

Mohammad Shoyaib
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
Email: shoyaib@du.ac.bd

Abstract—**Bugs are inevitable in every software system. Resolving these bugs requires proper classification based on their severity. This is because the severe bugs need to be assigned to the developers immediately for fixing so that the vulnerability of software can be reduced. For this reason, several automated bug severity classification methods are already proposed to date that use the terms extracted from bug reports. However, most of the time they lack to identify the desired set of features that enhance the classification results. Besides, the existing works mainly focuses on classifying the bug severity within the different versions of a software. However, identifying the bugs in the newly released software is also crucial. To resolve these issues, we propose a method namely Bug Feature Selection method that mainly uses Pareto Optimality in order to find the most informative features. To evaluate the effectiveness of the proposed approach, we use the bug reports of three open source projects namely Eclipse, Mozilla and GCC. The experimental results show that our technique achieves promising performance compared to other state-of-the-art algorithms.**

I. Introduction

Software bug management is an important part of software maintenance process that requires a lot of cost and effort. Moreover, more than 90% cost is spent on its maintenance and evolution activities [1]. To reduce this cost and effort, the bugs need to be organized and fixed within a short period of time to improve the quality of the subsequent versions of the software. Usually, a bug tracking system (e.g., Bugzilla[1]) is used for the open source projects where the developers submit bug reports and mark their severity. However, with the increasing number of reported bugs, it becomes almost impossible to inspect them manually. Without proper automatic classification, managing a huge number of bugs leads to many difficulties such as delay in bug resolution and product release, surpassing project estimated cost.

To solve the aforementioned issues, in recent years extensive researches have been conducted in automated bug classification. Most of the existing methods of automating the bug classification process focus on using summary and description of the bugs in the bug report. These methods employ text mining approaches such as tokenization, stop word removal, word stemming as a preprocessing step along with different machine learning algorithms for classification[2], [3], [4], [5]. In [3], authors extracted the features from both summary and description of the bug reports mixing two projects: Mozilla and Eclipse. They built a set of 59 features for bug severity classification and compared it using different classifiers. They also claimed that incorporation of boosting algorithm further improved the classification results. Chaturvedi *et. al.* [4] also analyzed the results of several classifiers and concluded that satisfied output could be obtained when the number of features is more than 125 where the top k number of features are taken using information gain.

Mining bug reports or selecting terms for classification are also found in the existing literature. In [6], the authors presented a method to improve the classification of severe and non-severe bugs by creating a dictionary of important terms and using Naı̈ve Bayes Multinomial classifier. Another attempt where the authors in [7] obtained the dictionary terms applying information gain and χ^2 based feature selection and selected the top k (125) features. However, the above-mentioned approaches only consider the uni-gram model assuming all the terms independent which is not true for all cases. Addressing the problem with uni-gram model, Nivir *et. al.* [8] proposes a method using bigram model along with χ^2 feature selection. Experimental results on the Mozilla and Eclipse datasets reveal that adopting bigram model does not always increase the classification accuracy rather it may produce poor results depending on the datasets.

Instead of text feature selection, topic modeling is also performed for severity classification [9], [10]. In [9], the authors computed the textual similarity of bug reports using KL divergence and used K-Nearest Neighbor classifier for prediction of severity status. Topic modeling is also performed in [10]. These topics were used as additional feature of the REP algorithm, which is a similarity function proposed in [11] to calculate the similarity between bug reports. Based on the enhanced REP (i.e., REP topic), the authors utilized K-Nearest Neighbor classifier. The method produced reasonable accuracy for the dataset that represents the real world problems. However, they had to tune several different parameters.

[1]https://www.bugzilla.org/

Most of the existing methods that use feature selection select top k features. However, detecting the value of k might be troublesome for a new project. Furthermore, the terms in the bug reports are usually used as features and the number of terms is usually very large. Selecting the desired set of terms that employs less computation for large set is also a generic problem. Here the desired set implies the terms that are very relevant with classes and if one term fails to identify any class(s) then the other terms will be able to supply additional information for the identification. Most of the aforementioned methods were proposed to improve the performance of bug severity classification within a project. However, it is desirable that we have to identify the bug severity levels for newly released projects. Hence, to overcome the aforementioned issues, we propose a bug feature selection method named as Bug Feature Selection (BFSp) that mainly uses Pareto Optimality for selecting the desired set of features.

The main contributions of BFSp are:

- the methods includes text mining and feature selection that automatically predicts the severity of the bug reports.
- the feature selection method is based on relevance, χ^2 independence test and Pareto Optimality that selects a small number of informative features.
- the method is parameter free and require less computation compared to other methods.
- cross-project evaluation for severity prediction is initiated.

II. Proposed Method

In this section, we will describe how our proposed approach namely Bug Feature Selection (BFSp) mainly uses Pareto Optimality for predicting the severity levels of bug reports. The whole process is divided into two parts: feature extraction and feature selection.

A. Feature Extraction

Usually a bug report contains multiple attributes such as bug-id, bug summary, bug description etc. For classification, we can mainly use two types of information: bug summary and description. To extract features from these two, we first pre-process these textual information implementing the Natural Language Processing (NLP) techniques including tokenization, stop word removal and stemming. Tokenization is the process of separating each word from text as a token removing the punctuation marks. However, some words such as adverbs, verbs, preposition etc are not necessary for statistical analysis. For this reason we filter the tokens by removing these stop words. Also, a particular word can appear in the token list in different form with the same meaning (e.g., closed, closing, close). Therefore, we use porter stemming [12] to convert the words into their base form. After performing the pre-processing we create a Term Document Matrix (TDM) where each column represents a unique word/term and row represents each bug summary/description. The value of each cell in TDM denotes the frequency of the terms. .

B. Feature Selection

In TDM, all the terms are not equally important for bug classification. Rather some terms may be unnecessary that can degrade the performance of the classifiers. On the contrary, some terms may contain relevant information for classification. Thus, we have to collect a proper subset of terms that will improve the result (we will consider each term as a feature). To achieve this, we adopt mutual information (MI) based selection methods as it is proven to be classifier independent and has been applied in different domains [13], [14], [15], [16]. In MI based selection methods, the main objective is to find a subset of features S from a given set of features $T = \{t_1, t_2, \cdots, t_n\}$ which maximizes the value of $I(S;C)$ given in (1).

$$\begin{aligned} I(S;C) &= I(t_1, t_2, \cdots, t_k; C) \\ &= \sum_{t_1,\cdots,t_k} \sum_{C} P(t_1, t_2, \cdots, t_k; C) \log \frac{P(t_1, t_2, \cdots, t_k; C)}{P(t_1, t_2, \cdots, t_k) P(C)} \end{aligned} \tag{1}$$

Here, $I(S;C)$ presents the joint mutual information among the selected features S and class variable C. However, the computation of $I(S;C)$ in (1) is not feasible for implementation [17] as it is an NP-hard problem. Several approximations of joint MI are found in the literature [18], [19], [20]. However, the computation time for large number of features is still very high for these methods. Addressing this problem, we propose a feature selection method that is computationally less expensive. This selection method consists of two stages: Candidate Feature Selection and Final Feature Selection.

1) Candidate Feature Selection: We first consider the individual term t_i and calculates MI (relevance) for each term using (2).

$$J_{rel}(t_i) = I(t_i; C) \tag{2}$$

Here, t_i denotes the each term of TDM and $J_{MI}(t_i)$ is the mutual information between the term t_i and class C. However, there exist some error (bias) while calculating MI for finite number of sample [21]. Hence we adjust the bias value for MI following the method discussed in [22]. We furthermore use χ^2 test statistic to measure the statistical dependency between the term t_i and C. If the MI value of a term is greater than the critical value (χ^2_C), it implies that the term possesses strong relevance with the class C and hence, we select that term. Otherwise, the term is discarded. This process is described in Algorithm 1 which returns a set of candidate terms and their corresponding MI values.

Algorithm 1 : MI based Candidate Term Selection

```
Input: Set of Term $T = \{t_1, t_2, \cdots, t_n\}$, Class $C$
Output: Set of Candidate Term $T_c$ and $J_c = \{J_1, J_2, \cdots, J_n\}$
1: $T_c \Leftarrow \emptyset$
2: for each $t_i \in T$ do
3:     Select $t_i$ if $J_{MI}(t_i) > \chi^2_C$
4:     $J_i \Leftarrow J_{MI}(t_i)$ ; $T_c \Leftarrow T_c \cup \{t_i\}$
5: end for
6: Return $T_c$ with its $J_c$
```

TABLE I: Contingency Table

Class	Appear(1)	Not appear(0)	Class Total
c_1	f_{11}	f_{10}	M_{1+}
c_2	f_{21}	f_{20}	M_{2+}
$\cdots$	$\cdots$	$\cdots$	$\cdots$
c_k	f_{k1}	f_{k0}	M_{k+}
P/A Total	M_{+1}	M_{+2}	M

2) Final Feature Set Selection: In candidate feature selection process, we only consider the relevance of each term with class C. However, we have to select those features that have complementary information about the class variable C to achieve better classification result. Thus, to obtain a feature subset that possesses better relevancy and complementary information with less computation, we introduce Pareto Optimality (PO) in feature selection. It helps to find the features which optimize the solution with multi-criteria objectives. PO set is the set of criteria for which no other criteria has better performance and thus its members are said to be non-dominated [20]. Now, we have to choose feature criteria for PO in such a way that it can supply the desired feature subset containing the relevant and complementary information. In our method, we consider five criteria for PO in this regard which are defined as follows: The first criteria is MI value calculated in Algorithm 1. The second one is Inverse Document Frequency (IDF) defined in (3)

$$IDF(t_i) = \log \frac{N}{n_i} \tag{3}$$

where N is the total number of documents, and n_i denotes the number of documents containing the term t_i. n_i is also used as the third criteria. For calculating fourth and fifth criteria, let us consider the Table I where the first column represents k different classes and the second and third column represents the $Presence(P)/Absence(A)$ of a term in all bug reports. We get the fourth criteria calculating variance of vector $Q = \{q_1, q_2, \cdots, q_k\}$ where q_i is defined from Table 1 in (4).

$$q_i = \frac{f_{i1}}{M_{i+}} \tag{4}$$

For the last criteria we calculate variance of vector $P = \{p_1, p_2, \cdots, p_k\}$ where p_i is defined from Table I in (5).

$$p_i = \frac{f_{i1}}{M_{+1}} \tag{5}$$

Using these five criteria, the feature that appears in the Pareto Set are superior to other features that do not appear in this set and thus, we get the optimal feature set in Pareto sense.

Based on the aforementioned process, we select the features for bug severity prediction and thus, answer the following research question related to the software bug prediction.

RQ 1: ***How much the bug severity prediction result can be improved using minimal number of features?***

RQ 2: ***Which criteria between summary and description is more suitable for severity prediction?***

TABLE II: Comparison between the number of original features and the number of selected features using BFSp

	GCC		Eclipse		Mozilla	
	Original	Selected	Original	Selected	Original	Selected
Description	14741	292	23360	976	10225	499
Summary	2422	14	4554	112	3072	47

RQ 3: ***How much the result is dependent on different classifiers using the proposed method?***

RQ 4: ***Can the existing data sets be used for cross project classification?***

III. Result Analysis and Discussion

In this section we first present the dataset description and the experimental setting of our study and analyze the results of the proposed BFSp and then, provide the answers of the research questions presented in the previous section.

A. Dataset Description and Experimental Settings

To evaluate BFSp, we choose three open source projects namely, Eclipse[2], Mozilla[2] and GCC[2] from Bugzilla. We also follow the same training and testing protocol given in [10]. We thus use five severity labels of bug namely Blocker, Critical, Major, Minor, Trivial from the given seven labels where we exclude the labels namely normal and enhancement following the work [23], [10]. This is because enhancement is not a real bug and normal is the default option for selecting severity levels. For evaluation, we follow the process given in [24] by dividing the chronologically sorted dataset into eleven non-overlapping sets of equal size. Usually we want to identify the severity levels of version $m + 1$ given the data of version $1 \cdots m$ of a software. To mimic this, we train with set 0 and test with set 1 from the eleven separate dataset. Following this manner, we train with set $0-9$ and test with set 10 in the final iteration which is also done in [24]. We follow this evaluation protocol for all our results unless otherwise stated separately. In order to evaluate the effectiveness of the proposed model, we adopt F-measure [25] as a performance metric which is given in (6).

$$F - measure = 2 * \frac{\frac{TP}{TP+FP} * \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}} \tag{6}$$

Here TP, FP and FN are true positive, false positive and false negative respectively. In this paper, we use Support Vector Machine (SVM) with linear kernel and Decision Tree(DT) to classify the datasets.

B. Results and Discussion

We first find all unique terms for a given set of bug reports that is used as features for our selection method. Table II compares the number of features for each individual projects (using either description or summary) and the number of selected features using BFSp.

Answer to RQ 1: It is clearly observed that the number of selected features are much less than the original set (i.e.,

[2]https://github.com/ProgrammerCJC/SPFR

TABLE III: F-measure (%) for cross-project

Train	CF	Test	Blocker	Major	Minor	Critical	Trivial
Eclipse	4173	GCC	10.57	16.14	24.91	22.88	10.05
	3888	Mozilla	8.63	27.81	27.67	27.94	25.84
GCC	4173	Eclipse	14.33	20.15	27.7	23.06	10.27
	3420	Mozilla	8.02	13.59	40.5	20.97	10.49
Mozilla	3420	GCC	13.09	13.22	19.03	37.85	5.94
	3888	Eclipse	14.78	39.35	25.56	16.1	20.63

TABLE IV: Comparison of result using F-measure(%)

Severity	Our Approach (%)			Existing Approach[10] (%)		
	GCC	Eclipse	Mozilla	GCC	Eclipse	Mozilla
Blocker	25.45	15.29	54.73	13.96	27.41	55.02
Critical	77.95	24.52	47.46	80.25	29.19	49.9
Major	20.22	53.19	37.51	32.1	53.12	43.13
Minor	34.43	31.04	42.68	36.61	44.04	46.99
Trivial	13.56	28.63	34.89	20.12	26.08	32.96

we require very few features to discriminate the bug severity). These selected features are then used for classification. The results (F-measure) for different datasets using two classifiers are given in Fig.1. From Fig.1, it is observed that reasonable f-measure can be achieved using the selected features taken from only the description of the bug reports. However, in few cases specially for Mozilla, summary shows better performances compared to description. we believe combination of these two will surely improve the overall accuracies. We also use these results to answer **RQ 2** and **RQ 3**.

Answer to RQ 2: Description of bug reports performs better for each project than that of summary. It is quite expected because description contains more information than summary.

Answer to RQ 3: Except Eclipse, DT performs better for all projects using either summary or description.

We train the model with features computed from one project (e.g., Eclipse) and use it to classify bug reports of other projects (e.g., Mozilla). Table III represents these results using DT as the performance of DT is better than SVM showed in Fig.1 and is used to answer **RQ 4**. It is noteworthy to mention here that CF (in second column of table III) represents the number of common features between two projects. For example, the number of common terms between Eclipse and GCC are 4173.

Answer to RQ 4: The overall performance of cross-project is rather low which is expected. This is because even though they share large number of common terms, the terms themselves may not be significantly discriminatory for classification. This can be easily understandable if we compare the highest common terms between the top 100 frequent words of Mozilla and Eclipse which is only around 30. Among the results presented in table III, Mozilla and Eclipse share more common terms compared to GCC and thus, this pair produce comparatively better results, i.e., when we train with Eclipse and test with mozilla or vice versa. However, the results of cross project is still low to use for practical application. We believe that following BFSp, it might be possible to develop a generic bug severity model by combining all the project in hand to classify the future bug reports which we will address in future.

Finally, we report our best performing results in Table IV which illustrates a comparison between our method and the method described in [10]. Our overall F-measure is slightly low compared to this state-of-the-art method. This is because we only use ”term frequency of description” as a feature where the method in [10] uses both summary and description along with topic modeling based on term frequency and IDF.

IV. Conclusion

In this paper, we introduce a new technique comprising of basic NLP processing and a feature selection algorithm for the automation of bug severity prediction. The proposed feature selection method identifies the important terms based on the condition that they are significantly relevant (using MI and χ^2 statistic) and also, appear in the Pareto optimality set. This process selects fewer number of terms and requires less computation than the existing approaches. Even though the f-measure of BFSp is slightly inferior compared to the existing state-of-the-art method, we believe incorporation of more sources of information (e.g., both summary and description) and features (e.g., instead of only term frequency, use of both term frequency and IDF that most existing papers use) will surely improve the prediction result. Moreover, we study the performance of severity assessment in cross-project experiment using BFSp and thus, suggest to open a new avenue of research that is necessary for reducing the cost of software maintenance.This will also improve the overall accuracy which is specially important for safety critical software and thus, contribute to the humanitarian software technologies. Finally, the proposed BFSp can also be used as a general feature selection method for different area of data mining and pattern recognition problems. To verify this claim, we experimented with several different datasets from UCI Machine Learning Repository [26] and obtained reasonable accuracies with much lower number of selected features. For example, in case of Spambase dataset, we obtained 10% higher accuracy (0.83) with 35 selected features compared to the method described in [13] that selects 41 features and obtain 0.73 accuracy. However, this proposed BFSp might require further improvement to get satisfactory result which will be addressed in future.

References

[1] L. Erlikh, “Leveraging legacy system dollars for e-business,” *IT professional*, vol. 2, no. 3, pp. 17–23, 2000.

[2] N. K. Nagwani and S. Verma, “A comparative study of bug classification algorithms,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 24, no. 01, pp. 111–138, 2014.

[3] A. F. Otoom, D. Al-Shdaifat, M. Hammad, and E. E. Abdallah, “Severity prediction of software bugs,” in *Information and Communication Systems (ICICS), 2016 7th International Conference on*. IEEE, 2016, pp. 92–95.

[4] K. Chaturvedi and V. Singh, “Determining bug severity using machine learning techniques,” in *Software Engineering (CONSEG), 2012 CSI Sixth International Conference on*. IEEE, 2012, pp. 1–6.

[5] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, “Comparing mining algorithms for predicting the severity of a reported bug,” in *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*. IEEE, 2011, pp. 249–258.

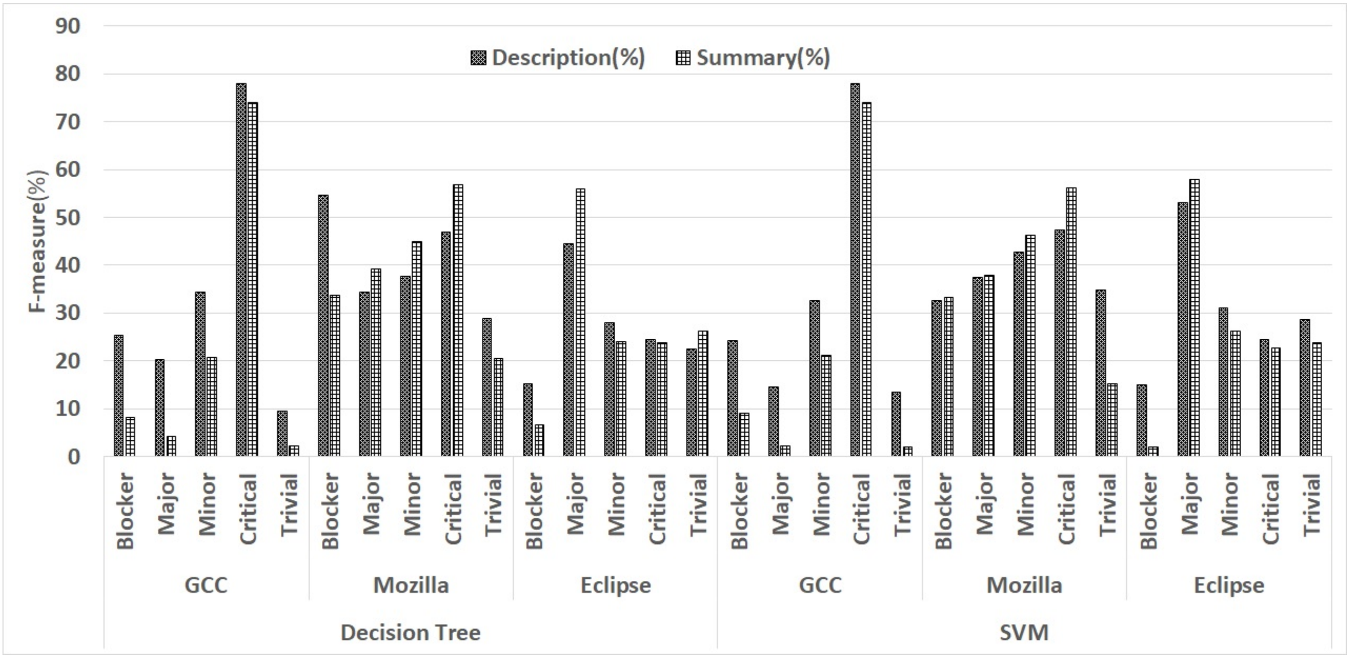

Fig. 1: Performance comparison of BFSp using Summary and Description of the bug reports

[6] S. Gujral, G. Sharma, S. Sharma *et al.*, "Classifying bug severity using dictionary based approach," in *Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), 2015 International Conference on*. IEEE, 2015, pp. 599–602.

[7] G. Sharma, S. Sharma, and S. Gujral, "A novel way of assessing software bug severity using dictionary of critical terms," *Procedia Computer Science*, vol. 70, pp. 632–639, 2015.

[8] N. K. S. Roy and B. Rossi, "Towards an improvement of bug severity classification," in *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. IEEE, 2014, pp. 269–276.

[9] G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *Computer software and applications conference (COMPSAC), 2014 IEEE 38th annual*. IEEE, 2014, pp. 97–106.

[10] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166–184, 2016.

[11] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *Reverse Engineering (WCRE), 2012 19th Working Conference on*. IEEE, 2012, pp. 215–224.

[12] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[13] T. Naghibi, S. Hoffmann, and B. Pfister, "A semidefinite programming based search strategy for feature selection with mutual information measure," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 8, pp. 1529–1541, 2015.

[14] M. Bennasar, Y. Hicks, and R. Setchi, "Feature selection using joint mutual information maximisation," *Expert Syst Appl*, vol. 42, no. 22, pp. 8520–8532, 2015.

[15] S. A. Fattah, C.-C. Lin, and S.-Y. Kung, "A mutual information based approach for evaluating the quality of clustering," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 2011, pp. 601–604.

[16] S. Sharmin, A. A. Ali, M. A. H. Khan, and M. Shoyaib, "Feature selection and discretization based on mutual information," in *Imaging, Vision & Pattern Recognition (icIVPR), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[17] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1226–1238, 2005.

[18] D. D. Lewis, "Feature selection and feature extraction for text categorization," in *Proceedings of the workshop on Speech and Natural Language*. Association for Computational Linguistics, 1992, pp. 212–217.

[19] R. Battiti, "Using mutual information for selecting features in supervised neural net learning," *IEEE Trans. Neural Netw*, vol. 5, no. 4, pp. 537–550, 1994.

[20] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján, "Conditional likelihood maximisation: a unifying framework for information theoretic feature selection," *Journal of Machine Learning Research*, vol. 13, no. Jan, pp. 27–66, 2012.

[21] S. Panzeri and A. Treves, "Analytical estimates of limited sampling biases in different information measures," *Network - Comp. Neural.*, vol. 7, pp. 87–107, 1995.

[22] R. Moddemeijer, "On estimation of entropy and mutual information of continuous distributions," *Signal processing*, vol. 16, no. 3, pp. 233–248, 1989.

[23] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 1–10.

[24] X. Xia, D. Lo, X. Wang, and B. Zhou, "Dual analysis for recommending developers to resolve bugs," *Journal of Software: Evolution and Process*, vol. 27, no. 3, pp. 195–220, 2015.

[25] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation." in *ECIR*, vol. 5. Springer, 2005, pp. 345–359.

[26] "Uci machine learning repository," http://archive.ics.uci.edu/ml/, (Accessed on 04/08/2017).