Analysis of Bug Triage using Data Preprocessing (Reduction) Techniques

G. Parthasarathy
Research Scholar, Dept of CSE
Sathyabama University,
Chennai, India

D.C. Tomar
Professor, Dept of IT
Jerusalem College of Engineering,
Chennai, India

Blessy John
Student, Dept of CSE
Maamallan Institute of Technology
Sriperumpudur, India

ABSTRACT

In the bug triage we have an unavoidable step of fixing the bugs which helps in correctly assigning a developer to a new bug. Text classification and binary classification techniques are applied to decrease the time cost in manual work and to enhance the working of automatic bug triage. We address the problem of data reduction and hence we combine the instance selection and the feature selection algorithms to simultaneously reduce the data scale and enhance the accuracy of the bug reports in the bug triage. We determine a predictive model to perform the algorithms which adds on to prioritize the developer to a new bug by extracting attributes and the bug data set from the historical table. By leveraging data mining techniques, mining software repositories can uncover interesting information in software repositories and solve real-world software problem like Eclipse, Mozilla and GNOME.

Keywords

Bug Triage, Data Reduction in bug report, preprocessing the bug report, Fixing Bugs

1. INTRODUCTION

A time-consuming step of handling software bugs is bug triage, which aims to assign a correct developer to fix a new bug [1]. In traditional software development, new bugs are manually triaged by an expert developer, i.e., a human triager. Due to the large number of daily bugs and the lack of expertise of all the bugs, manual bug triage is expensive in time cost and low in accuracy. In manual bug triage in Eclipse, 44 percent of bugs are assigned by mistake while the time cost between opening one bug and its first triaging is 19.3 days on average. Taking an open source project Eclipse as an example an average of 30 new bugs are reported to bug repositories per day in 2007[7]. From 2001-2010 333,371 bugs have been reported to Eclipse by over 34917 developers and users [5]. To avoid the expensive cost of manual bug triage, existing work [1] has proposed an automatic bug triage approach, which applies text-classification techniques to predict developers for bug reports. In this approach, a bug report is mapped to a document and a related developer is mapped to the label of the document.

In this paper, we address the problem of data reduction for bug triage, i.e., how to reduce the bug data to save the labor cost of developers and improve the quality to facilitate the process of bug triage. Data reduction for bug triage aims to build a small-scale and high-quality set of bug data by removing bug reports and words, which are redundant or non-informative. In our work, we combine existing techniques of instance selection and feature selection to simultaneously reduce the bug dimension and the word dimension. The reduced bug data contain fewer bug reports and fewer words than the original bug data and provide similar information over the original bug data. We evaluate the reduced bug data according to two criteria: the scale of a data set and the accuracy of bug triage.

Given an instance selection algorithm and a feature selection algorithm, the order of applying these two algorithms may affect the results of bug triage. In this paper, we propose a predictive model to determine the order of applying instance selection and feature selection. We refer to such determination as prediction for reduction orders. Drawn on the experiences in software metrics, we extract the attributes from historical bug data sets. Then, we train a binary classifier on bug data sets with extracted attributes and predict the order of applying instance selection and feature selection for a new bug data set.

In the experiments, we evaluate the data reduction for bug triage on bug reports of two large open source projects, namely Eclipse and Mozilla. Experimental results show that applying the instance selection technique to the data set can reduce bug reports but the accuracy of bug triage may be decreased; applying the feature selection technique can reduce words in the bug data and the accuracy can be increased. Meanwhile, combining both techniques can increase the accuracy, as well as reduce bug reports and words. For example, when 50 percent of bugs and 70 percent of words are removed, the accuracy of Naive Bayes on Eclipse improves by 2 to 12 percent and the accuracy on Mozilla improves by 1 to 6 percent[7]. Based on the attributes from historical bug data sets, our predictive model can provide the accuracy of 71.8 percent for predicting the reduction order. Based on top node analysis of the attributes, results show that no individual attribute can determine the reduction order and each attribute is helpful to the prediction.

The primary contributions of this paper are as follows:

1) We present the problem of data reduction for bug triage. This problem aims to augment the data set of bug triage in two aspects, namely a) to simultaneously reduce the scales of the bug dimension and the word dimension and b) to improve the accuracy of bug triage.

2) We propose a combination approach to addressing the problem of data reduction. This can be viewed as an application of instance selection and feature selection in bug repositories.

3) We build a binary classifier to predict the order of applying instance selection and feature selection. To our knowledge, the order of applying instance selection and feature selection has not been investigated in related domains.

2. MOTIVATION FOR DATA REDUCTION

In the bug repositories, all the bug reports are filled by the developers in natural languages. The low quality bugs accumulate in bug repositories with the growth in scale.

Consider some examples:

We list the bug report of bug 205900 of Eclipse in Example 1 (the description in the bug report is partially omitted) to study the words of bug reports.

a) **Example 1** (Bug 205900). Current version in Eclipse Europa discovery repository broken.

   [Plug-ins] all installed correctly and do not show any errors in Plug-in configuration view. Whenever I try to add a [diagram name] diagram, the wizard cannot be started due to a missing [class name] class ...

   In this bug report, some words, e.g., installed, show, started, and missing, are commonly used for describing bugs. For text classification, such common words are not helpful for the quality of prediction. Hence, we tend to remove these words to reduce the computation for bug triage. However, for the text classification, the redundant words in bugs cannot be removed directly. Thus, we want to adapt a relevant technique for bug triage . To study the noisy bug report, we take the bug report of bug 201598 as Example 2 (Note that both the summary and the description are included).

b) **Example 2** (Bug 201598). 3.3.1 about says 3.3.0. Build id: M20070829-0800. 3.3.1 about says 3.3.0.)

   This bug report presents the error in the version dialog. But the details are not clear. Unless a developer is very familiar with the background of this bug, it is hard to find the details. According to the item history, this bug is fixed by the developer who has reported this bug. But the summary of this bug may make other developers confused.

   Moreover, from the perspective of data processing, especially automatic processing, the words in this bug may be removed since these words are not helpful to identify this bug. Thus, it is necessary to remove the noisy bug reports and words for bug triage. To study the redundancy between bug reports, we list two bug reports of bugs 200019 and 204653 in Example 3 (the items description are omitted).

c) **Example 3**. Bugs 200019 and 204653.
   (Bug 200019) Argument popup not highlighting the correct argument ...(Bug 204653) Argument highlighting incorrect ...

   In bug repositories, the bug report of bug 200019 is marked as a duplicate one of bug 204653 (a duplicate bug report, denotes that a bug report describes one software fault, which has the same root cause as an existing bug report [4]). The textual contents of these two bug reports are similar. Hence, one of these two bug reports may be chosen as the representative one. Thus, we want to use a certain technique to remove one of these bug reports. Thus, a technique to remove extra bug reports for bug triage is needed.

   Based on the above three examples, it is necessary to propose an approach to reducing the scale (e.g., large scale words in Example 1) and augmenting the quality of bug data (e.g., noisy bug reports in Example 2) and redundant bug reports in Example 3).

3. RELATED WORK

Usually when a fault or a new bug report is encountered we have to assign a developer to fix the bug. We use a classifier to know to which developer we need to assign the bug. This can be done by retrieving the information from the history table where we can extract the details like which developer has fixed the bug efficiently, how much time each developer has taken to fix each bug and how many fixers were needed to fix a particular bug etc. Now we reduce the data scale in the bug data sets by using instance selection and feature selection algorithms.

First we use the text classification technique to classify the words in the bug triage. Then we propose a predictive model to determine the reduction order. After which the bug data reduction process takes place (i.e) we use the instance selection and the feature selection algorithms. The Feature selection aims to obtain a subset of relevant features by removing the uninformative words in the bug reports and the instance selection is used to obtain a subset of relevant instances (bug reports in the bug data). Hence by combining both these algorithms we get a reduced bug data set and replace it with the original bug data. Now this information is sent to the classifier and when a new bug report is encountered the classifier correctly assigns a developer.

3.1 Related Questions

a) If 500 reports are encountered at a time , then which report has to be checked first?

The solution to this is we take the entropy of severity of the bug reports(i.e) predicting the severity of the bug reports[3]. Severity is nothing but the importance.

The severity depends on the following priorities:

- How fast you can fix the bug?
- Whether it can be postponed?
- It can never be fixed.

b) How do we assign the right developer to fix the bug?

We can model a system to rank the developers. We face the problems like ranking, evaluation time and tolerance of noisy comments. All these can be removed and the prioritizing of the developers is done using a Socio Network technique[5]. This is useful to improvise the

- Bug triage
- Severity of the bug reports
- Prediction of the reopened bugs.

c) How do we select the report?

Opinion analysis is mainly the process of identifying the polarity used in any comments or sentences. The main aim is to identify the polarity used in the context with respect to a particular citation. Similarly when a report is posted the weight of the words and its frequencies are calculated.

3.2 Techniques Used

3.2.1 Text Classification Technique

In this technique, [8]some of the main concepts are :

a) Tokenization

It is the process of replacing sensitive data with unique identification symbols that retain all the essential information about the data.

b) Stop words
Removing the non-informative words which includes articles, prepositions, conjunctions and certain high frequency words (verbs, adverbs and adjectives).

c) Stemming
Stemming is a technique to reduce words to their grammatical roots so that they can be represented with a only term.

Text data can be represented as strings, though simplified representations are used for effective processing. The most common representation for text is the vector-space representation [9].

The vector-space representation treats each document as an unordered "bag of words". While the vector-space representation is very efficient because of its simplicity, it loses information about the structural ordering of the words in the document, when used purely in the form of individual word representations.

One advantage of the vector-space representation is that its simplicity lends itself to straightforward processing. The efficiency of the vector-space representation has been a key reason that it has remained the technique of choice for a variety of text processing applications.

On the other hand, the vector-space representation is very lossy because it contains absolutely no information about the ordering of the words in the document. While the vector-space representation maintains no information about the ordering of the words, the string representation is at the other end of spectrum in maintaining all ordering information.

Distance graphs [9] are a natural intermediate representation which preserve a high level of information about the ordering and distance between the words in the document. At the same time, the structural representation of distance graphs makes it an effective representation for text processing

.Distance graphs can be defined to be of a variety of orders depending upon the level of distance information which is retained.

3.2.2 *Instance Selection*

Instance Selection also known as Bug dimension is the process of removing redundant instances. Supervised learning which provides previously known information is used to classify new instances. In common, several instances are stored in the training set but some of them are not useful for classifying therefore we ignore the non-useful cases.

Given a training set T, the goal of an instance selection method (Fig. 1) is to obtain a subset $S \subset T$ such that S does not contain superfluous instances and Acc(S) ~= Acc(T)where Acc(X)is the classification accuracy obtained using X as training set (henceforth, S is used to denote the selected subset). Instance selection methods can either start with S = Ø (incremental method) or S = T (decremental method). The difference is that the incremental methods include instances in S during the selection process and decremental methods remove instances from S along the selection [14].

We define the problem of instance selection as the need to extract the most useful set of instances from a database which we know (or suspect) contains instances which are superfluous or harmful. In the context of instance-based learning, we seek to discard the cases which are superfluous or harmful to the classification process.

We want to isolate the smallest set of instances which enable us to predict the class of a query instance with the same (or higher) accuracy than the original set The Nearest Neighbour Classifier is a simple supervised concept learning scheme which classifies unseen (i.e., unclassified) instances by finding the closest previously observed instance, taking note of its class, and predicting this class for the unseen instance .Learners that employ this classification scheme are also termed Instance-Based Learners, Lazy Learners, Memory-Based Learners, and Case-Based Learners [13].

There are situations in which unlabeled data is abundant but manually labeling is expensive. In such a scenario, learning algorithms can actively query the user/teacher for labels. This type of iterative semi supervised learning is called active learning[13]. The need for automated mining and discovering knowledge from large-scale data, referred to as Knowledge discovery and data mining (KDD), is widely recognized. Common approaches in KDD are to either:

a) generate patterns without supervision, such as clustering

b) use some previously labeled instances to assist the pattern discovery process, such as supervised learning

To select most critical instances from the unlabeled sample set for labeling such that a model trained on them can achieve the maximum prediction accuracy, compared to simple solutions such as randomly labeling the same number of instances:

1. **Utility metrics merely based on uncertainty of IID instances**:
   Methods in this category treat samples as independent and identically distributed (IID) instances, where the selection criteria only depend on the uncertainty values computed with respect to each individual instance's own information. Accordingly, one possible problem is that this type of approach may select similar instances in the candidate set, which results in redundancy in the candidate set
2. **Utility metrics further taking into account instance correlations**
   To take the sample redundancy into consideration, uncertainty metrics based on instance correlation utilizes some similarity measures to discriminate differences between instances. By uncovering inherent relationships between instances, the utility of the instances calculated by this scheme integrates sample correlations, through which a selected candidate set may not always contain the "most uncertain" instances. Whereas, together, the selected instances form an optimal candidate set by balancing the uncertainty and diversity.

We can divide the instance selection methods in two groups[14]:

a) **Wrapper:** The selection criterion is based on the accuracy obtained by a classifier.

b) **Filter**: The selection criterion uses a selection function which is not based on a classifier.

Advantages

a) Accuracy is achieved at a higher level since the redundant and the superfluous instances are removed.

3.2.3 Feature Selection

The feature selection is the process of detecting the relevant features and discarding the irrelevant ones. This focuses mainly on constructing and selecting subsets of features that are useful to build a good predictor. This contrasts with the problem of finding or ranking all potentially relevant variables. Selecting the most relevant variables is usually suboptimal for building a predictor, particularly if the variables are redundant [11] . This Feature selection process [10] depends on two concepts:

a) Individual Evaluation: Feature ranking is based on the degree of relevance.

b) Subset Evaluation : Produces candidate features based on certain search strategy.

The feature selection process takes place in 3 approaches [10]:
a) **Filters:** relies on the general characteristic of training data and carries out the feature selection process as a preprocessing step with the independence of the induction algorithm (extracting some formal rules from a set of observations).

Information Gain is one of the most common attribute evaluation methods. This univariate filter provides an ordered ranking of all the features, and then a threshold is required. In this work the threshold will be set up selecting the features which obtain a positive information gain value.

The Interact algorithm is a subset filter based on symmetrical uncertainty (SU) and the consistency contribution, which is an indicator about how significantly the elimination of a feature will affect consistency. The algorithm consists of two major parts. In the first part, the features are ranked in descending order based on their SU values. In the second part, features are evaluated one by one starting from the end of the ranked feature list. If the consistency contribution of a feature is less than an established threshold, the feature is removed, otherwise it is selected. The authors stated that this method can handle feature interaction, and efficiently selects relevant features.

b) **Embedded Methods** Performs feature selection in the process of training and are usually specific to given learning machines.

SVM-RFE (Recursive Feature Elimination for Support Vector Machines)was introduced by Guyon. This embedded method performs feature selection by iteratively training a SVM classifier with the current set of features and removing the least important feature indicated by the SVM. Two versions of this methods will be tested: the original one, using a linear kernel and an extension using a nonlinear kernel in order to solve more complex problems.

FS-P (Feature Selection—Perceptron) is an embedded method based on a perceptron. A perceptron is a type of artificial neural network that can be seen as the simplest kind of feedforward neural network: a linear classifier. The basic idea of this method consists on training a perceptron in the context of supervised learning. The interconnection weights are used as indicators of which features could be the most relevant and provide a ranking.

c) **Wrappers** This involves in optimizing a predictor as a part of the selection process. This evaluates attribute sets by using a learning scheme. Cross-validation is used to estimate the accuracy of the learning scheme for a set of attributes. The algorithm starts with the empty set of attributes and searches forward, adding attributes until performance does not improve further. In this work, two well-known learning schemes will be used: SVM and C4.5.

Advantages

There are many potential benefits of variable and feature selection: facilitating data visualization and data understanding, reducing the measurement and storage requirements, reducing training and utilization times, defying the curse of dimensionality to improve prediction performance [11].

a) Reducing the data scale : In word dimension we use feature selection to remove noisy or duplicate words in a data set. Based on feature selection, the reduced data set can be handled more easily by automatic techniques (e.g., bug triage approaches) than the original data set. Besides bug triage, the reduced data set can be further used for other software tasks after bug triage (e.g., severity identification, time prediction, and reopened bug analysis)

b) Improving Accuracy: In Word dimension by removing uninformative words, feature selection improves the accuracy of bug triage .This can recover the accuracy loss by instance selection.

3.3. THE OVERALL ARCHITECTURE

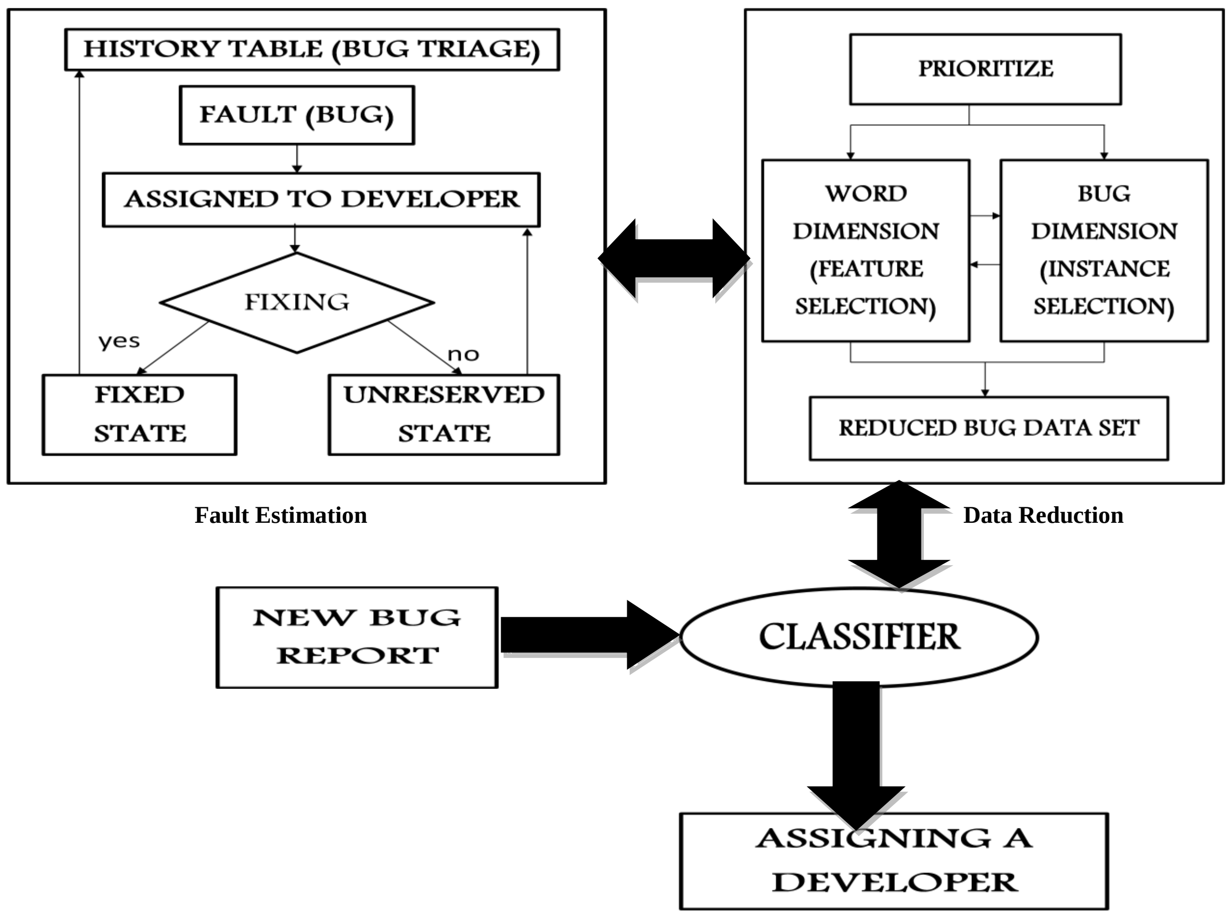


Fig 1. Architecture of Bug Triage

In this paper we propose a predictive model to determine the order of reduction (i.e) which of the two algorithms have to be performed first.

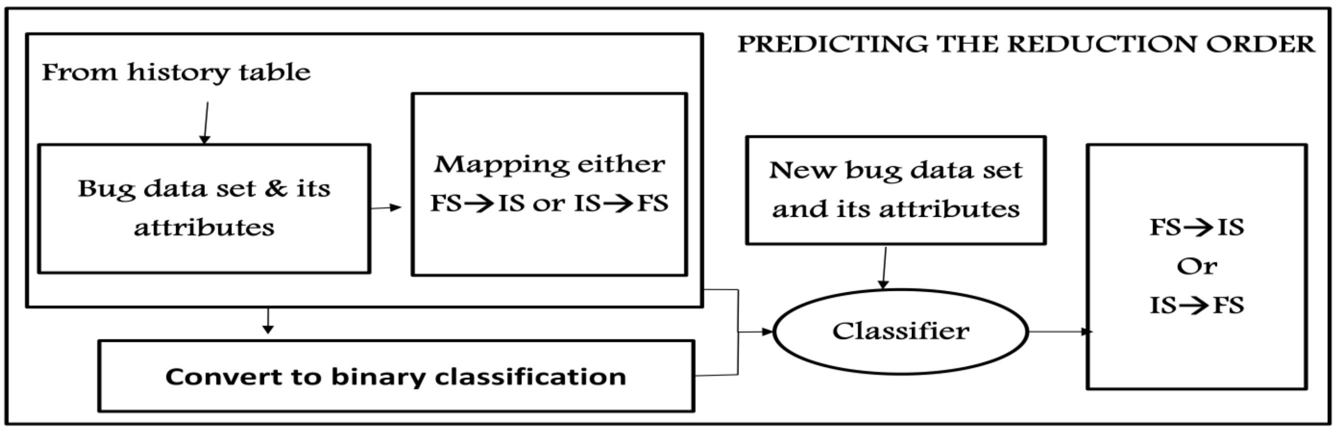

PREDICTING THE REDUCTION ORDER
From history table
Bug data set & its attributes
Mapping either FS→IS or IS→FS
Convert to binary classification
New bug data set and its attributes
Classifier
FS→IS Or IS→FS

Fig 2. Predicting the reduction order

a) First we retrieve the information from the history table (i.e) the bug data set.
b) We derive its attributes from the bug data set. With just one attribute we cannot predict the reduction order. Hence we use a list of attributes since each attribute contributes to the prediction.
c) Some of the attributes are :
    - Length of the bug report
    - Words per fixer
    - Entropy of severity

- Fixers
- Unique word
- Entropy of priority
- Entropy of components
- Ratio of sparseness
- Entropy of product
- Bug reports per reporter
- Similarities between fixers.

d) We are using a top node analysis for predicting the reduction order. This is done using the decision tree algorithm. Only nodes in Level 0 to level 2 of decision tree are presented and in each level we omit an attribute if its frequency is equal to 1.

e) Now in all the 3 levels we check for the attribute which has occurred in each level and consider it as the representative attribute.

Table 1. Top Node Analysis of Predicting Reduction Order

Level	Frequency	Index	Attribute Name
0	2	B3	Length of bug reports
	2	D3	#Words per fixer
1	3	B6	Entropy of severity
	3	D1	#Fixers
	2	B3	Length of bug reports
	2	B4	#Unique words
2	4	B6	Entropy of severity
	3	B7	Entropy of priority
	3	B9	Entropy of component
	2	B3	Length of bug reports
	2	B4	#Unique words
	2	B5	Ratio of sparseness
	2	B8	Entropy of product
	2	D5	#Bug reports per reporter
	2	D8	Similarity between fixers and reporters

f) Using this representative attribute, we can predict the reduction order i.e when a new bug report is encountered this representative attribute is used to check the reports in the history table for the reports which have their attributes similar to it.

g) By doing so we can obtain accuracy and at the same time enhance the bug triage

4. EXPERIMENTAL RESULTS

4.1 Data Preparation

In this part, we present the data preparation for applying the bug data reduction. We evaluate the bug data reduction on bug repositories of two large open source projects, namely Eclipse and Mozilla. Eclipse [16] is a multi-language software development environment, including an Integrated Development Environment (IDE) and an extensible plug-in system; Mozilla [15] is an Internet application suite, including some classic products, such as the Firefox browser and the Thunderbird email client. Up to December 31, 2011, 366,443 bug reports over 10 years have been recorded to Eclipse while 643,615 bug reports over 12 years have been recorded to Mozilla. In our work, we collect continuous 300,000 bug reports for each project of Eclipse and Mozilla, i.e., bugs 1-300000 in Eclipse and bugs 300001- 600000 in Mozilla. Actually, 298,785 bug reports in Eclipse and 281,180 bug reports in Mozilla are collected since some of bug reports are removed from bug repositories (e.g., bug 5315 in Eclipse) or not allowed anonymous access (e.g., bug 40020 in Mozilla). For each bug report, we download web- pages from bug repositories and extract the details of bug reports for experiments. Since bug triage aims to predict the developers who can fix the bugs, we follow the existing work [1], to remove unfixed bug reports, e.g., the new bug reports or will not fix bug reports. Thus, we only choose bug reports, which are fixed and duplicate (based on the items status of bug reports). Moreover, in bug repositories, several developers have only fixed very few bugs. Such inactive developers are removed.

In our work, we remove the developers, who have fixed less than 10 bugs. To conduct text classification, we extract the summary and the description of each bug report to denote the content of the bug. For a newly reported bug, the summary and the description are the most representative items, which are also used in manual bug triage [1]. As the input of classifiers, the summary and the description are converted into the vector space model [9]. We employ two steps to form the word vector space, namely tokenization and stop word removal. First, we tokenize the summary and the description of bug reports into word vectors. Each word in a bug report is associated with its word frequency, i.e., the times that this word appears in the bug. Non-alphabetic words are removed to avoid the noisy words, e.g., memory address like 0x0902f00 in bug 200220 of Eclipse. Second, we remove the stop words, which are in high frequency and provide no helpful information for bug triage, e.g., the word “the” or “about”. We do not use the stemming technique in our work since existing work [1], has examined that the stemming technique is not helpful to bug triage. Hence, the bug reports are converted into vector space model for further experiments.

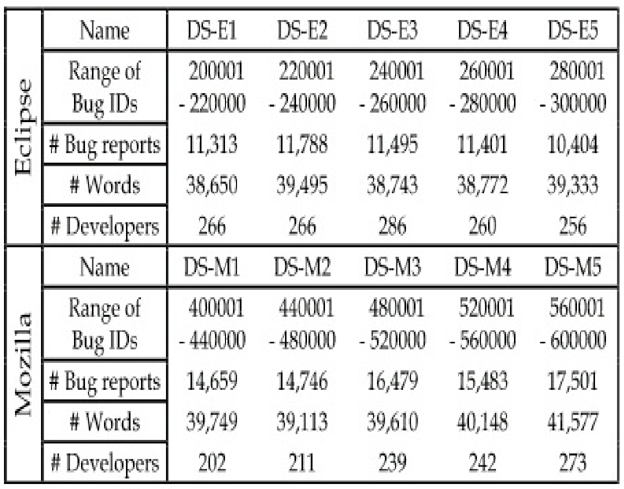
4.2 Experiments on Bug Data Reduction

4.2.1 Data Sets and Evaluation

The results of data reduction for bug triage can be measured in two aspects, namely the scales of data sets and the quality of bug triage. Based on Algorithm 1, the scales of data sets (including the number of bug reports and the number of words) are configured as input parameters. The quality of bug triage can be measured with the accuracy of bug triage, which is defined as

$$\text{Accuracy}_k = \frac{\text{\# correctly assigned bug reports in k candidates}}{\text{\# all bug reports in the test set}}$$

Table 2. Ten Data Sets in Eclipse and Mozilla



	Name	DS-E1	DS-E2	DS-E3	DS-E4	DS-E5
Eclipse	Range of Bug IDs	200001 - 220000	220001 - 240000	240001 - 260000	260001 - 280000	280001 - 300000
	# Bug reports	11,313	11,788	11,495	11,401	10,404
	# Words	38,650	39,495	38,743	38,772	39,333
	# Developers	266	266	286	260	256
	Name	DS-M1	DS-M2	DS-M3	DS-M4	DS-M5
Mozilla	Range of Bug IDs	400001 - 440000	440001 - 480000	480001 - 520000	520001 - 560000	560001 - 600000
	# Bug reports	14,659	14,746	16,479	15,483	17,501
	# Words	39,749	39,113	39,610	40,148	41,577
	# Developers	202	211	239	242	273

For each data set in Table 2, the first 80 percent of bug reports are used as a training set and the left 20 percent of bug reports are as a test set. In the following of this paper, data reduction on a data set is used to denote the data reduction on the training set of this data set since we cannot change the test set.

4.2.2 Results of Rates of Selected Bug Reports and Words

For either instance selection or feature selection algorithm, the number of instances or features should be determined to obtain the final scales of data sets. We investigate the changes of accuracy of bug triage by varying the rate of selected bug reports in instance selection and the rate of selected words in feature selection. We evaluate the results of data reduction for bug triage on data sets in Table 2. First, we individually examine each instance selection algorithm and each feature selection algorithm based on one bug triage algorithm, Naive Bayes. Second, we combine the best instance selection (ICF) algorithm and the best feature selection(CH) algorithm to examine the data reduction text based bug triage algorithms.

4.2.3 Results on Predicting the Reduction order

The results in Tables 3 and 4 show that the order of applying instance selection and feature selection can impact the final accuracy of bug triage.

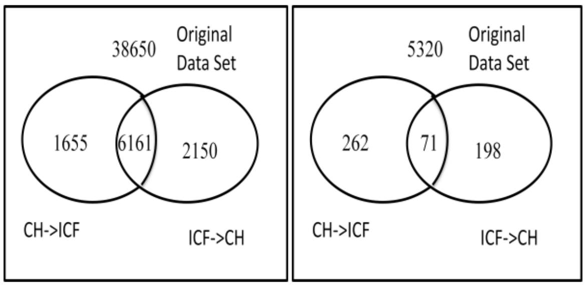
Table 3. Accuracy (percent) of Data Reduction on DS- E1

Naïve Bayes		
Origin	**CH-> ICF**	**ICF-> CH**
25.85	25.42	**27.24**
35.71	39.00	**39.56**
41.88	46.88	**47.58**
45.84	51.77	**52.45**
48.95	55.55	**55.89**

Table 4. Accuracy (percent) of Data Reduction on DS-M1

Naïve Bayes		
Origin	**CH->ICF**	**ICF->CH**
10.86	17.07	**19.45**
27.29	31.77	**32.11**
37.99	**41.67**	40.28
44.74	**48.43**	46.47
49.11	**53.38**	51.40

First we measure the differences of reduced data set by CH->ICF and ICF->CH. By referring fig.3 we observe that by using CH->ICF more no of words (2150) are removed and thus reduces the data set when compared with ICF->CH(1655) and it keeps the vive versa.



a)Words in data sets **b)Bug reports in data sets**

Fig 3.Bug reports and words in DS- E1(i.e bugs 200001-220000 in Eclipse by applying CH->ICF and ICF->CH

Second we check the duplicate the bug reports in the data set by CH->ICF and ICF->CH. By referring fig.3 we observe that by using ICF->CH(262) more no of redundant bug reports are removed and thus reduces the data set when compared with CH->ICF(198).

Thirdly, we check the blank bug reports during the data reduction. Here a blank bug report refers to a zero word bug report, whose words are removed by feature selection. Such blank reports are removed. The removed bug reports and words can be viewed as a kind of noisy data. ICF->CH removes comparatively a higher no of noisy blank bug reports. Thus we find out that the order of applying instance and feature selection impacts the ability of removing the noisy and redundant data.

5. CONCLUSION

Bug triage is an expensive step of software maintenance in both labor cost and time cost. In this paper, we combine feature selection with instance selection to reduce the scale of bug data sets as well as improve the data quality. To determine the order of applying instance selection and feature selection for a new bug data set, we extract attributes of each bug data set and train a predictive model based on historical data sets. We empirically investigate the data reduction for bug triage in bug repositories of two large open source projects, namely Eclipse and Mozilla. Our work provides an approach to leveraging techniques on data processing to form reduced and high-quality bug data in software development and maintenance. In future work, we plan on improving the results of data reduction in bug triage to explore how to prepare a high quality bug data set and tackle a domain-specific software task. For predicting reduction orders, we plan to pay efforts to find out the potential relationship between the attributes of bug data sets and the reduction orders.

6. REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in Proc. 28th Int. Conf. Softw. Eng., May 2006, pp. 361–370.

[2] S. Artzi, A. Kie_ zun, J. Dolby, F. Tip, D. Dig, A. Paradkar,

[3] and M. D.Ernst, "Finding bugs in web applications using dynamic test generation and explicit-state model checking," IEEE Softw., vol. 36,no. 4, pp. 474–494, Jul./Aug.2010.

[4] A.Lamkanfi, S. Demeyer, E. Giger, and B. Goethals,

[5] "Predicting the severity of a reported bug," in Proc. 7th IEEE Working Conf. Mining Softw. Repositories, May 2010, pp. 1–10.

[6] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in Proc. 30th Int. Conf. Softw. Eng., May 2008,pp. 461–470.

[7] J. Xuan, H. Jiang, Z. Ren, andW. Zou, "Developer prioritization in bug repositories," in Proc. 34th Int. Conf. Softw. Eng., 2012, pp. 25–35.

[8] Jifeng Xuan, He Jiang, "Towards effective bug triage with software dropping techniques", IEEE Transactions on Knowledge and Data Engineering, Vol 27, No.1 Jan 2015.

[9] J.R. Mendez, E.L Iglesias, F.Fdez Riverola, F.Diaz, "Tokenizing, Stemming and Stop word removal on Anti spam filtering Domain", CAEPIA 2005, LNAI4177,pp.449- 458,2006.

[10] C. C. Aggarwal and P. Zhao, "Towards graphical models for text processing," Knowl. Inform. Syst., vol. 36, no. 1, pp. 1– 21, 2013.

[11] V. Bolon-Canedo, N. Sanchez-Maro no, and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data," Knowl. Inform. Syst., vol. 34, no. 3, pp.483–519,2013.

A. Guyon and A. Elisseeff, "An introduction to variable and feature selection," J. Mach. Learn. Res., vol. 3, pp. 1157–1182, 2003.

[12] Y. Fu, X. Zhu, and B. Li, "A survey on instance selection for active learning," Knowl. Inform. Syst., vol. 35, no. 2, pp. 249–283, 2013.

[13] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," Data Mining Knowl. Discovery, vol. 6, no. 2, pp. 153–172, Apr. 2002.

[14] J. A. Olvera-Lopez, J. A.Carrasco-Ochoa, J. F. Martınez-Trinidad, and J. Kittler, "A review of instance selection methods," Artif. Intell. Rev., vol. 34, no. 2, pp. 133–143, 2010.

[15] Mozilla. (2014). [Online]. Available: http://mozilla.org/

[16] Eclipse. (2014). [Online]. Available: http://eclipse.org/.

[17] G.Parthasarathy and D.C Tomar, " Sentiment analysis of Journal Citations from citatation databases, IEEE, 2014.