

## Bug Severity Assessment in Cross Project Context and Identifying Training Candidates

V. B. Singh

*Delhi College of Arts & Commerce  
University of Delhi, Delhi, India  
vbsingh@dcac.du.ac.in*

Sanjay Misra

*Covenant University, Ota, Nigeria  
Department of Computer Engineering  
Atılım University, Ankara, Turkey  
ssopam@gmail.com*

Meera Sharma

*Department of Computer Science  
University of Delhi, Delhi, India  
meerakaushik@gmail.com*

Published 24 February 2017

**Abstract.** The automatic bug severity prediction will be useful in prioritising the development efforts, allocating resources and bug fixer. It needs historical data on which classifiers can be trained. In the absence of such historical data cross project prediction provides a good solution. In this paper, our objective is to automate the bug severity prediction by using a bug metric summary and to identify best training candidates in cross project context. The text mining technique has been used to extract the summary terms and trained the classifiers using these terms. About 63 training candidates have been designed by combining seven datasets of Eclipse projects to develop the severity prediction models. To deal with the imbalance bug data problem, we employed two approaches of ensemble by using two operators available in RapidMiner: Vote and Bagging. Results show that k-Nearest Neighbour (k-NN) performance is better than the Support Vector Machine (SVM) performance. Naive Bayes f-measure performance is poor, i.e. below 34.25%. In case of k-NN, developing training candidates by combining more than one training datasets helps in improving the performances (f-measure and accuracy). The two ensemble approaches have improved the f-measure performance up to 5% and 10% respectively for the severity levels having less number of bug reports in comparison of major severity level. We have further motivated the paper with a cross project bug severity prediction between Eclipse and Mozilla products. Results show that Mozilla products can be used to build reliable prediction models for Eclipse products and vice versa in case of SVM and k-NN classifiers.

**Keywords:** Bug severity; cross project prediction; text mining; ensemble approach.

### 1. Introduction

Bug reporting and fixing is an iterative and continuous activity throughout the software development life cycle, especially, in open source projects. Singh and Chaturvedi (2011)

have shown that a large number of bug tracking systems, namely Bugzilla, Promise, Mantis, Trac and Fossil have been developed which help in recording and fixing the bugs faced and reported by the users. The bug severity is the impact of the bug on the functionality of the software. The automatic bug severity prediction will be useful in allocating the resources and bug fixer. It is very important that the critical bugs need to be fixed on priority basis. Bug severity assessment can help in deciding the priority of a bug. In this paper, we consider the bug severity from the business point of view, not from the developer point of view (the impact of the bug on the functionality of the software). It is measured according to different levels from 1 to 7 (blocker, critical, major, normal, minor, trivial and enhancement). These levels are defined in repositories as 1 for high and 7 for lowest. The bug reports related to "Enhancement" have been removed from the dataset as they may not play any role in the severity identification because they specify the inclusion of a new feature or improvement in the existing feature not as a bug. The reports having the default severity, i.e. normal have also been removed from the dataset. The "Normal" severity level may confuse the classifier because it is the default level and assigned by the users when they are not sure about the severity level of such bugs. The summary attribute of a bug report contains a brief description about the bug.

In this paper, an attempt has been made to predict the severity level of bug reports opened in future releases/development cycles using already submitted bug reports with the predefined severity levels. We have used the bug summary attribute as it is more important than other bug attributes like component, product and version of the software for bug severity prediction. Using text mining, the extracted terms from bug summary have been used to train the classifiers. The datasets of six projects other than the testing project dataset have been combined to create training candidates. Training candidates consisting of no more than three datasets of other projects have been used to train the prediction models. Finally, we have identified the best training candidate for each testing dataset in cross project context. In order to address the inherent instability of results when applying models to imbalance datasets, we employed two approaches of ensemble by using two operators available in RapidMiner: Vote and Bagging. Vote operator has been used to make a combined prediction of the three constituent classifiers namely k-NN, NB and SVM. Vote operator uses the vote of each classifier for classification of bug severity level and the severity level with maximum votes is assigned to the newly reported bugs. Example, for a given bug report k-NN and SVM predict the severity level as 3 and NB predicts the severity level as 1 then the resultant combined prediction of the bug severity level will be 3 with majority votes. We used Bagging operator to make a combined prediction of the same type of model (using the same classifier) for different learning data where each learning data is repeated sub-sample (with replacement) of the original imbalance dataset. Bagging repeatedly creates sub-sample (with replacement) from the dataset, and make different classification models for each sub-sample using the same constituent classifier. The final classification is the one most often predicted by the different classification models.

The rest of the paper is organised as follows: related work has been discussed in Sec. 2. Section 3 describes the data and methods required to perform the analysis. Section 4 deals with the models building and evaluation. Section 5 describes experiments. Results have been discussed in Sec. 6. Section 7 presents threats to validity. Finally, the paper is concluded in Sec. 8 with the future research directions.

## 2. Related Work

This section represents a brief review of studies related to our paper. We highlight review of bug severity prediction in Sec. 2.1. In Sec. 2.2, studies related to Cross Project Defect Prediction (CPDP) have been reviewed. Section 2.3 presents other related works.

### 2.1. Bug severity prediction

In text-based bug severity assessment, [Menzies and Marcus \(2008\)](#) attempted to predict the severity of the reported bug using summary attribute. [Lamkanfi et al. \(2010\)](#) tried to predict the severity of a reported bug by analysing its textual description using text mining algorithms. Further, [Lamkanfi et al. \(2011\)](#) studied the severity prediction using Naïve Bayes (NB) classifier and categorise the bugs in severe and non-severe. The study was extended to make a comparison with other data mining techniques like Support Vector Machine (SVM), Multinomial Naïve Bayes (MNB) and k-Nearest Neighbour (k-NN) for open source projects. An attempt has made by [Chaturvedi and Singh \(2012\)](#) to determine the severity of reported bugs for closed source bug repositories of NASA datasets using NB, MNB, SVM, k-NN, J48, and RIPPER techniques. It is observed that different machine learning techniques are applicable in determining the bug severity levels with reasonable accuracy and f-measure. [Tian et al. \(2012\)](#) predicted fine grained severity labels by using extended BM25 and nearest neighbour classification. Further, an attempt has made by [Chaturvedi and Singh \(2013\)](#) to predict the severity of close source and open source projects using NB, MNB, SVM, k-NN, J48, and RIPPER techniques and found that SVM works with significant accuracy level for open source and NB works with significant accuracy level for closed source projects. In a study, [Yang et al. \(2014\)](#) proposed a method for the bug triage and bug severity prediction. The authors used historical bug reports in the bug repository for extracting topic(s) and then map the bug reports related to each topic. The authors identified corresponding reports having similar multi-feature (e.g. component, product, priority and severity) with the new bug report and then recommend the most appropriate developer to fix each bug and predict its severity. [Zhang et al. \(2015\)](#) suggested a concept profile-based severity prediction technique which first analysed historical bug reports in the bug repositories and then build the concept profiles from them. The authors evaluated the performance of their model by bug reports from the bug repositories of popular open-source projects — Eclipse and Mozilla Firefox and found that their method works effectively for prediction of the severity of a given bug. Recently,

Tian *et al.* (2015) proposed a new approach to better assess the accuracy of automated classifiers of severity labels given the unreliable nature of the data. Their approach shows that current automated approaches perform well 77–86% agreement with human-assigned severity levels.

## 2.2. Cross project defect prediction

Zimmermann *et al.* (2009) used 12 real world applications to run 622 cross-project defect predictions and found that only 21 predictions worked successfully with all precision, recall, and accuracy greater than 75%. Results show that cross-project defect prediction can fail if training data is not selected carefully and cross-project defect prediction is not symmetric. For example, data of Firefox can predict Internet Explorer defects well with precision 76.47% and recall 81.25%, but do not work in the opposite direction. The authors concluded that simply using historical data of projects in the same domain does not lead to good prediction results. A similar problem of cross-company defect prediction using data from other companies to predict defects for local projects has been done by Turhan *et al.* (2009b) using 10 projects (seven NASA projects from seven different companies and three projects from a Turkish software company SOFTLAB). It was concluded that cross-company data increases the probability of defect detection at the cost of increasing false positive rate. It was also concluded that defect predictors can be learned from small amount of local data. Turhan *et al.* (2009a) analysed a large-scale industrial system (25 projects) at the module level by using a state-of-the-art cross-company defect predictor. To predict defect proneness of modules, authors trained models on publicly available Nasa MDP data by using static Call Graph-Based Ranking (CGBR) and nearest neighbour sampling. Ma *et al.* (2011) attempted to transfer learning for cross-company software defect prediction and tried to exploit the transfer learning method to build fast and highly effective prediction model. Zhimin *et al.* (2012) made an attempt for cross-project defect prediction on three large-scale experiments on 34 datasets from 10 open source projects and found that training from data of other projects can provide better prediction results than the training data from the same project. Peters *et al.* (2013) introduced the Peters filter which is based on the conjecture that when local data is scarce, more information exists in other projects. Accordingly, this filter selects training data via the structure of other projects. A novel multi-objective approach has been proposed by Canfora *et al.* (2013) for cross-project defect prediction. It was based on a multi-objective logistic regression model and genetic algorithm. This approach allows software engineers to choose predictors for achieving a compromise between a number of likely defect-prone artefacts (effectiveness) and LOC to be analysed (which can be considered as a proxy of the cost of code inspection). Nam *et al.* (2013) applied a state-of-the-art transfer learning approach. Transfer Component Analysis (TCA) is used to make feature distributions in source and target projects similar. They extended the approach in TCA+ and showed that TCA+ significantly improves cross-project prediction performance.

Recently, Ryu *et al.* (2016) proposed a method called the Value-Cognitive Boosting with Support Vector Machine (VCB-SVM) to investigate the class imbalance learning under CPDP setting.

### 2.3. Other related works

In the other related works, Sharma *et al.* (2012) used cross project data in predicting the bug priority and found that the priority prediction in cross project context is satisfactory with a more than 70% accuracy and f-measure. Further, Sharma *et al.* (2014) made an attempt to predict the priority of a reported bug using different machine learning techniques and investigated the priority predictions in the cross project context by considering the combination of datasets for training candidates. Tian *et al.* (2014) presented an automated prediction of priority using multi-factor analysis. The multiple factors such as temporal, textual, author, related-report, severity, and product were considered which potentially affect the priority level of a bug report. Machine learning is used to recommend a priority level based on information available in bug reports. Bhatnagar *et al.* (2010) have made an attempt to handle the imbalance dataset problem using the ensemble method.

Assessing the severity class of a new bug report in cross project context is an untouched area till now. Our research has its roots in cross project bug severity prediction.

From the available literature it can be easily observed that the work of assessing the severity of a bug report in cross project context is untouched till now. This becomes the motivation for us to work on this challenging problem. In this paper, we conducted an experiment on seven datasets of different projects of Eclipse, namely CDTDebug (CD), EclipseDebug (Deb), EclipseJDTUI (TUI), EclipseSWT (SWT), EclipseUI (UI), IDEPlatform (IDE) and JDTUI (TUI2). Selection of training data from datasets of other six available projects has been done in an exhaustive way. We combined datasets in six projects other than the testing dataset project to create training candidates. Training candidates consisting of no more than three datasets of other projects have been used to train the prediction model because adding more datasets does not lead to improvement in the results. The number of combinations consisting of no more than  $K$  datasets for seven available datasets/projects can be calculated as

$$\sum_{i=1}^K \frac{7!}{i!(7-i)!} \quad (1)$$

In (1),  $K = 1, 2, 3$  results in 7 training datasets (single training datasets), 28 training datasets (single and combination of two training datasets) and 63 training datasets (single, combination of two and three training datasets). For every project, we get 63 training candidates out of which 41 training candidates are used to train the model because rest 22 training candidates contain the same dataset as the testing project.

3. Data Description and Methods

3.1. Data description

The present study has selected seven Eclipse projects (<https://bugs.eclipse.org/bugs/>) namely CDTDebug (CD), EclipseDebug (Deb), EclipseJDTUI (TUI), EclipseSWT (SWT), EclipseUI (UI), IDEPlatform (IDE) and JDTUI (TUI2). Eclipse uses Bugzilla for bug reporting. We have taken all the versions of the mentioned open source projects. The Bug reports have been collected for Eclipse from the start date (year 2003) of bug reporting till 4th August 2011 in Bugzilla reporting system.

The bug reports have been downloaded from bug reporting system Bugzilla using the search facility as shown in Fig. 1.

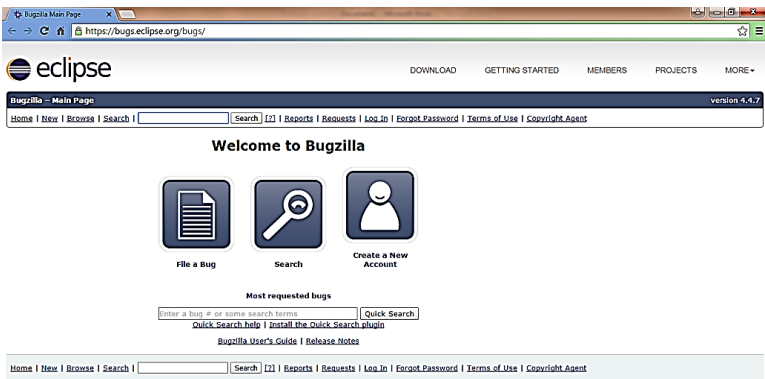


Fig. 1. Bugzilla bug reporting system for Eclipse software.

We have taken bug reports with resolution either “FIXED” or “WORKSFORME” (Fig. 2) because for these types of bug reports summary feature will remain unchanged.

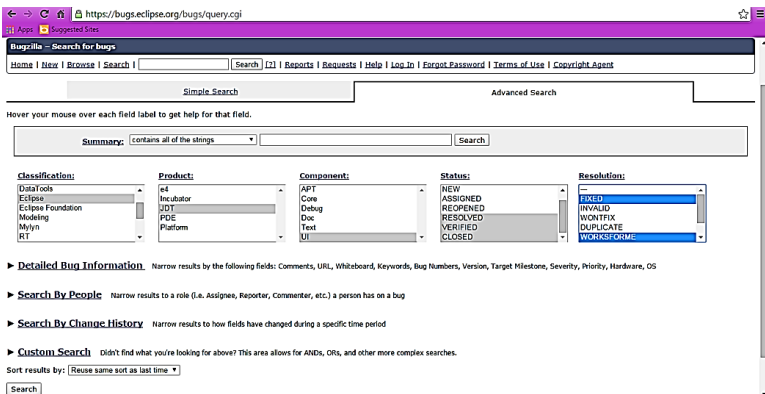


Fig. 2. Searching criteria for bug reports download.

Downloaded bug reports have been saved in excel format and used to build and train the models. The levels of severity attribute range from 1 to 5, i.e. from blocker to trivial. Severity-wise number of bug reports in various datasets of eclipse project has shown in Table 1.

Table 1. Severity-wise number of bug reports of Eclipse projects.

Projects	1 (Blocker)	2 (Critical)	3 (Major)	4 (Minor)	5 (Trivial)	Total
CDTDebug	25	25	122	53	8	233
EclipseDebug	23	97	213	72	39	444
EclipseJDTUI	23	81	282	281	81	748
EclipseSWT	71	161	298	64	36	630
EclipseUI	28	124	401	327	109	989
IDEPlatform	23	75	267	148	85	598
JDTUI	1	24	118	204	56	403

Figure 3 shows a sample of bug reports for the Eclipse JDTUI project.

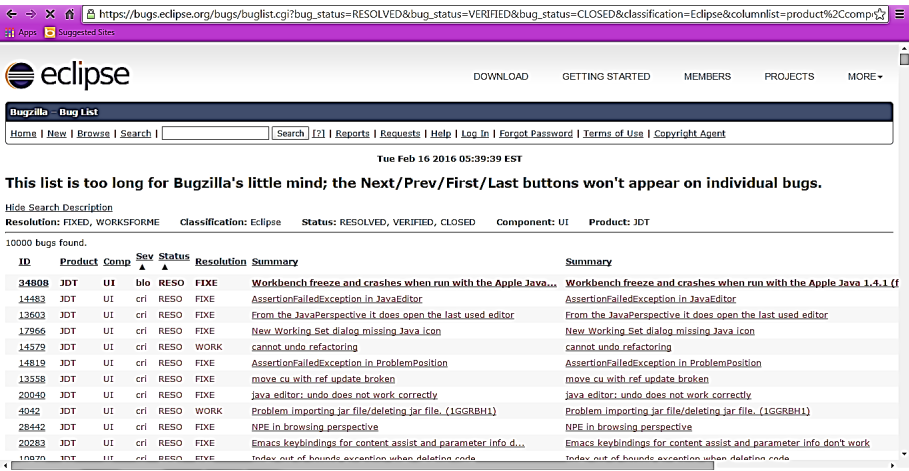


Fig. 3. Sample bug reports of EclipseJDTUI project.

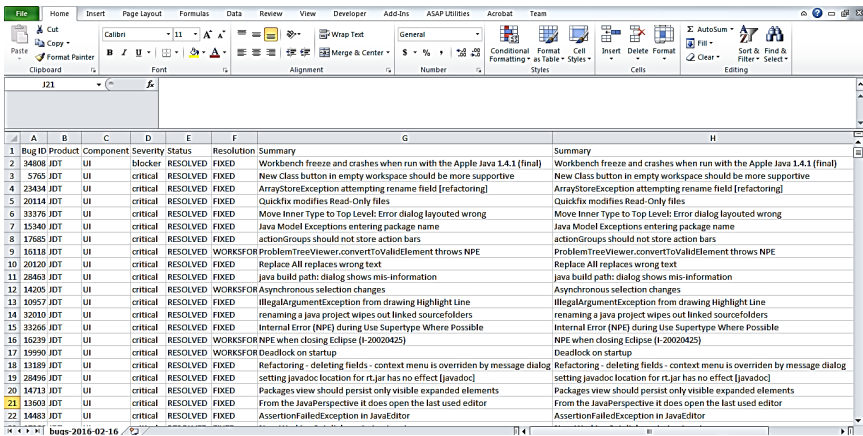
### 3.2. Methods

In this paper, terms of bug summary have been used to predict the severity of a reported bug. We have used SVM, NB and k-NN classification algorithms to classify the severity level of the newly coming bug report based on the already submitted bug reports. The foundations of SVMs have been developed by Vapnik (1995) and gained popularity because of its attractive features such as better empirical performance and a greater ability to generalise. SVMs were developed to solve the classification problem, but they have been extended to solve regression problems (Vapnik et al., 1997). Joachims (1997) defined SVM is a mathematical and statistical learning

technique having many applications started from handwriting recognition to genes identification. In a study, [Hsu and Lin \(2002\)](#) have done a comparison of methods for multi-class SVMs and showed that on the maximum margin hyperplane SVM tries to separate the report set into the given categories. In a review work, [Sun et al. \(2009\)](#) found that ensemble technique tends to yield better results when there is a significant diversity among the models. Ensembles have more flexibility in the functions. This flexibility can, in theory, enable them to overfit the training data more than a single model would, but in practice, some ensemble techniques, especially bagging tend to reduce problems related to overfitting of the training data.

#### 4. Models Building and Evaluation

In order to analyse the textual data of the summary attribute, preprocessing of the bug reports has to be done. This process consists of several steps. Studies conducted by [Bettenburg et al. \(2008\)](#) and [Yu et al. \(2010\)](#) showed that one line summary is sufficient to explain the longer description of the bug reports. This supports our approach to train classifiers with one line summary for our datasets, as the summary (full) and summary (one line) fields of bug reports in Bugzilla tracking system contain exactly the same text. Sample bug reports with summary (full) and summary (one line) fields have shown in Fig. 4.



Bug ID	Product	Component	Severity	Status	Resolution	Summary	Summary
34808	JDT	UI	blocker	RESOLVED	FIXED	Workbench freeze and crashes when run with the Apple Java 1.4.1 (final)	Workbench freeze and crashes when run with the Apple Java 1.4.1 (final)
5765	JDT	UI	critical	RESOLVED	FIXED	New Class button in empty workspace should be more supportive	New Class button in empty workspace should be more supportive
23434	JDT	UI	critical	RESOLVED	FIXED	ArrayStoreException attempting rename field [refactoring]	ArrayStoreException attempting rename field [refactoring]
20114	JDT	UI	critical	RESOLVED	FIXED	Quickfix modifies Read-Only files	Quickfix modifies Read-Only files
33376	JDT	UI	critical	RESOLVED	FIXED	Move Inner Type to Top Level: Error dialog layout wrong	Move Inner Type to Top Level: Error dialog layout wrong
15340	JDT	UI	critical	RESOLVED	FIXED	Java Model Exceptions entering package name	Java Model Exceptions entering package name
17685	JDT	UI	critical	RESOLVED	FIXED	actionGroups should not store action bars	actionGroups should not store action bars
16118	JDT	UI	critical	RESOLVED	WORKSFOR	ProblemTreeViewer.convertToValidElement throws NPE	ProblemTreeViewer.convertToValidElement throws NPE
20120	JDT	UI	critical	RESOLVED	FIXED	Replace All replaces wrong text	Replace All replaces wrong text
28403	JDT	UI	critical	RESOLVED	FIXED	java build path: dialog shows mis-information	java build path: dialog shows mis-information
14205	JDT	UI	critical	RESOLVED	WORKSFOR	Asynchronous selection changes	Asynchronous selection changes
11057	JDT	UI	critical	RESOLVED	FIXED	IllegalArgumentException from drawing Highlight Line	IllegalArgumentException from drawing Highlight Line
32010	JDT	UI	critical	RESOLVED	FIXED	renaming a java project wipes out linked source folders	renaming a java project wipes out linked source folders
13226	JDT	UI	critical	RESOLVED	FIXED	Internal Error (NPE) during Use Supertype Where Possible	Internal Error (NPE) during Use Supertype Where Possible
26259	JDT	UI	critical	RESOLVED	WORKSFOR	NPE when closing Eclipse (I-20020425)	NPE when closing Eclipse (I-20020425)
19990	JDT	UI	critical	RESOLVED	WORKSFOR	Deadlock on startup	Deadlock on startup
13189	JDT	UI	critical	RESOLVED	FIXED	Refactoring - deleting fields - context menu is overridden by message dialog	Refactoring - deleting fields - context menu is overridden by message dialog
28496	JDT	UI	critical	RESOLVED	FIXED	setting javadoc location for rt.jar has no effect [javadoc]	setting javadoc location for rt.jar has no effect [javadoc]
24713	JDT	UI	critical	RESOLVED	FIXED	Packages view should persist only visible expanded elements	Packages view should persist only visible expanded elements
13603	JDT	UI	critical	RESOLVED	FIXED	From the JavaPerspective it does open the last used editor	From the JavaPerspective it does open the last used editor
14483	JDT	UI	critical	RESOLVED	FIXED	AssertionFailedException in JavaEditor	AssertionFailedException in JavaEditor

Fig. 4. Sample bug reports of Eclipse with same summary (full) and summary (one line) fields.

The bug severity prediction models used in the paper are based on summary attribute entered by the user at the time of bug reporting. We preprocessed the bug summary to extract features/terms in RapidMiner tool (<http://www.rapid-i.com>) containing the following steps:

**Tokenisation:** Tokenisation is the process of breaking a stream of text into words, phrases, symbols, or other meaningful elements called tokens. In this paper a word or a term has been considered as a token.



*Stop Word Removal:* In bug summary, we have words which appear in the text often and do not have discriminate meaning. Such words are called stop words. Usually these are prepositions, conjunctions and articles. We have removed stop words from the summary.

*Stemming to Base Stem:* The mapping of a word to its root form is known as stemming. In this paper, we have used Standard Porter stemming algorithm for stemming given by Porter (1980).

*Feature Reduction:* The character length of feature/term rarely exceeds 50 (manually, we have observed from the collected data). Tokens of minimum length 3 and maximum length 50 have been filtered and considered. As frequently and less frequently occurring terms/features may not be able to discriminate the documents or may lead to overfit/underfit the model, the terms which has less than 3 or more than 40 occurrences have been removed from the dataset. Most of the data mining algorithms may not be able to handle large feature sets.

Aizawa *et al.* (2003) have shown that to make textual data structured for analysis, it is represented as document-term matrix where the rows are considered as documents or files and columns are considered as terms or tokens. The frequency of a term in the document will be counted and stored in the matrix form.  $TF_i$  is the occurrences of a term in the document. To normalise  $TF_i$  it is divided by the total number of terms ( $n$ ) in the document.

*TF \* IDF (Term Frequency (TF) Times Inverse Document Frequency (IDF)):* This weight is a statistical measure used to determine the importance of a term in the complete dataset or document set. The importance of a term increases with the frequency count of the term in the document, but is offset by the frequency of the word in the dataset. The term frequency is normalised to prevent a bias toward longer documents to give a measure of the importance of the term within a particular document. The inverse document frequency is obtained by dividing the number of all documents by the number of documents containing the term, and then taking the logarithm of that quotient. This measures the general importance of the term. TF \* IDF representation is used to rank the terms and selecting top few terms:

$$W_i = TF_i * IDF_i, \quad \text{where } IDF_i = \log \left( \frac{N}{DF_i} \right).$$

$DF_i$  is the document frequency which shows the appearance of a particular term in the number of documents and  $N$  is the total number of documents in the dataset.

*Weight by Information Gain or InfoGain:* In our study, the target concept is the distribution of severity levels. The best words/terms are those that mostly simplifies the target concept. We use information theory to measure concept “simplicity”. The number of bits required to encode an arbitrary class distribution  $C$  is  $H(C)$  defined as follows:

$$N = \sum_{c \in C} n(c),$$

$$p(c) = n(c)/N,$$

$$H(C) = - \sum_{c \in C} p(c) \log_2 p(c).$$
(2)

If  $A$  is a set of attributes, then the number of bits required to encode a class after observing an attribute is

$$H(C|A) = - \sum_{a \in A} p(a) \sum_{c \in C} p(c|a) \log_2 (p(c|a)).$$

The highest ranked attribute  $A_i$  is the one with the largest information gain, i.e. the one that mostly reduces the encoding required for the data after using that attribute:

$$\text{InfoGain}(A_i) = H(C) - H(C|A_i),$$

where  $H(C)$  comes from (2). The assumption was that the most informative terms are enough to fully describe their corresponding documents. We have used InfoGain to re-order the top  $k$  words selected from each bug report based on TF \* IDF. By using InfoGain to rank all the terms in the dataset, a significant dimensionality reduction is achieved.

We designed a process in RapidMiner to extract summary terms and build models using different classifiers to predict bug severity. Figure 5 shows the main process of cross project severity prediction for seven datasets used in this paper.

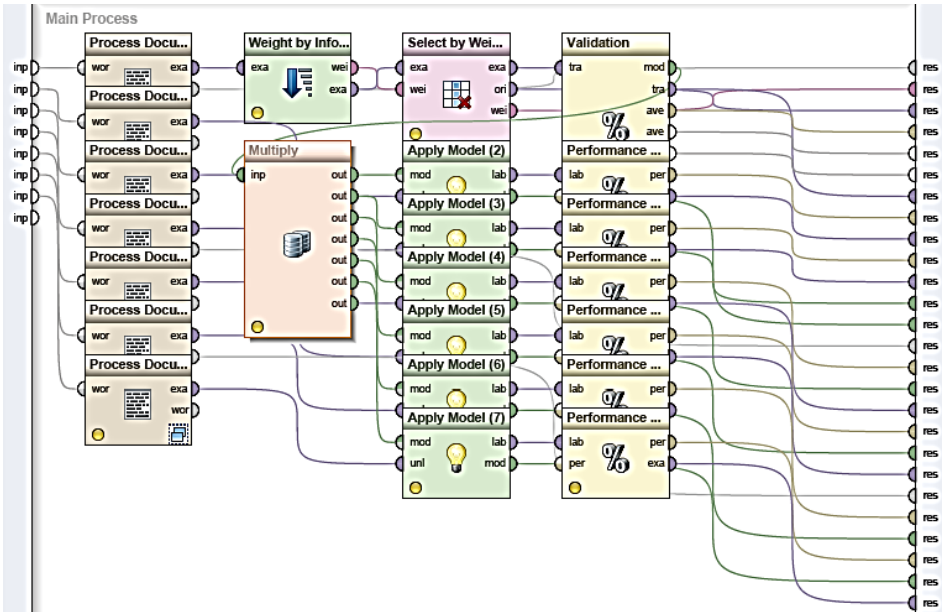


Fig. 5. Cross project bug severity prediction process in RapidMiner.

In the above process the first operator “Process Documents from Files” contains the inner subprocess for bug summary preprocessing to extract the terms as shown in Fig. 6.

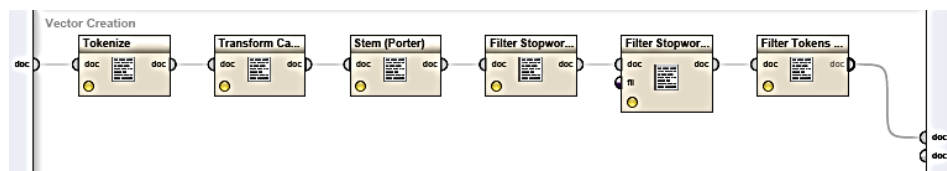


Fig. 6. Bug summary preprocessing process in RapidMiner.

The “Validation” operator in Fig. 5 contains an inner subprocess for training and testing with different classifiers as shown in Fig. 7.

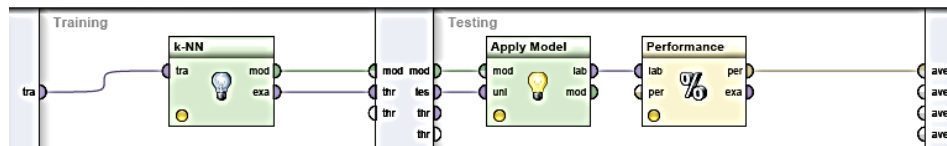


Fig. 7. Model building process in RapidMiner.

Table 2 shows the parameter values used for tuning the parameter  $k$  for k-NN and  $C$  (cost) and  $G$  (gamma) for SVM.

Table 2. Parameter values.

Classifier	Parameters
NB	No parameter
k-NN	$K$
SVM	$C$ (cost) = $\{2^{-5}, 2^{-3}, 2^{-1}, 2^1, 2^3\}$ $G$ (gamma) = $\{2^{-5}, 2^{-3}, 2^{-1}, 2^1, 2^3\}$

We obtained the optimal parameter values by using “Optimise Parameters (Grid)” operator in RapidMiner. Table 3 shows the optimal parameters values for each classifier according to each dataset.

Table 3. Optimal parameter values for Eclipse datasets.

Eclipse datasets	NB	k-NN	SVM	
	No parameter	$K$	$C$	$G$
CD	NA	11	2	8
Deb	NA	21	2	2

Table 3. (Continued)

Eclipse datasets	NB	k-NN	SVM	
	No parameter	$K$	$C$	$G$
TUI	NA	1	2	2
SWT	NA	21	2	2
UI	NA	11	8	2
IDE	NA	1	2	2
TUI2	NA	41	8	2

For different operators used to design the process (Figs. 5–7), we have used the parameter settings as shown in Table 4.

Table 4. Parameter settings for different operators.

Process documents from files	
vector creation	TF-IDF
prune method	Absolute
prune below absolute	3
prune above absolute	40
Tokenise	
Mode	Specify characters
Characters	[# “” (),==:; “.,”]
Transform case	
Transform	Lower case
Filter tokens	
min chars	3
max chars	50
Filter stop words (Dictionary)	
File	URL of self-created dictionary of words from processed documents
Select by weight	
weight relation	Top $k$
$K$	100
Validation ( $X$ -validation)	
number of validations	10
sampling type	Stratified sampling

Table 4. (Continued)

SVM	
Svmtype	C-SVC
Kernel	poly
Degree	3
Epsilon	0.001
Naïve Bayes	
laplace correction	Checked

In addition to the English dictionary of stop words, we have used a self-created dictionary of stop words which consists of some special characters/words and some numbers like error number, generated from the processed summary documents. We have constructed this dictionary by using an incremental approach from the processed summary documents and used it for the new summary documents to be processed.

In this paper, we have used terms of bug summary reports to predict the severity of a reported bug. Top 100 terms have been used in our study as we tuned the models with other numbers for top terms in the range of 10–150 with a step of 10 and found the optimal value at 100.

The two approaches of ensemble (Vote and Bagging) have been implemented to handle imbalance data problems and to improve the performance of cross project bug severity prediction.

For the first approach, Fig. 8 shows the inner subprocess for the operator “Validation” used in main process (Fig. 5).

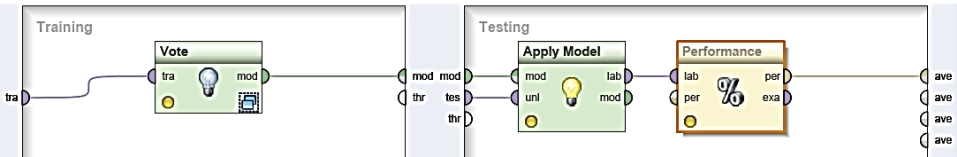


Fig. 8. Vote subprocess in RapidMiner.

Within “Vote” operator there is an inner subprocess as shown in Fig. 9. The same training dataset is supplied to all the three constituent classifiers. We used 10-fold cross validation with stratified sampling to split the training dataset randomly. Vote operator uses a majority vote on top of the predictions of the inner classifiers. We have taken equal voting weight for all the three classifiers.

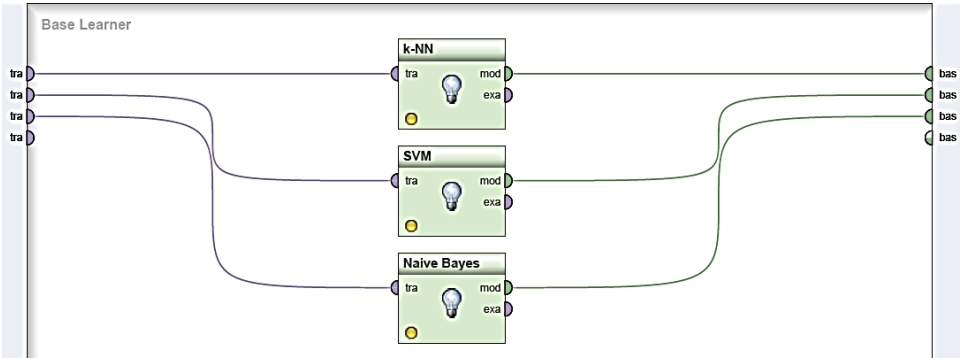


Fig. 9. Subprocess to make a combined prediction from different classifiers for same learning data.

For the second approach Fig. 10 shows the inner subprocess for the operator “Validation” used in main process (Fig. 5).

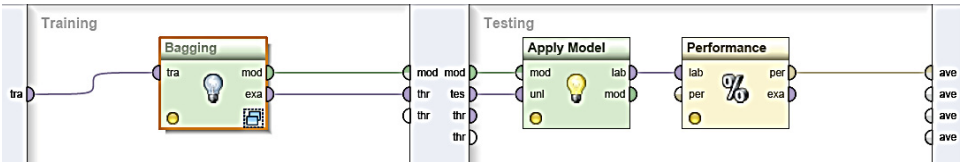


Fig. 10. Bagging subprocess in RapidMiner.

Within “Bagging” operator there is an inner subprocess as shown in Fig. 11.

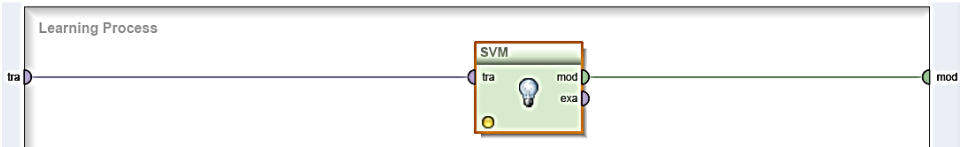


Fig. 11. Subprocess to make a combined prediction from the same classifier for different learning sub-samples of the original imbalance dataset.

For imbalance training dataset Bagging repeatedly creates sub-sample (with replacement) from the dataset, and make different classification models for each sub-sample using the same constituent classifier. The final classification is the one most often predicted by the different classification models. Similarly, we have created learning processes for k-NN and NB.

Different performance measures, namely accuracy, precision, recall and f-measure have been calculated to measure the performance of a classifier.

Accuracy of a classifier is defined as percentage of correct classification:

$$\text{Accuracy} = \frac{\text{No. of instances correctly classified}}{\text{Total number of instances}}.$$

Precision of a class A is defined as the number of instances correctly classified as class A divided by the total number of instances classified as class A. It measures the percentages of correct predictions related to the predictions made by the classifier:

$$\text{Precision} = \frac{\text{No. of instances correctly classified as class A}}{\text{Total number of instances classified as class A}}.$$

Recall of a class A is the number of instances correctly classified as class A divided by the total number of instances in the dataset having a class label A. It measures the percentage of correct predictions related to an actual class:

$$\text{Recall} = \frac{\text{No. of instances correctly classified as class A}}{\text{Total no. of instances in dataset having class label A}}.$$

f-measure is calculated to measure the average performance of the classifier to avoid the bias towards precision or recall. It is the harmonic mean of precision and recall:

$$\text{f-measure} = \frac{2 * \text{Precision} * \text{Recall}}{(\text{Precision} + \text{Recall})}.$$

To study and investigate the severity prediction and identification of training candidates in cross project context, we have set the following research questions:

**Research Question 1.** Can we predict bug severity in cross project context by using machine learning techniques?

**Research Question 2.** (a) Does the combination of training datasets help in improving the performance in case of cross project bug severity prediction?  
(b) Which training candidate is suitable for better prediction?

**Research Question 3.** Do the two ensemble approaches (Vote and Bagging) help in solving imbalance data problem in cross project bug severity prediction?

## 5. Experiments

We conducted two experiments on seven datasets of different Eclipse projects, namely CDTDebug (CD), EclipseDebug (Deb), EclipseJDTUI (TUI), EclipseSWT (SWT), EclipseUI (UI), IDEPlatform (IDE), and JDTUI (TUI2) to answer the three research questions. Three machine learning algorithms have been employed to construct the prediction models by using the open source data mining tool Rapid-Miner (Figs. 5–11).

**Experiment 1.** We conducted experiment 1 to answer the research questions 1 and 2. Training candidates by taking a combination of different datasets of collected bug reports have been developed. We used all the combinations consisting of maximum three datasets from other projects. This results in 63 combinations consisting of 1, 2, and 3 datasets for cross-project severity prediction. During making the combination of training datasets, we excluded dataset of the project, which is going to be tested and this results in 41 training candidates for each project. For example, for CD dataset the combinations of datasets other than CD project as training data like {Deb}, {IDE}, {Deb, IDE}, {Deb, SWT}, {Deb, IDE, SWT}, ..., {Deb, IDE, UI} have been used.

**Experiment 2.** We conducted experiment 2 to answer the research question 3. To handle the imbalance data problem we implemented two ensemble approaches by using Vote and Bagging operators available in RapidMiner. The complete setup of experiment 1 have been ran with all the three classifiers combined in Vote operator as shown in Figs. 8 and 9. We also ran the experiment once again for all the three classifiers with Bagging operator one at a time as shown in Figs. 10 and 11.

## 6. Results and Discussion

In answer to research question 1, we have conducted experiment 1. We set threshold values ranging from 40% to 100% to evaluate the performance of different classifiers. The results of f-measure in % have been shown in Tables A.1–A.3 for different classifiers. In all tables “—” shows that no experiment has been done for that particular combination of testing and training dataset because both training and testing datasets have common project.

Table A.1 shows the f-measure performance of k-NN machine learning technique for major severity class prediction. It is observed that for CD testing project the performance is more than 65% for 19 training candidates. In all these 19 cases the projects, namely TUI, TUI2, and UI have their individual f-measure performance more than 65%. Also, when we combine these projects with other training candidates whose individual performance is low as a single training candidate, the combined training candidate shows an improved performance, i.e. greater than 65%. We also found that these projects are similar to the testing project in their characteristics. These three training projects are user interface and CD is a debugger, which also handles the interfaces for multi-languages and multiple processors.

In case of Deb project all training candidates are giving performance greater than 60%. As Deb is a debugger project, it handles all the bugs coming from different projects, all the other projects are contributing towards its performance. For IDE project, the training candidates containing any one from CD, Deb and SWT give performance greater than 60%. All training candidates are giving performance



approximately 60% for SWT project. As SWT is a widget toolkit project, all other projects are contributing towards its prediction performance.

In case of TUI testing project, the contribution of TUI2 and UI is significant. All the training candidates containing TUI2 or UI alone or in a combination contribute towards improving the f-measure performance. In all such cases, the f-measure performance is greater than 60%.

For TUI2 project, UI project gives 60.34% performance. When TUI or UI combined with CD or Deb, it gives performance greater than 60%. TUI with combination of UI also gives performance greater than 60%. This shows that TUI and UI projects which have the characteristics similar to TUI2 project contribute towards improving the f-measure performance. For UI project, all training candidates having TUI, TUI2 or their combinations give f-measure performance greater than 60%. This shows that TUI and TUI2 projects which have the characteristics similar to UI project contribute towards improving the f-measure performance.

Table A.2 shows the f-measure performance for the SVM classifier. For CD, Deb, IDE and SWT testing projects, all training candidates' f-measure performance is greater than 60%, except for the training projects TUI, TUI2 and UI. In case of TUI testing project, all training candidates' f-measure performance is less than 60%, except for the training projects SWT and combination of TUI2 and UI. For TUI2 testing project, all training candidates' f-measure performance is less than 50%, except for the training project UI which is 57.39%. In case of UI testing project, CD, Deb, IDE and SWT as training candidates give f-measure performance greater than 60%. For all other training projects performance is less than 60%.

Table A.3 shows that for NB classifier, f-measure performance is below 34.25% for all training candidates. We deeply investigated the reason of low performance for NB and found that NB is not working well for cross project severity prediction as it strongly assumes feature independence and does not handle the class overlapping problem well. In our study, some training features/terms fall into more than one severity class. This can be take away message for researchers who are working in this area of research. Bold values in Tables A.1–A.3 show the best results of f-measure for every testing project for the respective classifier.

Figures 12–14 describe the performance of different classifiers for all severity levels by considering the number of cross validated bug severity prediction cases lying in the range of 40–100% by taking an interval of 10% for f-measure, precision and recall. As we have less number of bug reports for other severity levels in comparison of major severity level this results in low performance for these severity levels. For empirical validation, we have considered the performance of different machine learning techniques only for major severity level.

For f-measure ranging from 60% to 70%, we are getting the maximum cases for major severity level, i.e. 166 and 148 out of 287 for k-NN and SVM classifiers, respectively. For NB classifier, the f-measure performance across all the cases is below 34.25%.

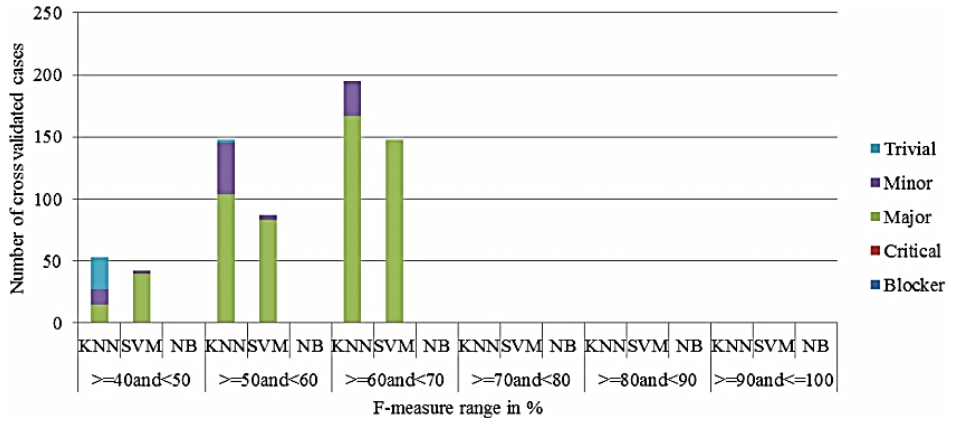


Fig. 12. f-measure performance of k-NN, SVM and NB for different bug severity levels.

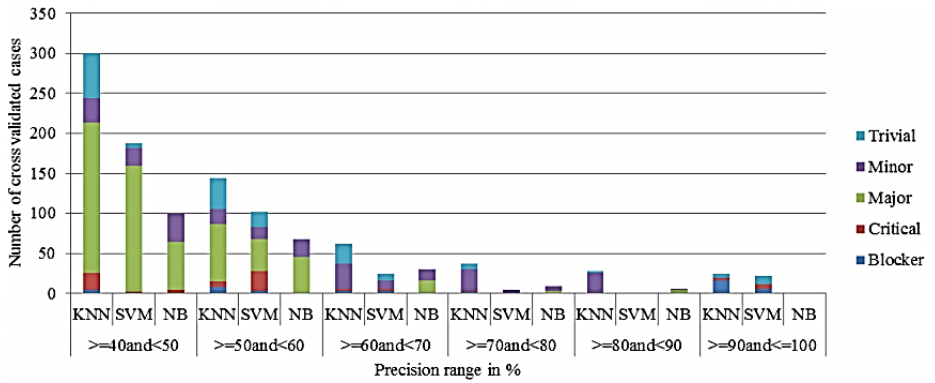


Fig. 13. Precision performance of k-NN, SVM and NB for different bug severity levels.

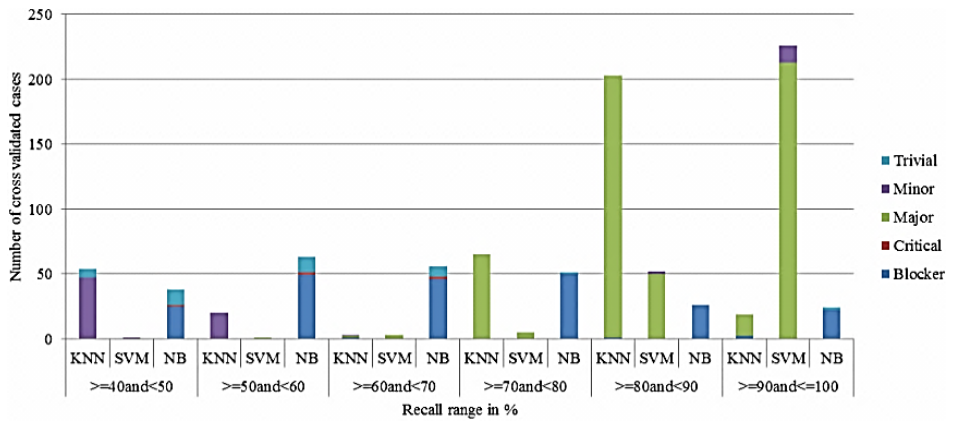


Fig. 14. Recall performance of k-NN, SVM and NB for different bug severity levels.

For precision ranging from 40% to 50%, we are getting the maximum cases for major severity level, i.e. 188,157 and 60 out of 287 for k-NN, SVM and NB classifiers, respectively.

For recall ranging from 90% to 100%, we are getting the maximum cases for major severity level, i.e. 213 out of 287 for SVM classifier. For k-NN, recall ranging from 80% to 90%, we are getting the maximum cases for major severity level, i.e. 202 out of 287. Due to highly skewed data (i.e. one or more rare severity classes 1, 2, 4, 5 and one prominent severity class 3), machine learning algorithms, SVM and k-NN just classify the severity level of a newly coming bug report as prominent severity class 3, yielding high recall but low precision.

Results show that k-NN performance is better than the SVM performance in terms of f-measure. NB performance is poor in terms of f-measure.

In the answer of research question 2, from Table A.4, we conclude that the combination of training datasets results in improvement of f-measure performance for k-NN classifier for six testing datasets out of seven testing datasets. In case of SVM classifier, combinations of training datasets result in improvement of f-measure performance for two testing datasets out of seven testing datasets. For NB classifier also we get improvement in f-measure performance for combined training datasets for four testing datasets out of seven testing datasets. In case of NB classifier combined training datasets lead to improvement in f-measure performance for four testing datasets. But, in case of k-NN classifier combined training datasets lead to improvement in f-measure performance. Characteristics of data are crucial factors for the success of cross project severity prediction. In our research, we have seen that the combining training datasets helps in the selection of best training candidate in case of all used algorithms (k-NN, SVM and NB).

From Table A.4, we concluded that across all the classifiers, on the basis of performance measure, f-measure, IDE+TUI is the best training candidate for CD testing dataset, CD is the best training candidate for Deb, IDE, SWT and UI testing datasets, TUI2+UI is best training candidate for TUI testing dataset and TUI+UI is the best training candidate for TUI2 testing dataset.

Table A.5 shows that a combination of training datasets results in improvement of accuracy performance in case of all seven testing datasets for the k-NN classifier. In case of SVM classifier, the combinations of training datasets result in improvement of accuracy performance for two testing datasets out of seven testing datasets. For NB classifier, we get an improvement in the accuracy performance for combined training datasets for one testing datasets out of seven testing datasets. Many cases have accuracy less than 50%. Built-in prediction models of such cases may not be practically applicable for bug severity prediction.

In our research, we observed that combining datasets helps in the selection of best training candidate in case of k-NN classifier only in terms of f-measure and accuracy performance measures. For SVM and NB classifiers, single training datasets having characteristics similar to the testing datasets alone are sufficient for cross severity prediction.

In the answer of research question 3, we applied Vote ensemble approach to make a combined prediction (using simple voting) of the three constituent classifiers, namely k-NN, NB and SVM. When we compared the accuracy performance of NB with Vote accuracy performance, the performance has improved in all 287 cases. In case of SVM, the accuracy performance has improved in 192 cases out of total 287 cross project severity prediction cases and degraded in 85 cases. In rest 10 cases it remains unchanged. In case of k-NN, the accuracy performance has improved in 117 cases out of total 287 cross project severity prediction cases and degraded in 162 cases. In rest eight cases it remains unchanged.

The f-measure performance has improved from 2% to 5% for severity levels where number of bug reports are less in comparison of bug reports in severity level 3 when we make a combined prediction of all the three classifiers by using the Vote ensemble approach. We have compared Vote performance with k-NN performance as k-NN performance was best among all the three classifiers.

We applied Bagging ensemble approach with k-NN, SVM, NB and found that f-measure performance has improved to 10% for severity levels where number of bug reports is low in comparison of number of bug reports in severity level 3.

In this paper, after preprocessing of bug reports of different datasets in Rapid-Miner tool we found 110, 182, 322, 254, 311, 191 and 403 terms in CD, Deb, IDE, SWT, TUI, TUI2 and UI, respectively. Out of which 19 terms were common across all the seven datasets. We have also computed the common terms between the testing and the different combined training candidates and found that the common terms are increasing and decreasing for different cases. In order to analyse that how number of common terms are increasing or decreasing with increasing the size of training candidate we calculated the number of common terms between TUI2 testing dataset and different training candidates like TUI, TUI+UI and Deb+TUI+UI. We found that number of common terms between TUI2 testing dataset and different training candidates, i.e. TUI, TUI+UI and Deb+TUI+UI are 175, 145 and 139, respectively. Similarly, the number of common terms between CD testing dataset and different training candidates, i.e. Deb, Deb+SWT and Deb+SWT+TUI are 72, 76 and 70, respectively. We have also observed the similar increase and decrease in common terms for different testing and training combinations. The reason behind this is that the distribution of different terms belonging to different severity levels is not uniform.

To test the statistical significance of the difference in f-measure performance of different training candidates for a particular testing dataset, we stated the following null and alternative hypothesis.

$H_0$ : The difference in performance (f-measure) of training candidates in cross project severity prediction is not significant.

$H_1$ : The difference in performance (f-measure) of training candidates in predicting cross project severity prediction is significant.

We have tested the given hypothesis by using Non-parametric Mann–Whitney U test in Statistical Package for Social Sciences (SPSS). At a time we have taken two sets of f-measure values where each set comprises of f-measure values of six testing datasets other than the training datasets to be tested for performance difference. The results for k-NN, SVM and NB classifiers in cross project severity prediction of statistical test, namely asymptotic significance (2-tailed) are shown in Table A.6.

We set the significance level,  $\alpha$ , the probability of making a Type I error, i.e. 0.05. We compared the  $p$ -value to  $\alpha$ . If the  $p$ -value is less than (or equal to)  $\alpha$ , we reject the null hypothesis in favour of the alternative hypothesis. If the  $p$ -value is greater than  $\alpha$ , we do not reject the null hypothesis. From Table A.6 it is clear that for k-NN classifier there is a significant difference in the 1 case (SWT, TUI2) out of total 21 cases. In this case  $p$ -value is less than 0.05, hence we reject the null hypothesis. In other cases ( $p > 0.05$ ), data do not provide statistically significant evidence of a difference between the f-measure performance of different training datasets. For SVM classifier there is a significant difference in 5 cases ((Deb,UI), (TUI,UI), (SWT, UI) (UI,IDE) and (UI,TUI2)) out of total 21 cases at a significance level of 0.05 as  $p$ -value is less than 0.05, hence we reject the null hypothesis for these five cases. For NB classifier, no training datasets are significantly different in terms of f-measure performance at significance level of 0.05.

We motivate this paper with a cross project bug severity prediction between Eclipse and Mozilla products. In this context, we have further analysed our study with two products of Mozilla project (<https://bugzilla.mozilla.org/>): Seamonkey and Firefox. The description of bug reports is shown in Table 5.

From the results of f-measure for major severity class across all the 63 training candidates of Eclipse products, we observe that the value of f-measure performance for Seamonkey lies in the range of 0.45–49.67%, 46.16–49.1% and 1.1–10.77% for SVM, k-NN and NB classifiers, respectively. Based on the maximum f-measure performance, SWT+TUI+UI, IDE and CD are the best training candidates for SVM, k-NN and NB classifiers, respectively. For Firefox, f-measure performance lies in the range of 3.03–59.90%, 48.65–59.79% and 4.45–25.92% for SVM, k-NN and NB classifiers, respectively. Based on the maximum f-measure performance CD+Deb, Deb and TUI2 are the best training candidates for SVM, k-NN and NB classifiers, respectively. We concluded that across all the classifiers, on the basis of performance measure, f-measure, SWT+TUI+UI is the best training candidate for Seamonkey testing dataset and Deb is the best training candidate for Firefox testing dataset.

Table 5. Severity-wise number of bug reports of Mozilla products.

Projects	1 (Blocker)	2 (Critical)	3 (Major)	4 (Minor)	5 (Trivial)	Total
Firefox	59	109	259	128	53	608
Seamonkey	508	688	871	381	225	2673

Across all the classifiers, on the basis of performance measure accuracy, IDE is the best training candidate for Seamonkey testing dataset and Deb is the best training candidate for Firefox testing dataset.

We tested all Eclipse products for severity prediction by taking Seamonkey and Firefox as training candidates. In case of SVM classifier, Seamonkey and Firefox give maximum f-measure, i.e. 66.67% and 68.05% for CD product. For k-NN classifier, Seamonkey gives maximum f-measure, i.e. 65.47% for CD product and Firefox gives maximum f-measure, i.e. 62.32% for Deb product. Seamonkey and Firefox give f-measure less than 12.23% for all the Eclipse products when NB classifier is used. In case of SVM classifier, Seamonkey and Firefox give maximum accuracy, i.e. 49.36% and 50.64% for CD product. For k-NN also, Seamonkey and Firefox give maximum accuracy, i.e. 48.50% and 43.78% for CD product. In case of NB, accuracy is less than 16.39% for Seamonkey and less than 29.03% for Firefox across all the Eclipse products.

We concluded that Mozilla products can be used to build reliable prediction models for Eclipse products and vice versa in case of SVM and k-NN classifiers.

## 7. Threats to Validity

Following are the factors that affect the validity of our approach:

*Internal Validity:* Our study is based on bug severity prediction using the summary feature of bug in cross project context. The best training candidates have been identified from the available cross project training datasets. We do not claim any general relationship in the findings. More investigation is required to find out how to best describe the selection of a training project automatically.

*External Validity:* In this paper, we studied seven different Eclipse projects, namely CDTDebug(CD), EclipseDebug (Deb), EclipseJDTUI (TUI), EclipseSWT (SWT), EclipseUI (UI), IDEPlatform (IDE) and JDTUI (TUI2). Although these open source products have good quality bug reports and have been used in [Chaturvedi and Singh \(2013\)](#), our results may not generalise to other software.

*Construct Validity:* We have used the bug summary feature available in Bugzilla and preprocessed it by using text mining. We have built bug severity prediction models and ensemble approaches by designing processes in RapidMiner. These processes have been developed manually and may contain any manual errors which can be a threat to the accuracy of the process. The processes developed in our study may give varied results for different data mining tools. Bugzilla repositories are constantly changing, and some of the bugs studied may be reopened in the near future and severity of them may be changed.

## 8. Conclusion

This paper investigates the bug severity prediction by identifying training data from other projects. Also, ensembles are used to add diversity and therefore deal with the

problems that may be caused by imbalance data. We conducted two exhaustive experiments with seven projects of Eclipse and three machine learning techniques. Results show that k-NN performance is better than the SVM performance in terms of different performance measures. NB f-measure performance is poor, i.e. below 34.25%. In case of k-NN, developing training candidates by combining more than one training datasets helps in improving the performance (f-measure and accuracy) in cross project bug severity prediction. Our results indicate that cross-project bug severity prediction is a serious challenge, i.e. simply using models from projects in the same domain does not lead to accurate predictions. The selection of project for developing training candidate needs to be done carefully. More research is needed to find out how to best describe the selection of a training project automatically. Vote and Bagging ensemble approaches help in solving imbalance data problem in case of cross project bug severity prediction. The two approaches have improved the f-measure performance by 5% and 10% respectively for the severity levels where bug reports are less in numbers. We motivate this paper with a cross project bug severity prediction between Eclipse and Mozilla products. Results show that Mozilla products can be used to build reliable prediction models for Eclipse products and vice versa in case of SVM and k-NN classifiers. In the future, we will develop classifiers for cross project bug severity predictions using fuzzy logic-based similarity measures.

Appendix A.

Table A.1. KNN f-measure (%) for major severity class.

<div>Testing</div> <div>Training</div>	CD	Deb	TUI	SWT	UI	IDE	TUI2
CD	—	63.9	53.67	<b>63.95</b>	56.95	61.59	44.54
Deb	62	—	54.49	62.02	56.71	60.85	44.85
TUI	65.57	64.61	—	58.75	60.56	0	43.38
SWT	63.43	59.9	52.16	—	54.22	20.28	43.38
UI	67.5	62.17	60.79	61.3	—	56.31	60.34
IDE	62.92	64.11	53.61	61.99	56.71	—	43.78
TUI2	67.37	60.9	63.85	57.65	62.54	60.9	—
CD+Deb	—	—	52.56	61.92	57.12	60.19	41.45
CD+IDE	—	63.18	53.72	60.41	56.17	—	43.27
CD+SWT	—	63.37	60.99	—	56.08	52.37	44.65
CD+TUI	—	62.8	—	59.25	60.38	58.69	60.41
CD+TUI2	—	62.74	59.88	60.48	61.56	<b>62.96</b>	—
CD+UI	—	60.58	61.02	61.52	—	58.69	60.29
Deb+IDE	63.79	—	52.56	60.54	53.93	—	43.23
Deb+UI	67.31	—	57.86	60.54	—	58.45	56.64
Deb+TUI2	64.92	—	60.24	61.89	61.02	55.86	—
Deb+TUI	64.03	—	—	57.75	60	58.05	61.4
Deb+SWT	60.45	—	55.06	—	54.88	60.76	42.03
IDE+SWT	62.86	63.14	53.67	—	57.03	—	44.69
IDE+TUI	59.93	63.49	—	61.2	59.5	—	58.59

Table A.1. (Continued)

Testing Training	CD	Deb	TUI	SWT	UI	IDE	TUI2
IDE+TUI2	62.92	64.51	58.75	62.31	61.41	—	—
IDE+UI	65.44	63.78	58.77	61.7	—	—	58.82
SWT+UI	66.06	63.48	59.38	—	—	61.69	57.84
SWT+TUI2	65.83	64.61	59.45	—	59.93	56.74	—
SWT+TUI	65.42	63.26	—	—	59.59	60	56.57
TUI+UI	67.08	63.93	—	59.9	—	55.75	<b>61.91</b>
TUI+TUI2	63.67	62.11	—	59.59	62.71	53.76	—
TUI2+UI	<b>68.15</b>	61.37	<b>64.52</b>	60.71	—	55.84	—
CD+Deb+IDE	—	—	53.95	61.43	56.32	—	45.48
CD+Deb+SWT	—	—	54.56	—	55.36	58.83	40.67
CD+Deb+TUI	—	—	—	59.16	60.39	57.02	56.59
CD+Deb+TUI2	—	—	61.01	60.51	61.62	58.69	—
CD+Deb+UI	—	—	57.07	59.76	—	60.05	56.09
CD+IDE+SWT	—	62.65	54.6	—	54.98	—	45.49
CD+IDE+TUI	—	64.2	—	60.84	60.62	—	57.85
CD+IDE+TUI2	—	63.4	61.07	62.28	61.18	—	—
CD+IDE+UI	—	62.68	58.26	61.52	—	—	56.84
CD+SWT+UI	—	62.01	58.77	—	—	58.77	54.29
CD+SWT+TUI2	—	62.7	59.59	—	58.52	59.09	—
CD+SWT+TUI	—	63.86	—	—	60.24	60.28	56.04
CD+TUI+TUI2	—	62.19	—	60.76	<b>62.82</b>	55.72	—
CD+TUI+UI	—	64.34	—	60.57	—	58.46	58.59
CD+TUI2+UI	—	62	62.08	61.08	—	58.06	—
Deb+IDE+SWT	61.78	—	52.8	—	56.14	—	46.81
Deb+IDE+TUI	66.25	—	—	61.63	60.13	—	55.41
Deb+IDE+TUI2	66.67	—	60.53	61.94	58.83	—	—
Deb+IDE+UI	64.64	—	58.31	62.62	—	—	55.82
Deb+SWT+UI	64.76	—	59.37	—	—	62.27	55.58
Deb+SWT+TUI2	62.95	—	60.67	—	59.25	61.56	—
Deb+SWT+TUI	67.29	—	—	—	59.77	60.42	59.52
Deb+TUI+UI	67.28	—	—	61.98	—	59.33	59.89
Deb+TUI+TUI2	61.43	—	—	60.17	61.96	56.14	—
Deb+TUI2+UI	67.28	—	61.16	61.83	—	57.02	—
IDE+TUI2+UI	66.25	63.65	61.48	61.79	—	—	—
IDE+TUI+UI	62.86	62.18	—	61.5	—	—	59.3
IDE+TUI+TUI2	63.84	64.36	—	63.83	61.48	—	—
IDE+SWT+TUI2	63.26	62.81	56.84	—	58.92	—	—
IDE+SWT+TUI	62.35	<b>65.79</b>	—	—	60.31	—	55.84
IDE+SWT+UI	64.2	63.92	59.23	—	—	—	57.64
SWT+TUI+UI	66.86	64.02	—	—	—	58.92	60.82
SWT+TUI2+UI	65.04	62.54	61.15	—	—	60.82	—
SWT+TUI+TUI2	65.14	62.87	—	—	61.97	57.59	—
TUI+TUI2+UI	64.84	63.21	—	60.69	—	57.61	—



Table A.2. SVM f-measure (%) for major severity class.

Testing Training	CD	Deb	TUI	SWT	UI	IDE	TUI2
CD	—	<b>69.94</b>	55.59	<b>68.39</b>	<b>67.28</b>	<b>67.81</b>	—
Deb	64.93	—	57.46	64.83	62.44	64.27	2.68
TUI	54.58	53.83	—	54.26	53.02	53.62	2.69
SWT	64.22	63.81	60.72	—	62.18	64.4	0.66
UI	57.8	57.83	53.87	57.29	—	57.39	<b>57.39</b>
IDE	61.8	61.7	57.98	61.86	61.19	—	0.73
TUI2	44.79	44.28	36.72	45.17	43.7	2.91	—
CD+Deb	—	—	54.06	63.72	57.21	61.38	45.01
CD+IDE	—	63.08	53.54	63.8	57.28	—	44.18
CD+SWT	—	65.02	54.15	—	57.57	61.66	45.05
CD+TUI	—	62.93	—	63.62	55.59	62.5	43.88
CD+TUI2	—	13.18	14.25	10.08	14.41	9.84	—
CD+UI	—	64.09	53.82	63.68	—	61.29	44.71
Deb+IDE	69.44	—	53.32	63.42	57.06	—	42.94
Deb+UI	65.83	—	53.76	63.11	—	61.97	45.53
Deb+TUI2	66.66	—	51.75	62.8	56.37	60.97	—
Deb+TUI	66.66	—	—	64.09	57.38	61.36	46.13
Deb+SWT	69.82	—	54.15	—	57.57	61.66	45.05
IDE+SWT	69.62	64.67	54.38	—	57.99	—	43.02
IDE+TUI	<b>69.97</b>	62.3	—	64.16	56.05	—	46.02
IDE+TUI2	64.42	62.8	51.45	62.6	55.2	—	—
IDE+UI	67.25	61.34	54.32	63.77	—	—	45.73
SWT+UI	67.46	62.01	54.6	—	—	62.2	44.59
SWT+TUI2	68.06	65.11	53.19	—	57.31	62.28	—
SWT+TUI	68.78	64.27	—	—	57.63	60.68	44.13
TUI+UI	68.06	61.9	—	62.32	—	60.37	45.71
TUI+TUI2	66.88	13.64	—	12.08	20.24	15.48	—
TUI2+UI	66.88	62.54	<b>61.38</b>	53.17	—	61.36	—
CD+Deb+IDE	—	—	53.13	63.33	57.22	—	42.88
CD+Deb+SWT	—	—	54.24	—	56.71	61.5	45.31
CD+Deb+TUI	—	—	—	63.8	56.93	61.76	44.85
CD+Deb+TUI2	—	—	55.15	63.97	56.53	60.65	—
CD+Deb+UI	—	—	54.43	63.4	—	61.67	45.64
CD+IDE+SWT	—	62.94	53.57	—	57.73	—	43.59
CD+IDE+TUI	—	62.95	—	63.37	57.14	—	45.04
CD+IDE+TUI2	—	60.64	51.89	63.46	56.05	—	—
CD+IDE+UI	—	63.3	54.66	63.02	—	—	45.51
CD+SWT+UI	—	62.74	52.66	—	—	62.29	45.36
CD+SWT+TUI2	—	63.47	52.57	—	56.78	61.89	—
CD+SWT+TUI	—	63.28	—	—	57.5	62.05	43.8
CD+TUI+TUI2	—	56.69	—	61.37	49.7	58.03	—
CD+TUI+UI	—	63.74	—	63.08	—	61.32	43.26
CD+TUI2+UI	—	62.23	54.17	63.78	—	60.81	—
Deb+IDE+SWT	68.26	—	53.89	—	57.37	—	42.48
Deb+IDE+TUI	69.25	—	—	63.88	57.58	—	44.09
Deb+IDE+TUI2	67.29	—	52.54	63.09	56.38	—	—
Deb+IDE+UI	67.25	—	54.21	63.15	—	—	45.58
Deb+SWT+UI	67.06	—	54.08	—	—	62.22	45.36
Deb+SWT+TUI2	67.48	—	51.14	—	55.9	62.04	—
Deb+SWT+TUI	69.76	—	—	—	57.57	62.55	47.54

Table A.2. (Continued)

Testing Training	CD	Deb	TUI	SWT	UI	IDE	TUI2
Deb+TUI+UI	66.06	—	—	63.59	—	61.06	45.49
Deb+TUI+TUI2	64.13	—	—	61.91	56.42	60.03	—
Deb+TUI2+UI	67.06	—	53.49	62.72	—	61.75	—
IDE+TUI2+UI	65.87	63.14	54.8	63.11	—	—	—
IDE+TUI+UI	67.46	64.17	—	62.83	—	—	45.99
IDE+TUI+TUI2	67.28	59.74	—	62.84	54.16	—	—
IDE+SWT+TUI2	66.08	63.16	52.38	—	56.5	—	—
IDE+SWT+TUI	66.86	63.4	—	—	57.25	—	45.84
IDE+SWT+UI	67.06	63.9	53.18	—	—	—	41.89
SWT+TUI+UI	67.06	62.01	—	—	—	60.6	45.83
SWT+TUI2+UI	69.18	62.66	56.02	—	—	60.99	—
SWT+TUI+TUI2	67.25	61.51	—	—	55.66	61.24	—
TUI+TUI2+UI	65.07	62.29	—	63.76	—	61.14	—

Table A.3. NB f-measure (%) for major severity class.

Testing Training	CD	Deb	TUI	SWT	UI	IDE	TUI2
CD	—	<b>33.33</b>	8.89	<b>16.33</b>	10.22	13.14	13.05
Deb	<b>34.25</b>	—	5.89	9.24	9.35	5.11	10.31
TUI	21.05	12.86	—	9	6.26	9.15	15.01
SWT	21.25	12.68	6.75	—	7.21	12.12	15.6
UI	22.14	17.93	6.01	13.04	—	9.4	20.12
IDE	19.37	16.16	12.3	12.78	5.25	—	15.25
TUI2	17.53	13.25	12.86	16.25	7.14	11.19	—
CD+Deb	—	—	<b>15.39</b>	8.51	14.8	<b>19.53</b>	12.42
CD+IDE	—	12.05	13.91	4.4	7.19	—	4.41
CD+SWT	—	6.2	5.13	—	6.26	4.81	5.88
CD+TUI	—	9.67	—	7.87	6.3	7.43	13.69
CD+TUI2	—	17.06	7.37	6.92	13.61	4.86	—
CD+UI	—	10.08	3.68	6.42	—	8.88	5.3
Deb+IDE	14.09	—	9.35	8.92	12.03	—	<b>23.37</b>
Deb+UI	7.47	—	9.22	5.14	—	5.6	2.61
Deb+TUI2	8.64	—	14.99	1.3	8.53	6.06	—
Deb+TUI	12.95	—	—	14.24	8.81	9.49	13.41
Deb+SWT	8.76	—	8.69	—	12.04	7.99	11.92
IDE+SWT	11.43	16	12.19	—	10.4	—	11.43
IDE+TUI	13.8	12.6	—	13.94	10.09	—	8.27
IDE+TUI2	12.86	12.45	13.7	12.5	<b>17.37</b>	—	—
IDE+UI	17.81	10.85	14.71	10.34	—	—	14.97
SWT+UI	3.18	2.71	7.41	—	—	7.03	5.48
SWT+TUI2	7.82	11.06	12	—	7.36	6.95	—
SWT+TUI	10.45	10.04	—	—	4.88	5.72	5.41
TUI+UI	3.08	11.47	—	6.92	—	7.99	—
TUI+TUI2	11.68	7.54	—	7.76	14.23	16.3	—
TUI2+UI	4.62	10.16	9.66	2.54	—	9.62	—

Table A.3. (Continued)

Testing Training	CD	Deb	TUI	SWT	UI	IDE	TUI2
CD+Deb+IDE	—	—	9.94	3.76	9.32	—	4.54
CD+Deb+SWT	—	—	—	—	5.33	9	4.19
CD+Deb+TUI	—	—	—	7.44	6.7	14.47	9.59
CD+Deb+TUI2	—	—	13.38	9.31	12.8	8.1	—
CD+Deb+UI	—	—	7.46	3.28	—	5.5	3.92
CD+IDE+SWT	—	16.19	5.64	—	6.06	—	9.15
CD+IDE+TUI	—	12.05	—	4.24	4.67	—	4.22
CD+IDE+TUI2	—	14.62	13.34	6.09	11.11	—	—
CD+IDE+UI	—	18.19	10.21	7.23	—	—	4.51
CD+SWT+UI	—	7.8	5.96	—	—	8.27	5.41
CD+SWT+TUI2	—	6.9	5.45	—	6.68	10.56	—
CD+SWT+TUI	—	11.96	—	—	5.89	4.98	5.52
CD+TUI+TUI2	—	9.08	—	5.52	6.82	10.24	—
CD+TUI+UI	—	13.06	—	6.94	—	4.98	4.34
CD+TUI2+UI	—	5.96	6.03	6.81	—	4.86	—
Deb+IDE+SWT	27.45	—	10.03	—	9.61	—	9.59
Deb+IDE+TUI	22.97	—	—	6.12	11.27	—	8.5
Deb+IDE+TUI2	11.04	—	10.27	5.07	—	—	—
Deb+IDE+UI	23.75	—	10.44	9.82	—	—	13.25
Deb+SWT+UI	3.05	—	5.1	—	—	3.54	4.38
Deb+SWT+TUI2	4.65	—	6.69	—	9.86	5.56	—
Deb+SWT+TUI	15.61	—	—	—	3.91	2.16	12.82
Deb+TUI+UI	9.03	—	—	6.38	—	8.16	9.09
Deb+TUI+TUI2	14.6	—	—	4.45	4.56	2.89	—
Deb+TUI2+UI	7.7	—	9.2	1.32	—	4.82	—
IDE+TUI2+UI	10	11.81	15.34	1.33	—	—	—
IDE+TUI+UI	12.22	6.96	—	1.95	—	—	9.09
IDE+TUI+TUI2	15.9	10.21	—	8.24	7.1	—	—
IDE+SWT+TUI2	8.76	9.35	7.46	—	8.57	—	—
IDE+SWT+TUI	10.45	3.53	—	—	7.71	—	14.86
IDE+SWT+UI	4.65	9.6	4.66	—	—	—	3.07
SWT+TUI+UI	6.25	3.56	—	—	—	6.36	7.15
SWT+TUI2+UI	6.3	3.62	7.44	—	—	7.61	—
SWT+TUI+TUI2	12.13	3.59	—	—	4.43	4.31	—
TUI+TUI2+UI	6.25	1.8	—	7.67	—	7.56	—

Table A.4. Classifier-wise best training candidate with maximum f-measure.

Testing datasets	Best training dataset by SVM (f-measure %)	Best training dataset by KNN (f-measure %)	Best training dataset by NB (f-measure %)
CD	IDE+TUI (69.97)	TUI2+UI (68.15)	Deb (34.25)
Deb	CD (69.94)	IDE+SWT+TUI (65.79)	CD (33.33)
IDE	CD (67.81)	CD+TUI2 (62.96)	CD+Deb (19.53)
SWT	CD (68.39)	CD (63.95)	CD (16.33)
TUI	TUI2+UI (61.38)	TUI2+UI (64.52)	CD+Deb (15.39)
TUI2	UI (57.39)	TUI+UI (61.91)	Deb+IDE (23.37)
UI	CD (67.28)	CD+TUI+TUI2 (62.82)	IDE+TUI2 (17.37)

Table A.5. Classifier-wise best training candidate with maximum accuracy.

Testing dataset	Best training dataset by SVM (Accuracy %)	Best training dataset by KNN (Accuracy %)	Best training dataset by NB (Accuracy %)
CD	Deb (53.22)	TUI2+UI (52.79)	Deb (21.03)
Deb	CD (47.97)	IDE+SWT+TUI (49.77)	CD (23.65)
IDE	SWT+TUI (45.32)	CD+TUI2 (56.15)	UI (18.06)
SWT	CD (47.46)	IDE+TUI+TUI2 (46.19)	IDE+TUI (16.19)
TUI	TUI2+UI (42.70)	TUI2+UI (58.82)	UI (22.59)
TUI2	TUI (36.97)	SWT+TUI+UI (61.54)	IDE (18.86)
UI	TUI (41.96)	Deb+TUI+TUI (53.89)	UI (24.37)

Table A.6. Mann–Whitney U test results ( $p$ -values) for KNN, SVM and NB.

Training dataset	Training dataset	Significance difference in KNN f-measure	Significance difference in SVM f-measure	Significance difference in NB f-measure
		Asymptotic significance (2-tailed)	Asymptotic significance (2-tailed)	Asymptotic significance (2-tailed)
CD	Deb	0.749	0.873	0.2
CD	TUI	1	1	0.423
CD	SWT	0.2	0.873	0.423
CD	UI	0.631	0.873	1
CD	IDE	0.873	0.262	0.873
CD	TUI2	0.423	0.873	0.873
Deb	TUI	1	0.873	0.631
Deb	SWT	0.262	0.873	0.337
Deb	UI	0.337	<b>0.01</b>	0.2
Deb	IDE	0.81	0.337	0.2
Deb	TUI2	0.078	1	0.15
TUI	SWT	0.47	0.873	0.873
TUI	UI	0.423	<b>0.01</b>	0.423
TUI	IDE	1	0.2	0.631
TUI	TUI2	0.336	0.749	0.575
SWT	UI	0.055	<b>0.016</b>	0.575
SWT	IDE	0.262	0.337	0.631
SWT	TUI2	<b>0.037</b>	0.873	0.631
UI	IDE	0.631	<b>0.016</b>	0.522
UI	TUI2	0.521	<b>0.013</b>	0.522
IDE	TUI2	0.521	0.522	1

## References

- Aizawa, A (2003). An information-theoretic perspective of TF-IDF measures. *Information Processing and Management*, 39(1), 45–65.
- Bettenburg, N, S Just, A Schroter, C Weiss, R Premraj and T Zimmermann (2008). What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations Software Engineering*, pp. 308–318. New York: ACM.

- Bhatnagar, V, M Bhardwaj and A Mahabal (2010). Comparing SVM ensembles for imbalanced datasets. In *Proceedings of the 10th International Conference on Intelligent Systems Design and Applications (ISDA)*, Cairo, pp. 651–657. New York: IEEE.
- Canfora, G, A De Lucia, M Di Penta, R Oliveto, A Panichella and S Panichella (2013). Multi-objective cross-project defect prediction. In *IEEE 6th International Conference on Software Testing, Verification and Validation (ICST)*, pp. 252–261. New York: IEEE.
- Chaturvedi, KK and VB Singh (2012). Determining bug severity using machine learning techniques. In *Proceedings of International Conference on Software Engineering*, pp. 378–387. New York: IEEE.
- Chaturvedi, KK and VB Singh (2013). An empirical comparison of machine learning techniques in predicting the bug severity of open and close source projects. *International Journal of Open Source Software and Processes*, 4(2), 32–59.
- Hsu, CW and CJ Lin (2002). A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13, 415–425.
- Joachims, T (1997). Text categorization with support vector machines: Learning with many relevant features. University at Dortmund, LS VIII-Report, *Technical Report*, 23.
- Lamkanfi, A, S Demeyer, E Giger and B Goethals (2010). Predicting the severity of a reported bug. In *7th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 1–10. New York: IEEE.
- Lamkanfi, A, S Demeyer, QD Soetens and T Verdonck (2011). Comparing mining algorithms for predicting the severity of a reported bug. In *CSMR*, Carl von Ossietzky University, Oldenburg, Germany, pp. 249–258. New York: IEEE.
- Ma, Y, G Luo, X Zeng and A Chen (2011). Transfer learning for cross-company software defect prediction. *Information and Software Technology*, 54, 248–256.
- Menzies, T and A Marcus (2008). Automated severity assessment of software defect reports. In *Proceedings of International Conference on Software Maintenance*, pp. 346–355. New York: IEEE.
- Nam, J, SJ Pan and S Kim (2013). Transfer defect learning. In *Proceedings of International Conference on Software Engineering*, pp. 382–391. New York: IEEE.
- Peters, F, T Menzies and A Marcus (2013). Better cross company defect prediction. In *10th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 409–418. New York: IEEE.
- Porter, M (1980). An algorithm for suffix stripping. *Program*, 14(3), 130–137.
- Ryu, D, O Choi and J Baik (2016). Value-cognitive boosting with a support vector machine for cross-project defect prediction. *Empirical Software Engineering*, 21(1), 43–71.
- Sharma, M, P Bedi, KK Chaturvedi and VB Singh (2012). Predicting the priority of a reported bug using machine learning techniques and cross project validation. In *Proceedings of International Conference on Intelligent Systems Design and Applications (ISDA)*, Kochi, India, pp. 539–545. New York: IEEE.
- Sharma, M, P Bedi and VB Singh (2014). An empirical evaluation of cross project priority prediction. *International Journal of Systems Assurance Engineering and Management*, 5(4), 651–663.
- Singh, VB and KK Chaturvedi (2011). Bug tracking and reliability assessment system. *International Journal of Software Engineering and its Applications*, 5(4), 17–30.
- Sun, Y, AC Wong and MS Kamel (2009). Classification of imbalanced data: A review. *International Journal Pattern Recognition*, 23(4), 687–719.
- Tian, Y, D Lo and C Sun (2012). Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In *WCRE*, Kingston, ON, Canada, pp. 215–224. New York: IEEE.

- Tian, Y, D Lo, X Xia and C Sun (2014). Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering*, 20(5), 1354–1383.
- Tian, Y, N Ali, D Lo and AE Hassan (2015). On the unreliability of bug severity data. *Empirical Software Engineering*, 21(6), 2298–2323.
- Turhan, B, A Bener and G Kocak (2009a). Data mining source code for locating software bugs: A case study in telecommunication industry. *Expert Systems with Applications Journal*, 36, 9986–9990.
- Turhan, B, T Menzies, AB Bener and JD Stefano (2009b). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14(5), 540–578, doi: 10.1007/s10664-008-9103-7.
- Vapnik, V (1995). *The Nature of Statistical Learning Theory*. Berlin: Springer-Verlag.
- Vapnik, V, SE Golowich and A Smola (1997). Support vector method for function approximation, regression, and signal processing. In *Advances in Neural Information Processing Systems*, Vol. 9, pp. 281–287. Cambridge, MA: MIT Press.
- Yang, G, T Zhang and B Lee (2014). Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In *IEEE 38th Annual Computer Software and Applications Conference (COMPSAC)*, pp. 97–106. New York: IEEE.
- Yu, L, W Tsai, W Zhao and F Wu (2010). Predicting defect priority based on neural networks. In *Proceedings of 6th International Conference on Advanced Data Mining and Applications*, Wuhan, China, pp. 356–367.
- Zhang, T, G Yang, B Lee and AT Chan (2015). Predicting severity of bug report by mining bug repository with concept profile. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, Salamanca, Spain, pp. 1553–1558. New York: ACM.
- Zhimin, H, S Fengdi, Y Ye, L Mingshu and W Qing (2012). An investigation on the feasibility of cross-project defect prediction. *Autom Software Engineering*, 19, 167–199, Doi:10.1007/s10515-011-0090-3.
- Zimmermann, T, N Nagappan and H Gall (2009). Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In *Proceedings of the 7th Joint Meeting European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Amsterdam, Netherlands, pp. 91–100. New York: ACM.
-