TRAM: An Approach for Assigning Bug Reports using their Metadata

Shadi Banitaan
Department of Mathematics, Computer Science and Software Engineering
University of Detroit Mercy
Detroit, MI 48221, USA
banitash@udmercy.edu

Mamdouh Alenezi
College of Computer & Information Sciences
Prince Sultan University
Riyadh 11586, Saudi Arabia
enezi.firstName@gmail.com

***Abstract*—Bug triage is an essential phase in the bug fixing process. The aim of bug triage is to assign an experienced developer to a new coming bug report. Existing bug triage approaches are mainly based on machine learning techniques. These approaches suffer from low prediction accuracy. In this paper, we propose TRAM (TRiaging Approach using bug reports Metadata). The goal is to improve the prediction accuracy of bug triage by utilizing the most discriminating terms of bug reports, the components in which the bugs belong to, and the reporter who filed the bug. We perform experimental evaluation on open-source projects namely Freedesktop, NetBeans, Eclipse, and Firefox. The results show that TRAM outperforms existing machine learning-based approaches in terms of classification accuracy. TRAM improves the F-score by approximately 34%, 40%, 20%, and 21% for Freedesktop, NetBeans, Eclipse, and Firfox respectively.**

***Keywords*—bug triage, term selection method, classification, mining bug repositories**

I. Introduction

Essential part of the software development process is bug reporting and fixing. In large open source software projects the bug tracking system is the central core for developers coordination and communication about a collection of bug reports and development issues. Bug tracking systems (BTS) enable many users to report their findings in a unified environment. These bug reports are utilized to guide software maintenance activities in order to produce more reliable software systems. Through the bug tracking systems, users are able to report new bugs, track changes of bug reports, and comment on existing bug reports.

Open source software systems receive abundant rate of bug reports daily. In an ideal world, each bug should be assigned to a developer who can fix it within a reasonable time. However, this is not usually the case. Many bug reports get assigned to inappropriate developers resulting in delaying the fixing time of these bugs. In addition, bug triaging, the process of assigning bugs to developers, is labor-intensive, time-consuming and fault-prone if done manually. Moreover, for open-source projects, keeping track of active developers and their expertise is very difficult. For instance, Anvik found that in the Eclipse project an average of 37 bugs per day are submitted to the BTS and 3 person-hours per day are required for the manual triage [1]. The empirical study by Jeong et al. [2] showed that 44% of bugs have been assigned to the wrong developer after the first assignment. In addition, the study reported that the first developer's assignment takes on average 40 and 180 days for Eclipse and Mozilla projects respectively. The second assignment takes on average 100 and 250 days for Eclipse and Mozilla projects respectively. These numbers indicate that manual triaging is an error-prone time-consuming process.

Previous work formulated the problem as a classification task where instances represent bug reports, features represent the textual description of bug reports, and developers who fixed those bugs as class labels. They built predictive models that can be used to predict a developer for a new coming bug report. Different classification techniques have been used to build classification models such as Naive Bayes and Support Vector Machines [3], [4]. All of these approaches suffer from low prediction accuracy. In this paper, the goal is to improve the prediction accuracy by using features other than the textual description of a bug such as the reporter who filed the bug.

To summarize, we make the following key contributions in this work:

- We reach much higher prediction accuracy compared to two other classification based bug triaging approaches by:
  - Using features of bug reports other than the textual data. Besides the bug description used in prior work, we incorporate more features namely the reporter who filed the bug and the component in which the bug belongs to.
  - Employing the most discriminating terms as a representation of the bug report instead of using all terms.
- We perform experimental evaluation using four bug reports datasets obtained from real projects. We use larger datasets compared to previous work. The experimental results show that building a classifier model using more features than the textual description of bugs can increase the prediction accuracy dramatically.

The rest of the paper is organized as follows: Section II discusses related work. Section III describes the proposed approach. The experimental evaluation and discussion are presented in Section IV. Section V discusses threats to validity and Section VI concludes the paper.

978-1-4673-5307-6/13/$31.00 ©2013 IEEE

II. Related Work

Many approaches adopted both machine learning and information retrieval techniques to improve the bug triaging process. Čubranić et al. [3] were the first to use a text classification approach to automatically assign bug reports to developers. Anvik et al. [5], [4] improved the approach proposed by Čubranić et al. by removing inactive developers (i.e, developers with a too low bug fixing frequency or developers whom not working on the project anymore). They employed SVM, Naive Bayes and Decision Trees classification techniques, and reported prediction accuracy of up to 64%. TRAM is different from their work in the following ways. We use the discriminating terms of bug reports as features instead of using all terms. In addition, we use other information about bug reports such as the reporter who filed the bug to obtain higher accuracy.

Several approaches were proposed in literature to enhance bug assignment accuracy. Park et al. [6] proposed a bug triaging approach. Their approach incorporated collaborative filtering recommender and topic modeling to improve bug triaging, reduce the sparseness of the training data, and enhance the quality of the triaging recommendation. Zou et al. [7] proposed the training set reduction with both feature selection and instance selection techniques for bug triage. They combine feature selection with instance selection to improve the accuracy of bug triage. They evaluated their approach on Eclipse where their approach removed 70% words and 50% bug reports. Alenezi et al. [8] employed five state-of-the-art term selection methods on the textual description of bug reports to produce discriminating terms. After that, they built a classification model on the discriminating terms using Naive Bayes classifier. Moreover, they re-balanced the load between developers. Their experimental results on four real datasets showed that by selecting a small number of discriminating terms, the classification accuracy can be significantly improved. In this work, we not only utilize a term selection method to select the most discriminating terms in bug reports but also we use other metadata about bug reports in order to boost the accuracy.

Other approaches have been proposed to automate bug assignment using different techniques other than classification. Matter et al. [9] represent a developer's expertise using the vocabulary found in the developer's source code. They recommend experienced developers by extracting information from new bug reports and looking it up in the vocabulary. Their approach does not require a history of bug reports and was tested on 130,769 Eclipse bug reports. They achieved 33.6% top-1 precision and 71.0% top-10 recall using eight years of Eclipse project. Tamrawi et al. [10] used fuzzy-sets to model bug-fixing expertise of developers based on the hypothesis that developers who recently fixed bugs are likely to fix them in the near future. They only considered recent reports to build the fuzzy-sets representing the membership of developers to technical terms in the reports. For new incoming reports, developers are recommended by comparing their membership to the terms included in the new report.

III. Approach

In this Section we present our proposed approach, TRAM. Figure 1 shows a high level description of TRAM. Adopted from previous research, we formulate bug assignment as a classification task where each instance represents a bug report, features represent several information about each bug, and a class label represents the developer who fixed the bug report. TRAM consists of the following main steps: 1) select the most discriminating terms from the textual contents of bug reports, 2) build a classification model using features from bug reports' metadata, 3) predict a developer with relevant experience to newly coming bug reports using the model built in the previous step. A detailed description of these steps is shown next.

A. Bug Representation

A collection of bug reports are represented as B = $\{b_1, \cdots, b_{|B|}\}$. Each bug report b_i has a collection of features, F = $\{f_1, \cdots, f_{|F|}\}$, and a class label (developer), $c \in C = \{c_1, \cdots, c_{|C|}\}$.

B. Feature Space

Feature extraction is an essential part of building an accurate classification model. We examine the impact of the textual representation of bug reports, and other information about bug reports namely the component and the reporter. The following is a set of features that we use to build the classification model:

- Discriminating terms: 1% of the most discriminating terms of the textual description of bug reports selected by X^2 method. Using discriminating terms instead of all terms as features improve the classification accuracy dramatically and reduce the dimensionality of data as shown in literature [7], [8].
- Component: It represents the component in which the bug belongs to. For example, in the Netbeans dataset, some of the components are Java, Compiler, UI, and Ant.
- Reporter: It represents the person who filed this bug report.

C. Data Pre-processing

Previous studies used both summary and description to represent the textual contents of bug reports. The description of bug reports contains too many terms that are unrelated to the actual functionality of bug reports which distract the classifier from detecting the actual developers who fixed the bugs (e.g., stack traces and steps to reproduce the bugs). Nevertheless, the summary of bug report contains terms that are related to the functionality of bug report and include less terms [11]. For these reasons, we only consider the title (summary) as the textual representation of a bug report.

Bug reports are unstructured data which contain irrelevant terms. Therefore, we apply the traditional text processing approach to transform the text data into a meaningful representation. The text processing includes white-spaces, punctuation, numbers, and stopwords removal and stemming (i.e., process of conflating the variant forms of a word into a common representation). After that, the approach constructs a bug-term matrix weighted by Term Frequency (TF) [12]. After that, a filtering is performed to refine the training set further to remove reports that are assigned to inactive developers (i.e., developers

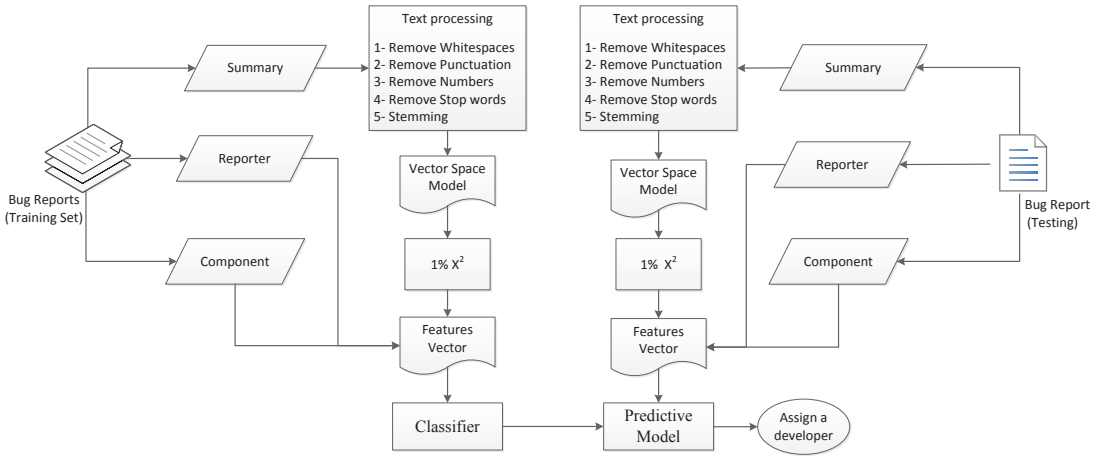

Fig. 1. The TRAM Approach.

who no longer work on the project or developers who have only fixed a small number of bugs) or reports that does not have sufficient words to describe a meaningful description. We eliminate bug reports that have less than 3 words since it is too short to hold relevant information. We consider developers who have fixed at least 20 bug reports in the dataset as active developers.

D. Naive Bayes and Chi-Square

Naive Bayes classifier is used to build the predictive model. Although known for its simplicity, the Naive Bayes algorithm have been found to perform astonishingly well in information retrieval. Naive Bayes is the best classifier compared to many common classifiers such as decision tree, neural network, and support vector machines in terms of accuracy and computational efficiency [13].

Feature or term selection methods include the removal of non-informative terms according to corpus statistics. For selecting the discriminating terms, we use the X^2 term selection method for two reasons. First, X^2 is verified as the best feature selection method for text classification [14]. Second, X^2 gave the best performance compared to other term selection methods in bug triaging [8]. In statistics, the X^2 test is used to examine independence of two events. The events, X and Y, are assumed to be independent if P(XY) = P(X)P(Y). In term selection, the two events are the occurrence of the term and the occurrence of the class. Terms are ranked with respect to the following equation [12]:

$$X^2(t,c) = \sum_{t\in\{0,1\}} \sum_{c} \frac{(N_{t,c} - E_{t,c})^2}{E_{t,c}}$$

where N is the observed frequency and E is the expected frequency for each state of term t and class c. X^2 is a measure of how much expected counts E and observed counts N deviate from each other.

IV. Experimental Evaluation

In this Section, we report the results of TRAM on real datasets. A description of the datasets that are used is shown in Section IV-A and the results are shown and discussed in Section IV-B. Section IV-C shows the most influential features for each project.

A. Datasets

We evaluate TRAM based on Bugzilla bug repositories of Freedesktop[1], NetBeans[2], Eclipse[3], and Firefox[4]. In our work, we collect the bug reports that have the status of [Closed, Verified, and Resolved] and the resolution of [Fixed]. Choosing these statuses means that most of the fields in these bug reports have been confirmed. For each bug report, we extract the bug ID, the summary, the reporter, the component, and the assignee. For Freedesktop, we choose (9881) bug reports from January 1^{st}, 2011 until December 31^{st}, 2012. For NetBeans, we choose (9450) bug reports from January 1^{st}, 2012 until December 31^{st}, 2012. For Eclipse, we choose (7898) bug reports from January 1^{st}, 2011 until December 31^{st}, 2012. For Firefox, we choose (5101) bug reports from January 1^{st}, 2011 until December 31^{st}, 2012. Table I shows a summary of the datasets and Table II shows a statistics about the refined datasets.

TABLE I. Summary of the Datasets

Project	# of Bugs	From	To
Freedesktop	9881	Jan, 01, 2011	Dec 31, 2012
NetBeans	9450	Jan, 01, 2012	Dec 31, 2012
Eclipse	7898	Jan, 01, 2011	Dec 31, 2012
Firefox	5101	Jan, 01, 2011	Dec 31, 2012

TABLE II. Statistics About the Datasets

Project	Bugs	Developers	Compon.	Reporters
Freedesktop	8583	73	237	3249
NetBeans	8947	55	256	1677
Eclipse	7339	61	29	1019
Firefox	3856	41	40	719

[1]https://bugs.freedesktop.org/
[2]http://netbeans.org/bugzilla/
[3]https://bugs.eclipse.org/bugs/
[4]https://bugzilla.mozilla.org/

B. Results

Each dataset is divided into training and testing sets. To obtain unbiased evaluation results, we perform a 5-fold cross-validation. The 5-fold cross-validation is used to avoid over-fitting problem by calculating the average of the results [15]. Three methods are compared in this work as follows:

- Baseline: The vector space model is constructed by considering all the terms (after processing) in the bug summary weighted by TF.
- X^2: The vector space model is constructed by selecting the 1% most discriminating terms chosen by X^2.
- TRAM: The vector space model is constructed by selecting metadata from bug reports namely the most discriminating terms chosen by X^2, Component, and Reporter.

The classification of bug reports is evaluated using Precision, Recall, and F-score as follows:

$$\text{Precision} = \frac{\text{Number of correct recommendations}}{\text{Number of recommendations made}}$$

$$\text{Recall} = \frac{\text{Number of correct recommendations}}{\text{Number of possible relevant developers}}$$

$$\text{F-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Table III shows the Precision and Recall of the three methods. The X^2 approach improves the Precision over the baseline approach whereas it diminishes the Recall. For instance, X^2 improves the Precision over the baseline approach by 22.4% while it diminishes the Recall over the baseline approach by 7.8% in NetBeans. On the other hand, TRAM improves both the Precision and Recall dramatically over both baseline and X^2 on all datasets. For example, TRAM improves Precision by 39.5% and 17.1% over baseline and X^2 respectively in NetBeans. TRAM also improves Recall by 40.6% and 46.4% over baseline and X^2 respectively in NetBeans.

TABLE III. CLASSIFICATION RESULTS

Project	Method	Precision	Recall
Freedesktop	Baseline	0.31	0.303
	X^2	0.539	0.251
	TRAM	0.663	0.623
NetBeans	Baseline	0.271	0.232
	X^2	0.495	0.154
	TRAM	0.666	0.638
Eclipse	Baseline	0.296	0.289
	X^2	0.408	0.188
	TRAM	0.493	0.482
Firefox	Baseline	0.326	0.318
	X^2	0.365	0.322
	TRAM	0.532	0.528

Figure 2 shows the F-score of the approaches investigated in this work. For Freedesktop, the F-score is 0.306, 0.343, and 0.642 for baseline, X^2, and TRAM respectively. TRAM improves the F-score by 33.6% and 30% over baseline and X^2 respectively. For NetBeans, the F-score is 0.250, 0.235, and 0.652 for baseline, X^2, and TRAM respectively. TRAM improves the F-score by 40.2% and 41.7% over baseline and X^2 respectively. For Eclipse, the F-score is 0.292, 0.257, and 0.487 for baseline, X^2, and TRAM respectively. TRAM improves the F-score by 19.5% and 23% over baseline and X^2 respectively. For Firefox, the F-score is 0.322, 0.342, and 0.530 for baseline, X^2, and TRAM respectively. TRAM improves the F-score by 20.8% and 18.8% over baseline and X^2 respectively.

In summation, the experimental results appear in Table III and Figure 2 clearly indicate that our hypothesis of adding other features to the textual representation is valid. In this work, we incorporate metadata of bug reports which include the most discriminating terms in the report, the component that the bug belongs to, and the reporter. Incorporating such additional information helped in distinguishing between developers which clearly shown by the classification accuracy.

C. Significant Features

We have shown that TRAM gives high classification accuracy for all projects. It is noteworthy knowing the most influential features that determine correct developers for bug reports. The most influential features can be computed using gain ratio [16]. Gain ratio provides a normalized measure of the contribution of each feature to classification. We report the highest five influential features for each project in Table IV. The higher the gain ratio, the more important the feature to identify the developer.

TABLE IV. TOP 5 INFLUENTIAL FEATURES USING THE GAIN RATIO MEASURE.

Project	Feature	Feature Set	Gain Ratio
Freedesktop	fprinted	Component	1
	pixman	Component	0.973
	src	Component	0.973
	Driver/Radeon	Component	0.968
	SyncEvolution	Component	0.933
NetBeans	Dashboard	Component	0.973
	GlassFishv3	Component	0.949
	ReportException	Component	0.899
	Persistence	Component	0.873
	Inspection	Component	0.865
Eclipse	Core	Component	0.886
	UserAssistance	Component	0.861
	SWT	Component	0.81
	Relenge	Component	0.798
	Build	Component	0.769
Firefox	Installer	Component	0.744
	netzen	Reporter	0.709
	mano	Reporter	0.691
	robert.bugzilla	Reporter	0.665
	mconley	Reporter	0.648

V. THREATS TO VALIDITY

We consider the developer in which the report assigned to as the expert developer to fix that bug. This assumption is not always valid. The bug fixing is usually a collaborative effort. Therefore, other factors to determine the experience developers

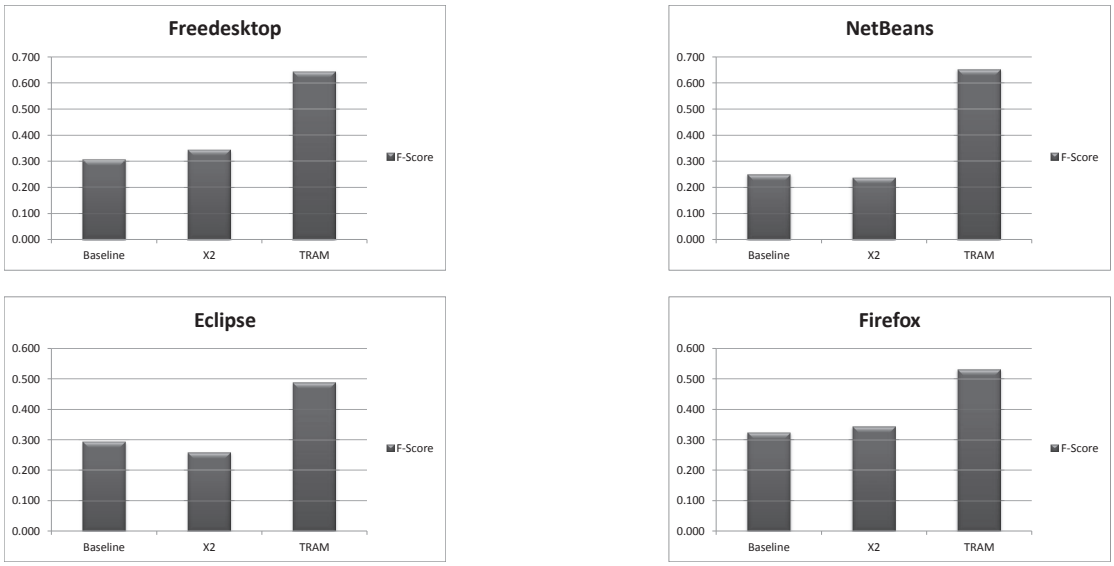

Fig. 2. F-Score of classification on four datasets.

should be considered. Moreover, we have validated TRAM on open source bug repositories; commercial projects may have different characteristics that may require some adaptation. Also, we only use projects that use Bugzilla as their bug tracking system. Other bug tracking systems are available such as Gnats and Trac that model bug reports in a different way. Therefore, TRAM should be applied to more open source and commercial projects in order to generalize the results.

VI. Conclusion

In this paper, we presented TRAM, a new approach to improve the prediction accuracy of the bug assignment task. We utilized three sources of information namely the most discriminating terms of bug reports, the component that the bug report belongs to, and the reporter who filed the bug. We built a classification model using the Naive Bayes classifier. We applied TRAM on Freedesktop, NetBeans, Eclipse, and Firefox. TRAM improves the F-score by approximately 34%, 40%, 20%, and 21% for Freedesktop, NetBeans, Eclipse, and Firefox respectively. As a result, using component and reporter information in addition to the discriminating terms of the textual description improved the accuracy of the classification significantly. Future directions include applying TRAM on other projects. In addition, we would like to investigate the effect of using other information such as the severity of bug reports on classification accuracy.

References

[1] J. Anvik, "Automating bug report assignment," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 937–940.

[2] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2009, pp. 111–120.

[3] D. Čubranić and G. C. Murphy, "Automatic bug triage using text categorization," in *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering*. Citeseer, 2004, pp. 92–97.

[4] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 3, p. 10, 2011.

[5] J. Anvik, L. Hiew, and G. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 361–370.

[6] J.-W. Park, M.-W. Lee, J. Kim, S. won Hwang, and S. Kim, "Costriage: A cost-aware triage algorithm for bug reporting systems." in *AAAI*, W. Burgard and D. Roth, Eds. AAAI Press, 2011.

[7] W. Zou, Y. Hu, J. Xuan, and H. Jiang, "Towards training set reduction for bug triage," in *Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference*, ser. COMPSAC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 576–581.

[8] M. Alenezi, S. Banitaan, and K. Magel, "Efficient bug triaging using text mining," *To appear in Journal of Software*, 2013.

[9] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 2009, pp. 131–140.

[10] A. Tamrawi, T. Nguyen, J. Al-Kofahi, and T. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 365–375.

[11] A. J. Ko, B. A. Myers, and D. H. Chau, "A linguistic analysis of how people describe software problems," in *Proceedings of the Visual Languages and Human-Centric Computing*, ser. VLHCC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 127–134.

[12] C. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 1.

[13] S. Ting, W. Ip, and A. H. Tsang, "Is naïve bayes a good classifier for document classification?" *International Journal of Software Engineering and Its Applications*, vol. 5, no. 3, 2011.

[14] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Machine Learning-International Workshop Then Conference-*. Morgan Kaufmann Publishers, Inc., 1997, pp. 412–420.

[15] M. Rogati and Y. Yang, "High-performing feature selection for text classification," in *Proceedings of the eleventh international conference on Information and knowledge management*. ACM, 2002, pp. 659–661.

[16] M. A. Hall and L. A. Smith, "Practical feature subset selection for machine learning," 1998.