Journal of King Saud University – Computer and Information Sciences 31 (2019) 481–496

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Three-level learning for improving cross-project logging prediction for if-blocks

Sangeeta Lal [a,*], Neetu Sardana [a], Ashish Sureka [b]

[a] Jaypee Institute of Information Technology, Noida, Uttar-Pradesh, India
[b] ABB Corporate Research, Bangalore, India

ARTICLE INFO

Article history:
Received 27 January 2017
Revised 13 June 2017
Accepted 17 July 2017
Available online 7 August 2017

ABSTRACT

Log statements present in the source code provide important execution information to the software developers while debugging. Predicting source code constructs that must be logged is a technically challenging task. Machine learning models are popularly used in literature for logging prediction. These models may not be suitable for small or new projects as these projects do not have sufficient prior data to train the prediction model. For such scenarios, cross-project logging prediction can be useful, in which knowledge gained from the standard project(s) is used to build the model. This paper proposes a three-level model, CLIF, for Cross-project if-blocks logging prediction. CLIF first converts textual features into numeric-textual features(Level-1). CLIF then combines numeric-textual features with Boolean and numeric features(Level-2). Since only few code constructs are logged, CLIF finally learns an effective imbalance decision threshold boundary to predict logged/non-logged code constructs(Level-3). We use 2 text mining classifiers(Level-1) and 8 machine learning classifiers(Level-2) and generate a total of 16 CLIF classifiers. We evaluate performance of CLIF classifiers on three open source projects(six source & target project pair). Experimental results show that CLIF outperforms the existing LogOpt-Plus model and give an improvement of up to 8.21% in average F1-score, and 4.89% in average ROC-AUC.

1. Introduction

Logging is an important software development practice that is performed by inserting log statements in the source code. Log statements are helpful in recording the execution information of the program. Execution information means debugging statements and values of any variables that the developer wants to record in a log file which can be used for diagnosis or troubleshooting. For example, a web server application may record the timestamp, the client IP address and the complete URL including name of the file being requested whenever an exception on file not found is thrown. Not just errors and warnings, a developer may also want to log information that indicate success of an event. This execution information can be used by the software developers at the time of debugging. Logging is important because often this is the only information available to the software developers for debugging. Logging is also useful in several other applications such as anomaly detection (Fu et al., 2009), performance problem diagnosis (Nagaraj et al., 2012), and remote issue resolution (Blackberry enterprise server logs submission, 2016).

Listing 1 shows an example of an if-block containing a log statement. If the expression in the if-blocks condition evaluates to true, the system will record the string in the log statement. Developers can use this information in debugging or in other software development applications. This log optput is found to be useful in fixing the bug 5501 in the Hadoop project (Hadoop, 2016a). Log statements have option to assign verbosity level such as 'error', 'fatal', 'warn'. Verbosity level of a log statement indicates severity of a log statement. Log statements are enabled at software startup using log enable commands such as '–enable-logging –v = 1'. Developers can also insert log statements inside if-blocks which check for log levels i.e., they have expression like 'log.isDebugEnabled()'. The statements inside such if-blocks will be logged only if the log level is set to 'debug' or 'less verbose level'.

Listing 1: Example of a logged if-block and part of the log produced from it (taken from the Hadoop project).

* Corresponding author.
E-mail addresses: sangeeta@jiit.ac.in (S. Lal), neetu.sardana@jiit.ac.in (N. Sardana), ashish.sureka@in.abb.com (A. Sureka).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2017.07.006
1319-1578/

```
if (System.currentTimeMillis() − retrystartTime >= retryInterval) {
      LOG.error("Could not contact RM after " + retryInterval + " milliseconds.");
                  eventHandler.handle(new JobEvent(this.getJob().getID(), JobEventType.INTERNAL_ERROR));
      throw new YarnException("Could not contact RM after " + retryInterval + " milliseconds.");
    }

// output
//2013−09−09 22:26:47,179 ERROR [RMCommunicator Allocator]
//org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: Error communicating with RM: Could not contact RM after
//360000 milliseconds.

//org.apache.hadoop.yarn.exceptions.YarnRuntimeException: Could not contact RM after 360000 milliseconds.

//at org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator.getResources(RMContainerAllocator.java:563)

//at org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator.heartbeat(RMContainerAllocator.java:219)

//at org.apache.hadoop.mapreduce.v2.app.rm.RMCommunicator\$1.run(RMCommunicator.java:236)

//at java.lang.Thread.run(Thread.java:680)
```

Logging is important but it has a tradeoff between cost and benefit. Excessive logging can cause cost and performance overhead, whereas, sparse logging can lessen the benefits of logging by leaving out important debugging information. Ding et al. (2015) and Sigelman et al. (2010) based on experimental studies reveal reveal that in the case of search engines, logging impact on system's performance can be expressed as a 1.48% decrease in throughput or a 16.3% increase of average execution time of the requests. The documentation on Apache Log4j library[1] provides details on the response time, throughput and latency of various types of Log4j variants which demonstrates that logging can have a noticeable impact on the performance depending on the workload. The benchmarking exercise reveals that logging decisions and choices can impact the performance to the extent of 30–100 times. Hence, it is important to optimize the number of log statements in the source code. However, previous research shows that optimizing log statements in the source code is a non-trivial task and software developers often face difficulty in it (Fu et al., 2014). Hence, several recent studies propose machine learning based logging prediction models to help software developers in making strategic logging decision (Fu et al., 2014; Zhu et al., 2015; Lal and Sureka, 2016; Lal et al., 2016b).

Machine learning based prediction requires a sufficient amount of training data to train the prediction model. Xia et al. (2015a) performs an experiment where they compare the performances of machine learning models that were built using different amount of training data. Results of their experiment show that the machine learning model built using large amount of data gives much better prediction performance than that of model built using small amount of data. For example, model built using 90% of the data give F1-score of 65.8% whereas model built using 5% of data give F1-score of 55.4%. However, it is noticed that several real world software projects are new or small. New projects (projects which are in their starting phase) or small projects (projects which have small sizes) do not have sufficient prior data to train the prediction model. In such cases, we can use *cross-project* prediction to train the logging prediction model on previously developed source project(s) to perform logging prediction on target project.

Cross-project prediction is a popular machine learning research area (Dai et al., 2007; Pan et al., 2011) and has also been used in many software engineering applications such as defect prediction (Nam et al., 2013) and build co-change prediction (Xia et al., 2015a). Cross-project prediction is challenging because of two main reasons. First, the source project might not be generalizable to the target project. Second, there is an imbalanced distribution of positive and negative class datapoints in the source code i.e., the *class-imbalance* problem (He and Garcia, 2009). For example, only 8.37%, 8.64%, and 10.6% of if-blocks are logged in the Tomcat, CloudStack, and Hadoop project, respectively. Lal et al. (2017) worked on cross-project catch-blocks logging prediction, however, currently there is no study (to the best of our knowledge) on cross-project if-blocks logging prediction. Hence, cross-project if-blocks logging prediction is a relatively unexplored area.

In this work, we propose a novel three level, machine learning based **C**ross-project **If**-blocks **L**ogging Prediction Model (CLIF). At the first level, CLIF uses a text mining approach to convert textual features to numerical textual features (refer to Section 3 to get details about conversion of textual features to numeric textual features). At the second level, CLIF combines the numeric textual features with numeric and Boolean features and trains a machine learning classifiers. At the third level, CLIF learns an effective decision threshold boundary for if-blocks logging prediction. CLIF searches an effective threshold boundary that maximizes Logged F-measure (LF) (i.e., F1-score and refer to Section 4.3 for details) achieved in the training data to address the class-imbalanced challenge. In this work, we use two text mining classifiers (Naive Bayesian (NB) and Bayesian Network (BN)) to convert textual features to numerical textual features. We use BN and NB text mining classifies because they provide good results in our initial investigation (refer to Section 5.1). CLIF combines these two text mining classifiers with eight machine learning classifiers (Adaboost (ADA), ADTree (ADT), BN, J48, Logistic Regression (LOG), NB, Random Forest (RF), and Support Vector Machine (SVM)) to generate 16 CLIF classifiers i.e., eight $CLIF_{NB}$ and $CLIF_{BN}$ classifiers.

We evaluate CLIF on three large, open-source project i.e., Tomcat (Apache, 2016), CloudStack (Cloudstack, 2016), and Hadoopt (2016b), jointly consisting of 104,043 if-blocks. We consider *LoptOptPlus* model proposed by Lal et al. (2016b) for within-project if-blocks logging prediction on Java projects, as baseline classifier. LogOptPlus classifier was not designed for a cross-project setting and was designed for log statement predictions in a within-project setting. However, in this work we consider LogOptPlus as a baseline classifier as there is no other classifier which has been proposed and evaluated on exactly the same dataset as the one used in this work. Experimental results show that CLIF is effective in improving the performance of cross-project if-block logging prediction. Overall, Several $CLIF_{NB}$ and $CLIF_{BN}$ classifiers outperform the baseline classifier. $CLIF_{BN}^{ADT}$, $CLIF_{NB}^{LOG}$, $CLIF_{BN}^{LOG}$, $CLIF_{NB}^{ADA}$, $CLIF_{BN}^{ADA}$, and $CLIF_{NB}^{ADT}$, give improvement of 8.21%, 8.18%, NB BN NB BN NB

[1] https://logging.apache.org/log4 j/2.x/performance.html.

8.16%, 8.15%, 7.66%, and 7.16%, respectively, in average Logged F-measure (LF), and improvement of 3.85%, 4.15%, 3.34%, 4.89%, 2.23%, and 4.12%, respectively, in average Area Under the ROC Curve (RA). We believe that our work serves as a first step towards effective cross-project if-block logging prediction.

2. Related work and research contribution

In this section, we discuss previous studies closely related to the work presented in this paper, and the novel research contributions of the work presented in this paper.

2.1. Logging prediction

Logging prediction has attracted attention from many researchers, in the software engineering research community (Fu et al., 2014; Zhu et al., 2015; Lal and Sureka, 2016; Lal et al., 2016b,a, 2017). Fu et al. (2014) and Zhu et al. (2015) propose machine learning based models for within-project logging prediction on C# projects for exception types and return value check snippets. C/C#/Java languages, all have many difference in their syntax. For example, C is not an object oriented language. C also doesn't have any concept of try/catch block. In contrast both C# and Java are object oriented languages and have try/catch block to catch exception conditions in the code. Similarly, Java has a concept of 'checked exceptions' but neither C nor C# have checked exceptions. Hence, dedicated studies on each language are required for deep understanding of logging practices in each language. Lal et at. (Lal and Sureka, 2016; Lal et al., 2016b) propose *LogOpt* and *LogOptPlus* models for within-project logging prediction on catch-blocks and if-blocks, respectively, for Java projects. In another work, Lal et al. (2017) propose *ECLogger*, model for cross-project catch-blocks logging prediction. In contrast to these studies, we work on improving cross-project if-blocks logging prediction on Java projects. We use *LogOptPlus* model proposed by Lal et al. (2016b) as a baseline approach, as at present, this is the only model available for if-blocks logging prediction on Java projects. In this work, we target if-blocks because they are one of most frequently logged code snippets (Fu et al., 2014). However, the proposed model, CLIF, can be easily extended to other code snippets.

2.2. Cross-project logging prediction

Cross-project prediction or transfer learning is a popular machine learning research area (Dai et al., 2007; Pan et al., 2011). Cross-project prediction can be unsupervised or supervised (Dai et al., 2007; Pan et al., 2011). In unsupervised prediction, all instances from the target project are unlabeled, while in supervised prediction, only a proportion of the target project instances is labeled. For example, Lal et al. (2017) worked on cross-project prediction for catch-blocks. They did not consider any instance from the target project for model building and hence worked on unsupervised cross-project prediction. Xia et al. (2015a) worked on cross-project build co-change prediction. Xia et al. consider all the instances from the source project and 5% on the instances from the target project to build the cross-project build co-change prediction model and hence, worked on supervised cross-project prediction. Cross-project prediction have been applied to several applications such as defect prediction (Zhang et al., 2015), build co-change prediction (Xia et al., 2015a), sentiment prediction (Pan and Yang, 2010). However, cross-project logging prediction is a relatively unexplored area. Zhu et al. (2015) test effectiveness of their proposed tool, *LogAdvisor*, for cross-project logging prediction and reported 11.9% degradation in model performance for cross-project prediction as compared to within-project prediction. Lal et al. (2017) propose, *ECLogger*, an ensemble based model for cross-project catch-blocks logging prediction. In contrast to Lal et al., our work is novel in several dimensions. First, we work on cross-project if-blocks logging prediction, whereas, Lal et al. work on cross-project catch-blocks logging prediction. Second, ECLogger uses bagging, average vote and majority vote ensemble techniques for cross-project logging prediction. Whereas, CLIF uses three level learning for cross-project prediction. ECLogger does not address the class-imbalance problem. Whereas, CLIF learns a suitable threshold boundary for improving cross-project prediction on imbalanced dataset.

2.3. Learning on imbalanced data

The *class-imbalance* is a well known problem in the area of machine learning research. It occurs when count of negative class instances is considerably higher than the count of positive class instances. Older machine learning algorithms were designed to work on balanced data and, hence, often give poor performance in case of imbalanced data. Several approaches have been proposed in the literature to address the class-imbalance problem (Chawla et al., 2002, 2003; Kubat et al., 1997; Guo and Viktor, 2004). Data sampling is the primary approach to address the class-imbalance problem. Previous studies on logging prediction either use majority class under-sampling (Fu et al., 2014; Lal and Sureka, 2016; Lal et al., 2016b) or minority class over-sampling (Zhu et al., 2015) to address the class-imbalance problem. However, identifying a suitable sampling technique is a non-trivial task. Evaluation metrics based approaches proposed new evaluation metrics for performance evaluation on imbalanced data. F1-score and area under the ROC curve are two popular and widely used evaluation metrics for imbalanced dataset (Xia et al., 2016; Xia et al., 2015a; He and Garcia, 2009). Hence, we use F1-score and area under the roc curve as evaluation metric for comparing the performances of different models. Threshold adjustment based approaches work on learning a better threshold, as the default decision threshold of 0.5 may not be suitable in case of imbalanced dataset. Xia et al. (2015b) propose a threshold learning method for blocking bug prediction on imbalanced dataset. We use the approach proposed by Xia et al. for threshold learning in cross-project if-blocks logging prediction, which to the best of our knowledge has never been used in this context.

2.4. Research contribution

In terms of related work, the study presented in this paper makes the following novel and unique research contributions.

1. We propose, *CLIF*, a novel, three level learning based model for cross-project if-block logging prediction on Java projects. We use two textual feature processing classifiers and eight machine learning classifiers. We create eight $CLIF_{NB}$ and eight $CLIF_{BN}$ classifiers i.e., a total of 16 CLIF classifiers (refer to Section 3).
2. We present results of comprehensive evaluation of the proposed model on three large open-source Java based projects i.e., Tomcat (2016), Cloudstack (2016) and Hadoopt (2016b). We use two metrics LF and Area under the ROC curve for comparing the prediction performances of CLIF and the baseline (LogOptPlus) classifier. The results demonstrates that CLIF is effective in improving the cross-project if-blocks logging prediction performance (refer to Section 5).

3. CLIF model creation

Fig. 1 presents the overview of the proposed CLIF framework. CLIF consists of two main phases: model building phase and

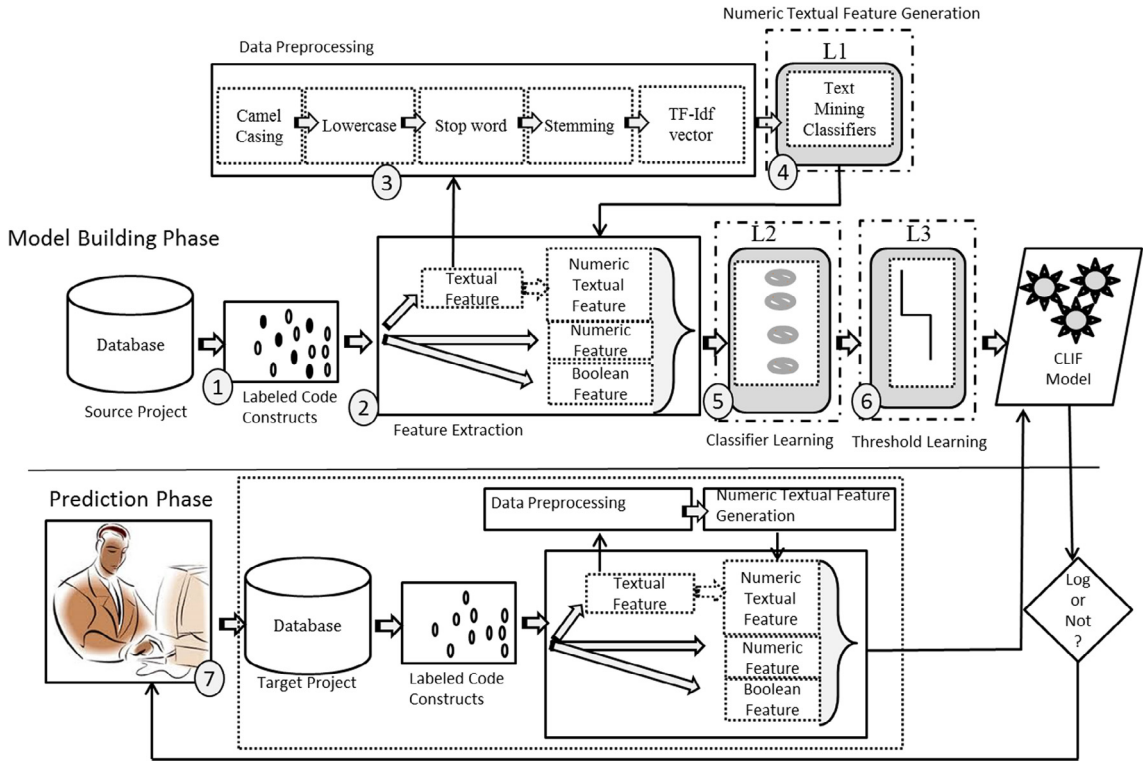


Fig. 1. Overview of the Proposed CLIF Framework, L1: Level 1, L2: Level 2, L3: Level 3.

Table 1
Notations used in the CLIF algorithm (algorithm 1) and threshold estimation algorithm (algorithm 2).

Notation	Meaning	Notation	Meaning
$\mathcal{P}$	Project	$\mathcal{P}_X$	Project X, where X ∈ Tomcat (T), CloudStack (C), and Hadoop (H)
$\mathcal{SP}$	Source project	$\mathcal{A}$	Algorithms
$\mathcal{TP}$	Target project	$\mathcal{A}_X$	Algorithm X, where X ∈ {ADA, ADT, BN, J48, LOG, NB, RF, SVM}
$\mathcal{TH}$	Decision threshold	$\mathcal{TEXT} - \mathcal{ALGO}$	Text mining classifier
$\mathcal{IB}$	If-blocks	$\mathcal{IB}_X$	If-blocks of type X, where X ∈ {$\mathcal{SP}$, $\mathcal{TP}$}
$\mathcal{FV}$	Feature vector	$\hat{\mathcal{FV}}$	Feature vector obtained after preprocessing textual features
$\hat{\mathcal{FV}}$	Feature vector obtained after filtering undesired features	${}_Z\mathcal{FV}_X^Y$	Feature vector for phase X of domain Y and type Z, where X ∈ {$\mathcal{SP}$, $\mathcal{TP}$},Y ∈ {Initial (I), Final (F)} and Z ∈ {Textual (T), Numerical (N), Boolean (B), Numeric Textual Feature (NT)}
$\mathcal{PD}$	Prediction results	${}_{\mathcal{TH}}\text{CLIF}_{\mathcal{ML}}^{\mathcal{A}}$	CLIFclassifierusingtext mining classifier $\mathcal{ML}$, algorithm $\mathcal{A}$, and with threshold value $\mathcal{TH}$
$\mathcal{RTD}$	Randomized training data	i, j, $\mathcal{ML}$, α	Temporary variables

prediction phase (refer to Fig. 1). In the model building phase, we build a cross-project if-blocks logging prediction model from the labeled instances of the source project (step 1-step 6). In the prediction phase, CLIF classifiers learned in the model building phase are used to predict label of new instances (step 7) in the target project (see Table 1).

3.1. Phase 1: (Model Building Phase)

3.1.1. Training Instances Collection (Step 1)

Our experimental dataset consists of three projects: Tomcat, CloudStack and Hadoop. We consider one project as the source project (SP), i.e., training project, and the other two projects as the target project (TP), i.e., testing project, a single project at a particular instance. Using this, we create six source and target project pairs (refer to lines 6–10 in Algorithm 1). CLIF extracts all logged and non-logged if-blocks ($\mathcal{IB}_{SP}$) from the source project for training.

3.1.2. Feature Extraction (Step 2)

Lal et al. (2016b) propose 28 features for if-block logging prediction on Java based projects. Lal et al. categorize features in terms of *domains* (if, method_bi, and other), *types* (boolean, numerical, and textual), and *classes* (positive and negative). *Domain* specifies part of the source code from where the feature is extracted. If the feature is extracted from if-block then it will have domain 'if', if the feature is extracted from the first line of the containing function (the function that consists of the target if-block) to the previous line of the target if-block it will have domain 'method_bi'. If the feature is extracted from some other part of the source code it will have domain 'other'. For example, the feature 'Logged Method_bi [LM]' (which checks whether the method_bi region has any log statement or not) has domain 'method_bi'. *Type* of a feature specifies whether a feature is 'textual', 'Boolean' or 'numeric'. Boolean features can take value '0' or '1'. Numeric features can take any real (+ve) value. Textual features can take any 'text' value. For example, 'If-expression [EI]' is a textual feature that extracts 'checking condition' of the target if-block. The *class* of a feature can be 'positive' or 'negative'. Positive class features are beneficial in predicting logged if-blocks and negative class features are beneficial in predicting non-logged if-blocks. For example, 'Null Condition (NC)' (checks whether the expression of if-block checks any null value), is a positive class feature. We use all the 28 features

Table 2
Features used for cross-project if-blocks logging prediction. These features are taken from the previously published work by by Lal et al. [23]. P,I: class = positive, domain = if; P,M: class = positive, domain = method_bi; P,O: class = positive, domain = other; N,I: class = negative, domain = if; and N,M: class = negative, domain = method_bi.

Feature Type	S. No	Features	Feature Description
Boolean	1.	Logged Method_BI [LM] (P,M)	Presence/absence of log statement in method_bi section
	2.	Method have Parameter [PM](P,O)	Containing method have parameters
	3.	IF in Method_BI [IM] (P, M)	Presence/absence of if-statement in method_bi section
	4.	Null Condition [NC] (P,I)	The condition of if-block checks for 'Null' value
	5.	InstanceOf Condition [IOC](P,I)	The condition of if-block checks for 'InstanceOf'
	6.	Throw/Throws in IF Block [TTI] (N,I)	Presence/absence of Throw/ Throws in if-block
	7.	Throw/Throws in Method_BI [TTM]	Presence/absence of Throw/ Throws in method_bi section
	8.	Return in IF Block [RI] (N, I)	Presence/absence of 'return' statement in if-block
	9.	Return in Method_BI [RM] (N, M)	Presence/absence of 'return' statement in method_bi section
	10.	Assert in Method_BI [AM] (N, M)	Presence/absence of 'assert' statement in method_bi section
	11.	Assert in IF Block [AI] (N, I)	Presence/absence of 'assert' statement in if-block
Numerical	1.	Size of Method_BI [SM] (P, M)	SLOC of method_bi section.
	2.	Log Count in Method_BI [LCM] (P, M)	Number of log statements in method_bi section.
	3.	Count of Operators in Method_BI [COM] (P, M)	Number of arithmetic operators in method_bi section
	4.	Variable Declaration Count in Method_BI [VCM](P,M)	Number of variable declared in method_bi section
	5.	Method Parameter Count [PCM] (P,O)	Number of parameters in the containing function
	6.	Method Call Count in Method_BI [MCM] (P,M)	Number of methods called in method_bi section.
	7.	IF Count in Method_BI [ICM] (P,I)	Number of if-statements in method_bi section
Textual	1.	Log Levels in Method_BI [LLM] (P,I)	Log levels of the log statements present in the method_bi section
	2.	Operators in Method_BI [OM] (P,M)	Arithmetic operators used in the method_bi section
	3.	Variable Declaration Name in Method_BI [VNM] (P,M)	Name of variables declared in the method_bi section
	4.	Method Call Name in Method_BI [MNM] (P,M)	Names of the methods called in the method_bi section
	5.	Method Parameters (type) [PTM] (P,O)	Type of the parameters called in the containing function
	6.	Method Parameters (name) [PNM](P,O)	Names of the parameters called in the containing function
	7.	Container Package Name [CPN] (P,O)	Name of the package that consists of the if-block
	8.	Container Class Name [CCN] (P,O)	Name of the class that consists of the if-block
	9.	Container Method Name [CMN] (P,O)	Name of the method that consists of the if-block
	10.	IF Expression [EI] (P,I)	Checking Condition (Expression) of the if-block

proposed by Lal et al. (refer to line 11–13 in Algorithm 1). Table 2 presents listing of all the 28 features, and their respective *domain*, *class* and *type*.

Algorithm 1 CLIF Algorithm

1: **procedure** CLIF
2: $\mathcal{P} = \{\mathcal{P}_T, \mathcal{P}_C, \mathcal{P}_H\}$
3: $\mathcal{ALGO} = \{\mathcal{A}_{ADA}, \mathcal{A}_{ADT}, \mathcal{A}_{BN}, \mathcal{A}_{J48}, \mathcal{A}_{LR}, \mathcal{A}_{NB}, \mathcal{A}_{RF}, \mathcal{A}_{SVM}\}$
4: $\mathcal{TEXT} - \mathcal{ALGO} = \{\mathcal{A}_{NB}, \mathcal{A}_{BN}\}$
5: **for** all $\mathcal{ML} \in \mathcal{TEXT} - \mathcal{ALGO}$ **do**
6: **for** all S $\in \mathcal{P}$ **do**
7: **for** all T $\in \mathcal{P}$ **do**
8: **if** S$\neq\mathcal{T}$ **then**
9: $\mathcal{SP}$ = S, $\mathcal{TP}$ = T
10: $\mathcal{IB}_{\mathcal{SP}}$ = **ReadCompleteData** ($\mathcal{SP}$)
11: ${}_T\mathcal{FV}^{\mathcal{I}}_{\mathcal{SP}}$ = **ExtractTextualFeatures** ($\mathcal{IB}_{\mathcal{SP}}$)
12: ${}_N\mathcal{FV}^{\mathcal{F}}_{\mathcal{SP}}$ = **ExtractNumericalFeatures** ($\mathcal{IB}_{\mathcal{SP}}$)
13: ${}_B\mathcal{FV}^{\mathcal{F}}_{\mathcal{SP}}$ = **ExtractBooleanlFeatures** ($\mathcal{IB}_{\mathcal{SP}}$)
14: ${}_T\mathcal{FV}^{\mathcal{F}}_{\mathcal{SP}}$ = **Preprocess**(${}_T\mathcal{FV}^{\mathcal{I}}_{\mathcal{SP}}$)
15: ${}_NT\mathcal{FV}^{\mathcal{F}}_{\mathcal{SP}}$ = **GenerateNumericalTextualFeature** (${}_T\mathcal{FV}^{\mathcal{F}}_{\mathcal{SP}}, \mathcal{ML}$)
16: $\mathcal{FV}^{\mathcal{F}}_{\mathcal{SP}}$ = $\{{}_{NT}\mathcal{FV}^{\mathcal{F}}_{\mathcal{SP}}, {}_N\mathcal{FV}^{\mathcal{F}}_{\mathcal{SP}}, {}_B\mathcal{FV}^{\mathcal{F}}_{\mathcal{SP}}\}$
17: **for** all $\mathcal{A} \in \mathcal{ALGO}$ **do**
18: CLIF$^{\mathcal{A}}_{\mathcal{ML}}$ = **BuildModel** ($\mathcal{FV}^{\mathcal{I}}_{\mathcal{SP}}, \mathcal{A}$)
19: ${}_{TH}$CLIF$^{\mathcal{A}}_{\mathcal{ML}}$ = **ThresholdEstimation** (CLIF$^{\mathcal{A}}_{\mathcal{ML}}$)
20: ${}_T\mathcal{FV}^{\mathcal{I}}_{\mathcal{TP}}$ = **ExtractTextualFeatures** ($\mathcal{IB}_{\mathcal{TP}}$)
21: ${}_N\mathcal{FV}^{\mathcal{F}}_{\mathcal{TP}}$ = **ExtractNumericalFeatures** ($\mathcal{IB}_{\mathcal{TP}}$)
22: ${}_B\mathcal{FV}^{\mathcal{F}}_{\mathcal{TP}}$ = **ExtractBooleanlFeatures** ($\mathcal{IB}_{\mathcal{TP}}$)
23: ${}_T\mathcal{FV}^{\mathcal{F}}_{\mathcal{TP}}$ = **Preprocess** (${}_T\mathcal{FV}^{\mathcal{I}}_{\mathcal{TP}}$)
24: ${}_N{}^T\mathcal{FV}^{\mathcal{F}}_{\mathcal{TP}}$ = **GenerateNumericalTextualFeature** (${}_T\mathcal{FV}^{\mathcal{F}}_{\mathcal{SP}}, {}_T\mathcal{FV}^{\mathcal{F}}_{\mathcal{TP}}, \mathcal{ML}$
25: $\mathcal{FV}^{\mathcal{F}}_{\mathcal{TP}} = \{{}_{NT}\mathcal{FV}^{\mathcal{F}}_{\mathcal{TP}}, {}_N\mathcal{FV}^{\mathcal{F}}_{\mathcal{TP}}, {}_B\mathcal{FV}^{\mathcal{F}}_{\mathcal{TP}}\}$
26: $\mathcal{PD}$=**ApplyModel**(${}^{\mathcal{TH}}$CLIF$^{\mathcal{A}}_{\mathcal{ML}}, \mathcal{FV}^{\mathcal{F}}_{\mathcal{TP}}$)
27:
28: **procedure** ReadCompleteData(P)
29: $\mathcal{IB}$ = **ReadIfBlocks** (P)
30: **return** $\mathcal{IB}$
31:
32: **procedure** ExtractTextualFeatures$\mathcal{IB}$
33: ${}^T\mathcal{FV}$ = **getTextualFeatures** $\mathcal{IB}$)
34: **return** ${}^T\mathcal{FV}$
35:
36: **procedure** ExtractBooleanFeatures ($\mathcal{IB}$)
37: ${}^B\mathcal{FV}$ = **getBooleanFeatures** ($\mathcal{IB}$)
38: **return** ${}^B\mathcal{FV}$
39:
40: **procedure** ExtractNumericalFeatures($\mathcal{IB}$)
41: ${}_N\mathcal{FV}$ = **getNumericalFeatures** ($\mathcal{IB}$)
42: **return** ${}_N\mathcal{FV}$
43:
44: **procedure** Preprocess($\mathcal{FV}$)
45: ${}_T\hat{\mathcal{F}}\mathcal{V}$ = **TF_IDFConversion**(**Stemming** (**StopWordRemoval**(**LowerCase** (**CamelCaseSeparation** (${}_T\bar{\mathcal{F}}\mathcal{V}$)))))
46: **if** It is Test Data **then**
47: ${}^T\tilde{\mathcal{F}}\mathcal{V}$ = **FilterFeatureNotTrainData** (
48:${}_T\hat{\mathcal{F}}\mathcal{V}$)
else
${}_T\tilde{\mathcal{F}}\mathcal{V} = {}_T\hat{\mathcal{F}}\mathcal{V}$
return ${}_T\tilde{\mathcal{F}}\mathcal{V}$

$$TF - IDF_{\{t,d\}} = TF_{\{t,d\}} \times log\frac{N}{DF_t} \quad (1)$$

3.1.3. Pre-processing (Step 3)

We apply five feature pre-processing techniques on ini-tial textual features ($_T\mathcal{FV}^{I}_{SP}$) to obtain final textual feature vectors ($_T\mathcal{FV}^{F}_{SP}$) . We ap-ply camel case conversion, lower case conversion, stop word removal, stemming and tf-idf conversion (Manning et al., 2008). We first, separate all the terms joined using camel casing. For example, 'LoginFailure' is converted to 'Login' and 'Failure'. Second, we convert all the terms to lowercase. For example, the terms 'Login' is converted to 'login'. Using both 'Login' and 'login' as separate features can result in large amount of redundant features. Third, we remove all the English stop words such as 'the', 'is' from the text. Stop words are frequently occurring terms in the document. Since, they occur in most of the documents they are considered as non-content bearing. Fourth, we apply stemming and convert all the terms to their root form. Stemming is the process of reducing inflected words to their root form. For example, the terms 'modifier' and 'modify' will be converted to root word 'modifi'. Conversion of inflected forms results in lowering down the dimensionality and hence, is beneficial in reducing the time and space complexity of the feature space. Fifth, we convert all the terms to their Tf-Idf (term frequency-inverse document frequency) representation. Tf-idf is a numerical statistic that is used to identify importance of a word in a given document in a collection of corpus (Manning et al., 2008). Eq. (1) presents the formula for Tf-idf weight computation. In Eq. (1), Term Frequency ($TF_{t,d}$) presents frequency of a word t in a document d, N is the total number of documents in the corpus and Document Frequency (DF_t) represents frequency of a term t in the corpus. Hence, tf-idf value increases for frequent words in a document but it decreases as the frequency of word increases in the corpus. Using these pre-processing steps we obtain our final features ($_T\mathcal{FV}^{F}_{SP}$) (refer to line 11, 14, 44–51 in Algorithm 1). These pre-processing steps have been used in other logging prediction studies as well (Zhu et al., 2015; Lal and Sureka, 2016; Lal et al., 2016b). Fig. 2 shows an illustrative example of all the pre-proessing steps applied in this work.

3.1.4. Numeric Textual Feature Generation (Step 4)

Tf-idf conversion performed in the previous step, can lead to thousands of features, since they derive from all the terms in a document corpus representing the code blocks. As a result, combining all these textual features directly with numerical and Boolean features may dilute the weight of the numerical and Boolean features. We thus need a way to normalize the representation of each class or type of predictor to ensure that no feature class biases the outcome or decreases the representation of other classes. Conversion of textual feature or consolidating the textual feature into a numeric feature is one way towards dimensionality reduction of one class of feature with respect to other class of features. Dimensionality reduction of features to remove irrelevant features or identifying principal components is a well-known technique in machine learning (Guyon and Elisseeff, 2003; Liu and Yu, 2005; Yu and Liu, 2004). Hence, in this work we explore the possibility of converting all the textual features to a single feature i.e., numeric textual feature (refer to Level 1 or step 4 in Fig. 1). In this manner, we can combine the power of textual features without diluting the effects of boolean and numeric features. Conversion of textual features to numeric was used by Valdivia Garcia and Shihab (2014) for blocking bugs prediction and by Xia et al. (2016) for closed question prediction. However, its uses and effectiveness with respect to cross-project if-blocks logging prediction is unexplored yet.

Fig. 3a shows the steps of conversion of textual features to numerical textual features for the model building phase. In the model building phase, we first divide the training dataset into two equal parts using stratified random sampling. The main advantage with stratified sampling is that it captures key population characteristics in the sample. This method of sampling produces characteristics in the sample that are proportional to the overall population. Stratified sampling have also been used by other researches in software engineering research community (Xia et al., 2016). We have used stratified sampling to divide our training dataset into two parts such that the distribution of logged and

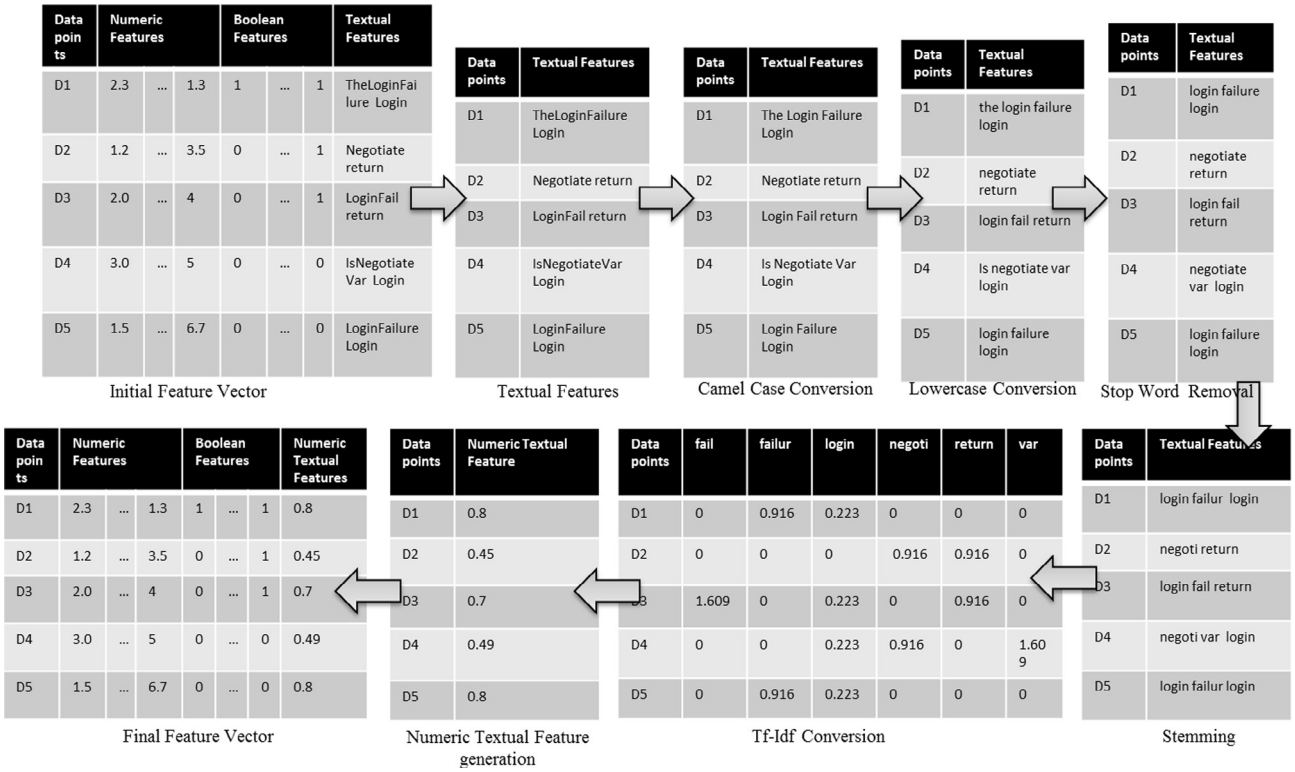
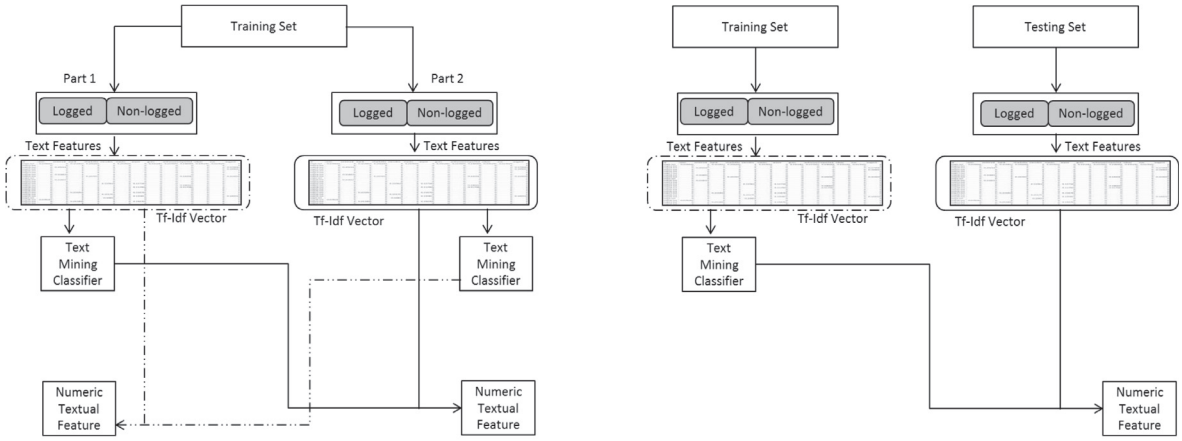


Fig. 2. Conversion of Textual Features to Numeric Textual Features.



(a) Training Data (b) Testing Data

Fig. 3. Conversion from textual to numeric textual features.

non-logged if-blocks is same in both the samples. We train a text mining classifier on the first sample (using tf-idf representation of textual features), and use it to assign a confidence score (i.e., likelihood of an if-block to be logged) to each if-block in the second sample. Similarly, we also train a text mining classifier on the second sample (using tf-idf representation of textual features) and use it to assign a confidence score on the first sample. These confidence score are termed as numerical textual features ($_{NT}\mathcal{FV}^{F}_{SP}$) (refer to line 15 in Algorithm 1 and Fig. 3a).

These numeric textual features are then combined with Boolean and numeric features to obtain the final feature (FV^{F}_{SP}). Fig. 2 shows an illustrative example of conversion of textual to numeric textual features. In this work, we use two text min-ing classifiers i.e., NB and BN (refer to line 4 in Algorithm 1). We select NB and BN because these two classifiers give the best result for cross-project prediction in our initial investigation (refer to Section 5.1).

Algorithm 2 Threshold Estimation

1: Input: Train Data ($\mathcal{FV}^{F}_{SP}$), Machine Learning Algorithm($\mathcal{A}$), Text Mining Classifier ($\mathcal{ML}$)
2: Output: Threshold value ($\mathcal{TH}$)
3: **procedure** ThresEstimation (Train)
4: **for** i=1, 1≤10 , i=i+1 **do**
5: $\mathcal{RTD}$ = **Randomize**($\mathcal{FV}^{F}_{SP}$ i)
6: Divide $\mathcal{RTD}$ into two parts T1 and T2. T1 consists of 80% of the instances and T2 consists of remaining 20%.
7: $CLIF^{\mathcal{A}}_{\mathcal{ML}}$ =**BuildClassifier**(T1, $\mathcal{A}, \mathcal{ML}$)
8: score[i] = **ApplyModel**($CLIF^{\mathcal{A}}_{\mathcal{ML}}$, T2)
9: **for** i =1, 1≤10 , i=i+1 **do**
10: **for** all α =0, α≤1, α = α+0.01 **do**
11: Fmeasure[i][α]=**ComputeFmeasure**(score[i], α)
12: **for** all α =0, α 1, α = α+0.01 **do**
13: AvgFmeasure[α] = $\frac{\sum_{i=1}^{10} Fmeasure[i][\alpha]}{10}$
14: **return** α which maximizes the average LF (i.e., AvgFmeasure) for if-blocks in T2
15:

$$Predict(if-block_i) = \begin{cases} Logged, & if\ Score_{log}(if-block) \geqslant TH \\ Non-logged, & otherwise \end{cases} \quad (2)$$

3.1.5. CLIF Classifier Learning (Step 5)

We combine numerical, boolean and numerical textual features to obtain the final feature vector ($\mathcal{FV}^{F}_{ScalP}$) (line to 16 in Algorithm 1). Using this we obtain total 19 features i.e., 11 boolean, 7 numeric, and 1 numeric textual feature. We then train eight main machine learning classifiers (ADA, ADT, BN, NB, LR, NB, RF, SVM) to obtain our initial CLIF classifiers (line 8 in Algorithm 1). For example, $CLIF^{ADA}_{NB}$ is created using NB as text mining classifier and Adaboost as a main classifier. Using this approach we create sixteen CLIF classifiers i.e., $CLIF^{ADA}_{NB}$ $CLIF^{ADT}_{NB}$ $CLIF^{BN}_{NB}$, $CLIF^{J48}_{NB}$, $CLIF^{LR}_{NB}$ $CLIF^{NB}_{NB}$, $CLIF^{RF}_{NB}$ $CLIF^{SVM}_{NB}$, $CLIF^{ADA}_{BN}$, $CLIF^{ADT}_{BN}$, $CLIF^{BN}_{BN}$, $CLIF^{J48}_{BN}$, $CLIF^{LR}_{BN}$ $CLIF^{NB}_{BN}$, $CLIF^{RF}_{BN}$, $CLIF^{SVM}_{BN}$ (refer to Level 2 or step 5 in Fig. 1).

3.1.6 Threshold Learning (Step 6)

At the time of logging prediction on new instances these classifiers will output a confidence score (refer to $Score_{log}$ in Eq. (2)). If the value of this confidence score is greater than threshold ($\mathcal{TH}$) the given instance will be predicted as logged, otherwise it is predicted as non-logged (refer to Eq. (2)). Default value of threshold is 0.5, which may not be appropriate in case of imbalanced dataset. Hence, we use threshold learning to find the best threshold for cross-project if-blocks prediction (refer to Level 3 or step 6 in Fig. 1). We use the threshold learning approach proposed by Xia et al. (2016) for deleted question prediction. Algorithm 2, presents the pseudo code of the threshold learning approach. It randomly divides the training set into part T1 and T2 using random sampling such that distribution of logged and non-logged instances is same in both T1 and T2. T1 consists of 80% of the instances and T2 constitutes of the remaining 20% instances. Next, we build a classifier using T1 and output a confidence score for each if-block in T2. Finally, to tune the threshold value we gradually increase the threshold value from 0 to 1 using a step size of 0.01. We compute LF for each threshold value. Since, random sampling can have biases we create 10 such random samples and output the confidence score that maximizes the average LF on these 10 random samples. For example, for $CLIF^{ADA}_{NB}$ classifier the threshold value of 0.18 is used. Hence, $CLIF^{ADA}_{NB}$ will predict an instance NB as 'logged' if the confidence score for that instance is greater than 0.18 otherwise it will be predicted as 'non-logged' (refer to Table 5 in the paper). Using this approach we obtain our CLIF prediction model ($(_{TH}CLIF^{\mathcal{A}}_{\mathcal{ML}})$) (refer to line 19 in Algorithm 1).

3.2. Phase 2: (Prediction Phase)

3.2.1. Prediction (Step 7)

In the prediction phase, CLIF classifiers are used to predict labels of new instances ($\mathcal{IB}_{TP}$) in the target project. We extract all the 28 features from the if-blocks present in the testing set. We extract all the three types of features (textual, Boolean, and numeric) from the datatset. We apply all the pre-processing steps (camel case conversion, lower case conversion, stop word removal, stemming and tf-idf) describe in step 3 on initial textual features ($_T\mathcal{FV}^I_{TP}$). In addition to all these pre-processing steps, we apply one more filtering step. In case of cross-project pre-diction, it is possible that some textual features present in the training set are absent in the testing set. We remove all such textual features from the test set and obtain our final textual features ($_T\mathcal{FV}^F_{TP}$) (line 46–48 in Algorithm 1). Next, we convert finalTP textual features to numeric textual features ($_{NT}\mathcal{FV}^F_{TP}$). Fig. 3b shows the steps for numeric textual feature generation for the test dataset. We learn a text mining classifier using final textual features from the training dataset. The classifier is then used to predict confidence scores of the test dataset. These confidence scores are then used as numeric textual features for test dataset (refer to line 24 in Algorithm 1). We obtain our final feature vector ($\mathcal{FV}^F_{TP}$)) by combining Boolean, numeric and numeric textual features. We then apply CLIF classifier for logging prediction on the instances present in the target project. The instance is predicted as 'logged', if the CLIF classifier gives the confidence score greater to the threshold values that is learned in the training phase (refer to step 5). Otherwise, it is predicted as 'non-logged'.

4. Experimental details

In this section, we present details related to the experiment performed in this work. We describe our experimental dataset, design of the experiment, and the evaluation metrics.

4.1. Details of Experimental dataset

We perform all experiments on three large open source projects: Tomcat, CloudStack and Hadoop. All three projects are large, long lived, and actively maintained Java based projects from Apache Software Foundation (ASF[2]). Table 3 presents the experimental dataset statistics. We extract all the if-blocks from all the three projects using Eclipse AST[3] source code parsing libraries. We mark all the if-blocks as 'logged' or 'non-logged' based on presence or absence of log statements in them. An if-block is marked as 'logged' if it consists of at least one log statement otherwise it is marked as 'non-logged'. Table 3 shows that 8.37%, 8.64% and 10.60% of if-blocks are logged in Tomcat, CloudStack and Hadoop project, respectively.

4.2. Design of the experiment

To conduct the cross-project logging prediction experiment, we create training and testing datasets. We use all the if-blocks from the source projects for training and all the if-blocks from the target project for testing. We use the default WEKA (Hall et al., Nov. 2009) implementation of all the machine learning classifiers.

[2] http://www.apache.org/.

[3] https://www.eclipse.org/articles/article.php?file=Article-JavaCodeManipulation_AST/index.html.

Table 3
Experimental dataset statistics.

	Tomcat	CloudStack	Hadoop
Version	8.0.9	4.3.0	2.7.1
SLOC	273419	849832	915659
Number of Java Files	2036	5350	6331
Total If-Blocks	16991	65392	32143
Logged If-Blocks	1423(8.37%)	5653 (8.64%)	3407 (10.60%)

4.3. Evaluation metrics

In this subsection, we describe the metrics used to evaluate the performance of the proposed model. We use five metrics: precision, recall, F1-score, accuracy, Are Under the ROC curve (RA). All the five metrics are widely used and have been used in previous studies (Kim et al., 2008; Lal et al., 2016b; Lal et al., 2017; Zhang et al., 2015). At the time of prediction classifier outputs a decimal score between 0 to 1. If the value of this score is above our learned decision threshold the corresponding if-block will be predicted as 'logged' otherwise it will be predicted as 'non-logged'. For example, for $CLIF^{ADA}_{NB}$ the value of threshold is 0.18 (refer to Table 5). If $CLIF^{ADA}_{NB}$ outputs a score of 0.19 for an if-block, the corresponding if-block will be predicted as 'logged'.

For each if-block we consider log statement inserted by software developers as ground truth or gold standard. For each if-block we compare the output of CLIF with ground truth. If the if-block does not contain a log statement in the ground-truth and CLIF predicts that a log statement should be inserted then we count it is a false positive case. This is an error or mis-classification by the classifier. Similarly, when an if-block in the ground-truth contains a log statement and CLIF estimates that a log statement is not required then it is a case of false negative and is the other type of mis-classification. True positive and true negatives are correct classification and occur when the actual and predicted values match. In total, at the time of prediction following four outcomes are possible:

True Positive: Predicting a logged code constructs as logged ($l \rightarrow l$).

True Negative: Predicting a non-logged code constructs as non-logged ($n \rightarrow n$).

False Positive: Predicting a non-logged code construct as logged ($n \rightarrow l$).

False Negative: Predicting a logged code construct as non-logged ($l \rightarrow n$).

We build our model using training data and perform prediction on testing data. We count the total number of true positive ($N_{l\rightarrow l}$), false negative ($N_{l\rightarrow n}$), true negative ($N_{n\rightarrow n}$), and false positive ($N_{n\rightarrow l}$) in the testing dataset. Using these four values we define following metrics:

Logged precision: It is the percentage of correctly labeled logged code constructs among those labeled as logged.

$$Logged\ Precision(LP) = \frac{N_{l\rightarrow l}}{N_{l\rightarrow l} + N_{n\rightarrow l}} \times 100 \quad (3)$$

Logged recall: It is the proportion of logged code constructs that are correctly labeled as logged.

$$Logged\ Recall(LR) = \frac{N_{l\rightarrow l}}{N_{l\rightarrow l} + N_{l\rightarrow n}} \times 100 \quad (4)$$

Logged F-measure: Precision and recall metrics have a tradeoff i.e., one can increase precision (recall) by decreasing recall (precision) (Han et al., 2011; Manning et al., 2008). Logged F-measure metric combines the logged precision and recall metric and hence is useful in overcoming the precision and recall tradeoff. It has been widely used in the software engineering literature for performance evaluation (Zhu et al., 2015; Valdivia Garcia and Shihab, 2014).

$$Logged\ F-measure(\mathrm{LF}) = \frac{2 \times LP \times LR}{LP + LR} \times 100 \quad (5)$$

Accuracy: It is the percentage of correctly labeled code constructs to all the code constructs.

$$Accuracy = \frac{N_{l\to l} + N_{n\to n}}{N_{|\to l} + N_{l\to n} + N_{n\to n} + N_{n\to l}} \times 100 \quad (6)$$

Area Under the ROC Curve (RA): RA measures the likelihood that a classifier will give higher probability to a 'logged' code construct as compared to a 'non-logged' code construct. Value of RA can range between 0 and 1. A classifier with higher RA value is considered better as compared to a classifier with lower RA value.

5. Experimental results

In this section, we address the five identified research questions (RQs) by conducting experiments on three open-source projects. The following subsections elaborate the motivation, approach, and results obtained for each RQs.

5.1. RQ 1: What is performance of baseline classifier for cross-project if-blocks logging prediction?

Motivation: In RQ 1, we evaluate effectiveness of the LogOptPlus model proposed by Lal et al. (2016b) for cross-project if-blocks logging prediction. We perform comprehensive evaluation of the LogOptPlus model with several machine learning classifiers, as this model has never been evaluated for cross-project if-blocks logging prediction.

Approach: We evaluate effectiveness of the LogOptPlus model for all the six source and target project pairs. In addition to RF, we use eight other machine learning classifiers i.e., ADA, ADT, BN, J48, LOG, NB, RF, RBF, SVM.

Results: Table 4 presents the results of cross-project if-blocks logging prediction using the LogOptPlus model. Table 4 presents that the $LogOptPlus_{BN}$ and $LogOptPlus_{NB}$ classifiers give better performances as compared to the other LogOptPlus classifiers. $LogOptPlus_{BN}$ gives the highest LF for four source and target projects pairs. $LogOptPlus_{NB}$ classifier give the second highest LF for four source and target projects. $LogOptPlus_{BN}$ and $LogOptPlus_{NB}$ give the highest LF of 48.15% and 44.78%, respectively, for Hadoop → CloudSta project pair. LogOptPlus classifier give the highest LF for two source and target project pairs. It give LF of 34.98% and 38.58% for CloudStack → Tomcat and CloudStack → Hadoop project pairs, respectively. However, it gives 0% LF for other four source and target projects. $LogOptPlus_{RBF}$ classifier performs the worst among all the LogOptPlus classifiers and give the 0% LF for all the six source and target project pairs. Hence, we remove RBF classifier from further analysis

> $LogOptPlus_{BN}$ and $LogOptPlus_{NB}$ classifiers performs the best for crossproject if blocks logging prediction.

Table 4
Cross project if-blocks logging prediction results by LogOptPlus model [23].

	Project: CloudStack → Tomcat Source Instances:65392,Target Instances: 16991 Features: 1246					Project: Hadoop → Tomcat Source Instances: 32143,Target Instances: 16991 Features: 1232				
Classifier	LP (%)	LR (%)	LF (%)	ACC (%)	RA (%)	LP (%)	LR (%)	LF (%)	ACC (%)	RA (%)
$LogOptPlus_{ADA}$	29.22	43.57	34.98	86.43	67.15	0	0	0	91.62	67.48
$LogOptPlus_{ADT}$	39.37	15.74	22.49	90.91	67.57	0	0	0	91.62	71.84
$LogOptPlus_{BN}$	12.35	65.35	20.77	58.24	66.92	26.31	46.45	33.6	84.62	70
$LogOptPlus_{J48}$	22.97	22.63	22.8	87.16	58.8	29.17	17.71	22.04	89.51	57.27
$LogOptPlus_{LOG}$	14.65	17.57	15.98	84.53	60.89	25.46	12.72	16.96	89.57	65
$LogOptPlus_{NB}$	9.69	92.97	17.54	26.8	70.02	23.85	48.98	32.08	82.63	73.27
$LogOptPlus_{RF}$	24.9	4.29	7.31	90.9	65.73	41.23	3.3	6.12	91.51	68.72
$LogOptPlus_{RBF}$	0	0	0	91.62	51.34	0	0	0	91.62	55.44
$LogOptPlus_{SVM}$	17.07	16.8	16.93	86.2	61.98	23.93	11.81	15.81	89.47	65.55
	Project: Tomcat → CloudStack Source Instances: 16991, Target Instances: 65392 Features:1398					Project: Hadoop → CloudStack Source Instances: 32143, Target Instances: 65392 Features:1232				
Classifier	LP (%)	LR (%)	LF (%)	ACC (%)	RA (%)	LP (%)	LR (%)	LF (%)	ACC (%)	RA (%)
$LogOptPlus_{ADA}$	0	0	0	91.36	77.11	0	0	0	91.36	76.07
$LogOptPlus_{ADT}$	41.94	0.46	0.91	91.34	80.93	0	0	0	91.36	77.84
$LogOptPlus_{BN}$	32.28	56.57	41.11	85.99	74.05	40.9	58.52	48.15	89.1	77.06
$LogOptPlus_{J48}$	8.72	18.96	11.95	75.84	57.08	28.05	24.41	26.11	88.05	56.94
$LogOptPlus_{LOG}$	7.5	18.57	10.68	73.15	42.86	40.33	5.61	9.85	91.12	57.07
$LogOptPlus_{NB}$	27.37	58.48	37.29	82.99	77.41	35.27	61.33	44.78	86.93	81.43
$LogOptPlus_{RF}$	40.64	1.8	3.46	91.28	62.31	59.46	2.72	5.21	91.43	74.69
$LogOptPlus_{RBF}$	0	0	0	91.36	64.29	0	0	0	91.36	56.8
$LogOptPlus_{SVM}$	6.86	15.8	9.57	74.19	45.21	46.2	9.15	15.27	91.23	70.65
	Project: Tomcat → Hadoop Source Instances: 16991, Target Instances: 32143 Features:1398					Project: CloudStack → Hadoop Source Instances: 65392, Target Instances: 32143 Features:1246				
Classifier	LP (%)	LR (%)	LF (%)	ACC (%)	RA (%)	LP (%)	LR (%)	LF (%)	ACC (%)	RA (%)
$LogOptPlus_{ADA}$	0	0	0	89.4	68.07	33.41	45.64	38.58	84.6	67.62
$LogOptPlus_{ADT}$	26.47	0.26	0.52	89.35	71.75	44.39	16.14	23.68	88.97	67.98
$LogOptPlus_{BN}$	29.75	34.16	31.81	84.47	68.81	14.34	75.84	24.12	49.41	67.1
$LogOptPlus_{J48}$	29.11	14.65	19.49	87.17	64.16	22.87	23.22	23.04	83.56	56.7
$LogOptPlus_{LOG}$	12.34	17.35	14.42	78.17	53	19.6	20.93	20.24	82.52	57.99
$LogOptPlus_{NB}$	24.75	37.33	29.77	81.33	70.46	11.31	96.04	20.23	19.72	68.52
$LogOptPlus_{RF}$	33.33	0.91	1.77	89.3	64.13	38.32	6.16	10.62	89	66.73
$LogOptPlus_{RBF}$	0	0	0	89.4	54.39	0	0	0	89.4	52.29
$LogOptPlus_{SVM}$	18.61	9.13	12.25	86.14	58.73	20.42	21.87	21.12	82.68	61.33

5.2. RQ 2: What are the performances of $CLIF_{NB}$ classifiers for cross-project if-blocks logging prediction?

Motivation: In RQ 2 we investigate the performances of $CLIF_{NB}$ classifiers. $CLIF_{NB}$ classifiers use NB classifier for numeric textual feature generation. We use NB classifier for numeric textual feature generation because it gives the best performance for cross-project if-blocks logging prediction in our initial investigation (refer to RQ 1).

Approach: We have total eight $CLIF_{NB}$ classifiers: $CLIF_{NB}^{ADA}$, $CLIF_{NB}^{ADT}$, $CLIF_{NB}^{BN}$, $CLIF_{NB}^{J48}$, $CLIF_{NB}^{LOG}$, $CLIF_{NB}^{NB}$, $CLIF_{NB}^{RF}$ $CLIF_{NB}^{SVM}$. We did not create $CLIF_{NB}^{RBF}$ because RBF classifier give 0% LF for all the source and target projects in our initial investigation (refer to RQ1). To answer this RQ, we compute LF and RA values of all the $CLIF_{NB}$ classifiers at a threshold learned in model building phase (refer to Section 3).

Results: Table 5 presents the LF and RA values of all the CLIFNB classifiers. Table 5 presents that $CLIF_{NB}^{LOG}$ and $CLIF_{NB}^{ADA}$ classifiers give the highest LF for four and two source and target project pairs, respectively. $CLIF_{NB}^{LOG}$ and $CLIF_{NB}^{ADA}$ give the highest LF of 50.96% and 50.86% on Hadoop → CloudStack and Tomcat → CloudStack project pairs, respectively. $CLIF_{NB}^{ADT}$ model give good performance i.e., near to the best CLIF classifier, on three source and target project pairs. We observe similar results for RA metric of these three classifiers, with $CLIF_{NB}^{ADA}$ classifier giving the highest RA on 4 source and target project pairs. $CLIF_{NB}^{BN}$, $CLIF_{NB}^{J48}$ and $CLIF_{NB}^{NB}$ classifier also give good performance for LF values. For example, $CLIF_{NB}^{NB}$ give LF value of 35.06% (Hadoop → Tomcat) and 37.91% (CloudStack → Hadoop) which close to the $CLIF_{NB}^{LOG}$ classifier (i.e., the best per forming classifier for these two source and target project pairs). However, RA values of the $CLIF_{NB}^{BN}$, $CLIF_{NB}^{J48}$ and $CLIF_{NB}^{NB}$ classifier are considerably lower as compared to the best performing CLIF classifiers. For example, for Hadoop CloudStack project pair, $CLIF_{NB}^{LOG}$ give 50.96% LF and 81.8% RA. Whereas, $CLIF_{NB}^{NB}$ give 47.74% LF and only 74.86% RA. $CLIF_{NB}^{RF}$ and $CLIF_{NB}^{SVM}$ perform the worst (among CLIFNB classifiers) and give poor results for both the metrics. We also observe that for all the CLIFNB classifiers threshold values are different from traditional 0.5 threshold value.

> $CLIF_{NB}^{LOG}$, $CLIF_{NB}^{ADA}$ and $CLIF_{NB}^{ADT}$ classifier performs the best among all the $CLIF_{NB}$ classifiers.

Table 5
Cross project if-blocks logging prediction by $CLIF_{NB}$ classifier.

	Project: CloudStack → Tomcat Source Instances:65392,Target Instances: 16991 Features:19			Project: Hadoop → Tomcat Source Instances: 32143,Target Instances: 16991 Features:19		
ALGO	TH	LF (%)	RA (%)	TH (%)	LF (%)	RA (%)
$CLIF_{NB}^{ADA}$	0.18	**34.81**	70.57	0.16	35.76	73.88
$CLIF_{NB}^{ADT}$	0.39	33.77	**71.91**	0.32	33.55	73.72
$CLIF_{NB}^{BN}$	0.93	33.08	65.11	0.37	35.09	66.62
$CLIF_{NB}^{J48}$	0.19	29.17	64.8	0.07	33.75	62.34
$CLIF_{NB}^{LOG}$	0.27	34.38	70.34	0.16	**35.85**	**75.08**
$CLIF_{NB}^{NB}$	0.99	33.6	68.08	0.19	35.06	69.34
$CLIF_{NB}^{RF}$	0.36	28.8	68.34	0.2	28.84	68.5
$CLIF_{NB}^{SVM}$	0.08	29.91	69.82	0.11	13.54	51.82
	Project: Tomcat → CloudStack Source Instances: 16991,Target Instances: 65392 Features:19			Project: Hadoop → CloudSatck Source Instances: 32143,Target Instances: 65392 Features:19		
ALGO	TH	LF (%)	RA (%)	TH (%)	LF (%)	RA (%)
$CLIF_{NB}^{ADA}$	0.13	**50.86**	**79.04**	0.16	49.9	**84.76**
$CLIF_{NB}^{ADT}$	0.32	49.89	77.92	0.32	49.98	81.37
$CLIF_{NB}^{BN}$	0.41	48.32	72.72	0.37	49.85	74.73
$CLIF_{NB}^{J48}$	0.08	42.18	63.79	0.07	45.21	66.21
$CLIF_{NB}^{LOG}$	0.13	50.03	78.49	0.16	**50.96**	81.8
$CLIF_{NB}^{NB}$	0.1	39.24	72.76	0.19	47.74	74.86
$CLIF_{NB}^{RF}$	0.26	30.19	69.94	0.2	35.41	74.41
$CLIF_{NB}^{SVM}$	0.1	31.72	65.47	0.11	26.52	69.7
	Project: Tomcat → Hadoop Source Instances: 16991,Target Instances: 32143 Features:19			Project: CloudStack → Hadoop Source Instances: 65392,Target Instances: 32143 Features:19		
ALGO	TH	LF (%)	RA (%)	TH (%)	LF (%)	RA (%)
$CLIF_{NB}^{ADA}$	0.13	38.8	**73.88**	0.18	38.3	**71.18**
$CLIF_{NB}^{ADT}$	0.32	38.82	73.32	0.39	36.51	70.35
$CLIF_{NB}^{BN}$	0.41	37.53	68.85	0.93	33.4	66.01
$CLIF_{NB}^{J48}$	0.08	34.81	61.49	0.19	31.21	65.44
$CLIF_{NB}^{LOG}$	0.13	**38.83**	73.28	0.27	**38.57**	69.88
$CLIF_{NB}^{NB}$	0.1	32.07	67.99	0.99	37.91	68.7
$CLIF_{NB}^{RF}$	0.26	28.45	66.08	0.36	31.83	68.09
$CLIF_{NB}^{SVM}$	0.1	27.02	61.17	0.08	30.28	68.51

5.3. RQ 3: What are the performances of $CLIF_{BN}$ classifiers for cross-project if-blocks logging prediction?

Motivation: In RQ 3 we investigate the performances of $CLIF_{BN}$ classifiers. $CLIF_{BN}$ classifiers use BN classifier for numeric textual feature generation. We use BN classifier for numeric textual feature generation because it give good performance for cross-project if-blocks logging prediction in our initial investigation (refer to RQ 1).

Approach: We have total eight $CLIF_{BN}$ classifiers: $CLIF_{BN}^{ADA}$, $CLIF_{BN}^{ADT}$, $CLIF_{BN}^{BN}$, $CLIF_{BN}^{J48}$, $CLIF_{BN}^{Log}$, $CLIF_{BN}^{NB}$, $CLIF_{BN}^{RF}$, $CLIF_{BN}^{SVM}$. We did not create $CLIF_{BN}^{RBF}$ because RBF classifier give 0% LF for all the source and target projects in our initial investigation (refer to RQ1). To answer this RQ, we compute LF and RA values of all the $CLIF_{BN}$ classifiers at a threshold learned in the model building phase (refer to Section 3).

Results: Table 6 presents the LF values of all the CLIFBN classifiers. $CLIF_{BN}^{LOG}$ gives the highest LF for four source and target project pairs. $CLIF_{BN}^{LOG}$ give the LF of 50.63% (Hadoop → CloudStack), 39.08% (Tomcat → Hadoop), 38.62% (CloudStack → Hadoop) and 35.9% (Hadoop → Tomcat). $CLIF_{BN}^{ADT}$ give the highest LF on two source and target project pairs. $CLIF_{BN}^{ADT}$ give LF of 49.89% (Tomcat CloudStack) and 35.63% (CloudStack → Tomcat). $CLIF_{BN}^{ADT}$ gives the highest RA on four source and target project pairs i.e., on CloudStack → Tomcat, Hadoop → Tomcat, Tomcat → CloudStack, and Hadoop → CloudStack. $CLIF_{BN}^{ADA}$ also per-form well and give LF values close to the best performing $CLIF_{BN}^{BN}$ classifier on all the source and target projects. CLIFBN give good performance for four source and target projects. For two source and target project pairs, CloudStack → Tomcat and CloudStack → Hadoop, we have NW entries in Table 6. For these two project we are not able to train the classifier. It is because, the default WEKA implementation of BN classifier was not able to discritize very low numeric values of numeric textual feature. Although, scaling can be applied at pre-processing step to overcome this problem. However, currently we use default values of the all the features for classifier learning and, hence, do not apply any further pre-processing. $CLIF_{BN}^{NB}$ give good i.e., close to the best CLIFBN classifier, LF values on Hadoop → Tomcat (35.9%) and Hadoop → CloudStack (47.54%) projects. However, its RA values are considerably lower than the best per-BNforming CLIFBN classifier. Among all the $CLIF_{BN}$ classifiers, $CLIF_{BN}^{RF}$ and $CLIF_{BN}^{SVM}$ perform BN the worst and give poor results for both the metrics.

> $CLIF_{BN}^{LOG}$, $CLIF_{BN}^{ADA}$ and $CLIF_{BN}^{ADT}$ classifier performs the best among all the $CLIF_{BN}$ classifiers.

Table 6
Cross project if-blocks logging prediction by $CLIF_{BN}$ classifier, NW: Not Working.

	Project: CloudStack → Tomcat Source Instances: 65392,Target Instances:16991 Features:19			Project: Hadoop → Tomcat Source Instances: 32143,Target Instances: 16991 Features:19		
ALGO	TH	LF (%)	RA (%)	TH (%)	LF (%)	RA (%)
$CLIF_{BN}^{ADA}$	0.18	34.57	67.02	0.12	35.56	73.29
$CLIF_{BN}^{ADT}$	0.39	35.63	70.5	0.34	35.83	75.35
$CLIF_{BN}^{BN}$	0.98	NW	NW	0.4	34.79	66.99
$CLIF_{BN}^{J48}$	0.19	29.99	62.87	0.07	33.31	61.98
$CLIF_{BN}^{LOG}$	0.26	34.58	70.22	0.18	35.9	74.57
$CLIF_{BN}^{NB}$	0.98	29.97	68.06	0.17	34.34	68.71
$CLIF_{BN}^{RF}$	0.31	28.54	67.73	0.21	26.57	67.63
$CLIF_{BN}^{SVM}$	0.11	24.34	62.11	0.1	20.43	63.42
	Project: Tomcat → CloudStack Source Instances: 16991,Target Instances:65392 Features:19			Project: Hadoop → CloudStack Source Instances: 32143,Target Instances:65392 Features:19		
ALGO	TH	LF (%)	RA (%)	TH (%)	LF (%)	RA (%)
$CLIF_{BN}^{ADA}$	0.2	49.36	77.33	0.12	49.83	81.11
$CLIF_{BN}^{ADT}$	0.33	49.89	78.14	0.34	50.27	81.77
$CLIF_{BN}^{BN}$	0.52	48.13	73.16	0.4	50.11	74.39
$CLIF_{BN}^{J48}$	0.09	43.29	67.25	0.07	46.33	66.81
$CLIF_{BN}^{LOG}$	0.11	49.72	76.76	0.18	50.63	80.11
$CLIF_{BN}^{NB}$	0.12	36.75	70.96	0.17	47.54	73.59
$CLIF_{BN}^{RF}$	0.23	32.09	71.94	0.21	39.48	73.95
$CLIF_{BN}^{SVM}$	0.11	33.04	62.23	0.1	23.53	67.03
	Project: Tomcat → Hadoop Source Instances: 16991,Target Instances: 32143 Features:19			Project: CloudStack → Hadoop Source Instances: 65392,Target Instances: 32143 Features:19		
ALGO	TH	LF (%)	RA (%)	TH (%)	LF (%)	RA (%)
$CLIF_{BN}^{ADA}$	0.2	37.69	70.15	0.18	38.52	68.42
$CLIF_{BN}^{ADT}$	0.33	38.82	72.2	0.39	38.37	69.12
$CLIF_{BN}^{BN}$	0.52	36.06	68.55	0.98	NW	NW
$CLIF_{BN}^{J48}$	0.09	33.28	62.54	0.19	31.39	62.3
$CLIF_{BN}^{LOG}$	0.11	39.08	72.26	0.26	38.62	70.08
$CLIF_{BN}^{NB}$	0.12	28.72	66.99	0.98	34.74	69.11
$CLIF_{BN}^{RF}$	0.23	28.6	66.54	0.31	31.92	68.25
$CLIF_{BN}^{SVM}$	0.11	20.35	55.82	0.11	24.33	62.13

5.4. RQ 4: What is the performance of CLIF model as compared to baseline classifier?

Motivation: In RQ 4, we compare the performances of $CLIF_{NB}$ and $CLIF_{BN}$ classifiers against the baseline classifiers. The answer to this RQ can provide important insight about effectiveness of CLIF classifiers for improving the cross-project if-blocks logging prediction performances.

Approach: To answer RQ 4, we compute value of the LF and RA values for all the LogOptPlus, $CLIF_{NB}$, and $CLIF_{BN}$ classifiers. For each source and target project pair LogOptPlus classifier giving the highest LF value is considered as baseline classifier. We then compare, for each source and target project pair, performances of the $CLIF_{NB}$ and $CLIF_{BN}$ classifiers against the baseline classifier.

Results: Tables 7 and 8 presents the LF and RA values and the respective improvements (as compared to the baseline classifier) for all the $CLIF_{NB}$ and $CLIF_{BN}$ classifiers. Tables 7 and 8 presents that several CLIF classifiers outperformed the baseline classifier in both LF and RA metrics. In Tables 7 and 8, cells containing ✓ indicate that the respective classifier improved the LF or RA values as compared to the baseline classifier and cell containing ★ indicate the classifier giving the best result for the respective source and target project pair. Among $CLIF_{NB}$ classifiers, $CLIF_{NB}^{ADA}$ and $CLIF_{NB}^{LOG}$ give the highest improvement on one and three source and target project pairs, respectively. $CLIF_{NB}^{ADA}$ give the highest improvement of 9.75% in LF on Tomcat → CloudStack project. $CLIF_{NB}^{LOG}$ give the highest improvement of 2.25% (Hadoop → Tomcat), 2.81% (Hadoop → CloudStack), and 7.02% (Tomcat → Hadoop) in LF. $CLIF_{NB}^{ADT}$, $CLIF_{NB}^{BN}$, $CLIF_{NB}^{J48}$ and $CLIF_{NB}^{NB}$ also show improvement in LF values on various source and target project pairs. $CLIF_{NB}^{ADA}$, $CLIF_{NB}^{ADT}$ and $CLIF_{LOG}$ give improvement in RA on all the six source and target projects, with $CLIF_{NB}^{ADA}$ giving the highest improvement on NB maximum (i.e, four) number of source and target projects. $CLIF_{NB}^{ADA}$ give improvement of 4.99% (Tomcat → CloudStack), 7.7% (Hadoop→CloudStack), 5.07% (Tomcat → Hadoop) and 3.56% (CloudStack → Hadoop).

Among CLIFBN classifiers, $CLIF_{BN}^{ADT}$ and $CLIF_{BN}^{LOG}$ give the highest improvement on one and three source and target project pairs, respectively. $CLIF_{BN}^{ADT}$ give the highest improvement of 8.78% on Tomcat → CloudStack project pair. $CLIF_{BN}^{LOG}$ give the highest improvement of 2.3% (Hadoop → Tomcat), 2.48% (Hadoop → CloudStack), and 7.27% (Tomcat → Hadoop). Overall, $CLIF_{BN}^{ADA}$, $CLIF_{BN}^{ADT}$, $CLIF_{BN}^{BN}$, and $CLIF_{BN}^{LOG}$ give improvement in LF on 4 out of 6 source and target projects. However, since we are not able to train $CLIF_{BN}^{BN}$ classifier for 2 source and target project pairs i.e., CloudStack → Tomcat and CloudStack → Hadoop, we are ignoring $CLIF_{BN}^{BN}$ from further analysis. $CLIF_{BN}^{ADT}$ give the highest improvement in RA on 4 source and target project pairs. $CLIF_{BN}^{ADT}$ give improvement of 3.35% (CloudStack → Tomcat), 5.35% (Hadoop → Tomcat), 4.09% (Tomcat → CloudStack) and 4.71% (Hadoop → CloudStack). $CLIF_{BN}^{ADT}$ and $CLIF_{BN}^{LOG}$ give improvement in RA on all source and target project pairs.

> Several $CLIF_{NB}$ and $CLIF_{BN}$ classifiers outperform the baseline classifier, with $CLIF_{NB}^{ADA}$ giving the maximum improvement of 9.75% (in LF metric) on Tomcat → CloudStack and 7.7% (in RA metric) on Hadoop → Tomcat source and target project pair.

5.5. RQ 5: What are the average performances of baseline, $CLIF_{NB}$, and $CLIF_{BN}$ classifiers over all the source and target project pairs ?

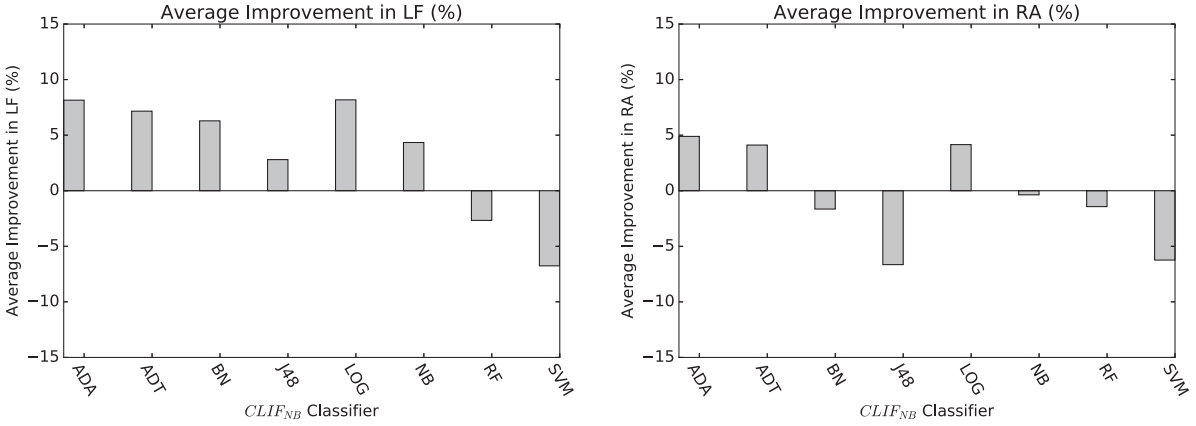
Motivation: In RQ 5, we investigate the performances off all classifiers over all source and target project pairs. We believe that answer to this research question can provide us important insight about the classifier that is generalizable to all the six source and target project pairs considered in this work

Table 7
LF and RA values for $CLIF_{NB}$ classifiers and the improvements obtained as compared to the baseline classifier, TC: Tomcat, CS: CloudStack, HD: Hadoop, ALGO: Algorithm, IMP: Improvement, NA: Not Applicable.

	Improvements obtained in LF using $CLIF_{NB}$ Clas sifiers											
	CS → TC		HD → TC		TC → CS		HD → CS		TC → HD		CS → HD	
ALGO	LF(%)	IMP (%)	LF(%)	IMP (%)	LF (%)	IMP (%)	LF (%)	IMP (%)	LF (%)	IMP (%)	LF (%)	IMP (%)
LogOptPlus(baseline)	34.98	NA	33.6	NA	41.11	NA	48.15	NA	31.81	NA	38.58	NA
$CLIF_{NB}^{ADA}$	34.81	−0.17	35.76	2.16 (✓)	50.86	**9.75**(★)	49.9	1.75 (✓)	38.8	6.99 (✓)	38.3	−0.3
$CLIF_{NB}^{ADT}$	33.77	−1.21	33.55	−0.05	49.89	8.78 (✓)	49.98	1.83 (✓)	38.82	7.01 (✓)	36.51	−2.1
$CLIF_{NB}^{BN}$	33.08	−1.9	35.09	1.49 (✓)	48.32	7.21 (✓)	49.85	1.7 (✓)	37.53	5.72 (✓)	33.4	−5.2
$CLIF_{NB}^{J48}$	29.17	−5.81	33.75	0.15 (✓)	42.18	1.07 (✓)	45.21	−2.9	34.81	3 (✓)	31.21	−7.4
$CLIF_{NB}^{LOG}$	34.38	−0.6	35.85	**2.25**(★)	50.03	8.92 (✓)	50.96	**2.81**(★)	38.83	**7.02**(★)	38.57	−0.01
$CLIF_{NB}^{NB}$	33.6	−1.38	35.06	1.46 (✓)	39.24	−1.87	47.74	−0.4	32.07	0.3 (✓)	37.91	−0.7
$CLIF_{NB}^{RF}$	28.8	−6.18	28.84	−4.76	30.19	−10.92	35.41	−12.7	28.45	−3.4	31.83	−6.8
$CLIF_{NB}^{SVM}$	29.91	−5.07	13.54	−20.06	31.72	−9.39	26.52	−21.6	27.02	−4.8	30.28	−8.3
	Improvements obtained in RA using $CLIF_{NB}$Classifiers											
ALGO	CS−>TC		HD−>TC		TC−>CS		HD−>CS		TC−>HD		CS−>HD	
LogOptPlus	RA(%)	IMP (%)	RA(%)	IMP (%)	RA (%)	IMP (%)	RA (%)	IMP (%)	RA (%)	IMP (%)	RA (%)	IMP (%)
(baseline)	67.15	NA	70	NA	74.05	NA	77.06	NA	68.81	NA	67.62	NA
$CLIF_{NB}^{ADA}$	70.57	3.42 (✓)	73.88	3.88 (✓)	79.04	**4.99**(★)	84.76	**7.7**(★)	73.88	**5.07**(★)	71.18	**3.56**(★)
$CLIF_{NB}^{ADT}$	71.91	**4.76**(★)	73.72	3.72 (✓)	77.92	3.87 (✓)	81.37	4.31 (✓)	73.32	4.51 (✓)	70.35	2.73 (✓)
$CLIF_{NB}^{BN}$	65.11	−2.04	66.62	−3.38	72.72	−1.33	74.73	−2.3	68.85	0.04	66.01	−1.6
$CLIF_{NB}^{J48}$	64.8	−2.35	62.34	−7.66	63.79	−10.26	66.21	−10.9	61.49	−7.3	65.44	−2.2
$CLIF_{NB}^{LOG}$	70.34	3.19 (✓)	75.08	**5.08**(★)	78.49	4.44 (✓)	81.8	4.74 (✓)	73.28	4.47 (✓)	69.88	2.26 (✓)
$CLIF_{NB}^{NB}$	68.08	0.93(✓)	69.34	−0.66	72.76	−1.29	74.86	−2.2	67.99	−0.8	68.7	1.08 (✓)
$CLIF_{NB}^{RF}$	68.34	1.19 (✓)	68.5	−1.5	69.94	−4.11	74.41	−2.7	66.08	−2.7	68.09	0.5
$CLIF_{NB}^{SVM}$	69.82	2.67 (✓)	51.82	−18.18	65.47	−8.58	69.7	−7.4	61.17	−7.6	68.51	0.9

Table 8
LF and RA values for $CLIF_{BN}$ classifiers and the respective improvements obtained as compared to the baseline classifier, TC: Tomcat, CS: CloudStack, HD: Hadoop, ALGO: Algorithm, IMP: Improvement, NA: Not Applicable, NW: Not Working.

	Improvements obtained in LF using $CLIF_{BN}$ Clas sifiers											
ALGO	CS → TC		HD → TC		TC → CS		HD → CS		TC → HD		CS → HD	
LogOptPlus	LF(%)	IMP (%)	LF(%)	IMP (%)	LF (%)	IMP (%)	LF (%)	IMP (%)	LF (%)	IMP (%)	LF (%)	IMP (%)
(baseline)	34.98	NA	33.6	NA	41.11	NA	48.15	NA	31.81	NA	38.58	NA
$CLIF_{BN}^{ADA}$	34.57	−0.41	35.56	1.96 (↙)	49.36	8.25 (↙)	49.83	1.68 (↙)	37.69	5.88 (↙)	38.52	−0.1
$CLIF_{BN}^{ADT}$	35.63	0.65	35.83	2.23 (↙)	49.89	**8.78**(★)	50.27	2.12 (↙)	38.82	7.01 (↙)	38.37	−0.2
$CLIF_{BN}^{BN}$	NW	NW	34.79	1.19 (↙)	48.13	7.02 (↙)	50.11	1.96 (↙)	36.06	4.25 (↙)	NW	NW
$CLIF_{BN}^{J48}$	29.99	−4.99	33.31	−0.29	43.29	2.18 (↙)	46.33	−1.8	33.28	1.47 (↙)	31.39	−7.2
$CLIF_{BN}^{LOG}$	34.58	−0.4	35.9	**2.3**(★)	49.72	8.61 (↙)	50.63	**2.48**(★)	39.08	**7.27**(★)	38.62	0.1
$CLIF_{BN}^{NB}$	29.97	−5.01	34.34	0.74 (↙)	36.75	−4.36	47.54	−0.6	28.72	−3.1	34.74	−3.8
$CLIF_{BN}^{RF}$	28.54	−6.44	26.57	−7.03	32.09	−9.02	39.48	−8.7	28.6	−3.2	31.92	−6.7
$CLIF_{BN}^{SVM}$	24.34	−10.64	20.43	−13.17	33.04	−8.07	23.53	−24.6	20.35	−11.5	24.33	−14.25
	Improvements obtained in RA using $CLIF_{BN}$ Classifiers											
ALGO	CS → TC		HD → TC		TC → CS		HD → CS		TC → HD		CS → HD	
LogOptPlus	RA(%)	IMP (%)	RA(%)	IMP (%)	RA (%)	IMP (%)	RA (%)	IMP (%)	RA (%)	IMP (%)	RA (%)	IMP (%)
(baseline)	67.15	NA	70	NA	74.05	NA	77.06	NA	68.81	NA	67.62	NA
$CLIF_{BN}^{ADA}$	67.02	−0.13	73.29	3.29 (↙)	77.33	3.28 (↙)	81.11	4.05 (↙)	70.15	1.34 (↙)	68.42	0.8 (↙)
$CLIF_{BN}^{ADT}$	70.5	**3.35**(★)	75.35	**5.35**(★)	78.14	**4.09**(★)	81.77	**4.71**(★)	72.2	3.39 (↙)	69.12	1.5 (↙)
$CLIF_{BN}^{BN}$	NW	NW	66.99	−3.01	73.16	−0.89	74.39	−2.7	68.55	−0.3	NW	NW
$CLIF_{BN}^{J48}$	62.87	−4.28	61.98	−8.02	67.25	−6.8	66.81	−10.3	62.54	−6.3	62.3	−5.3
$CLIF_{BN}^{LOG}$	70.22	3.07 (↙)	74.57	4.57 (↙)	76.76	2.71 (↙)	80.11	3.05 (↙)	72.26	**3.45**(★)	70.08	**2.46**(★)
$CLIF_{BN}^{NB}$	68.06	0.91 (↙)	68.71	−1.29	70.96	−3.09	73.59	−3.5	66.99	−1.8	69.11	1.49 (↙)
$CLIF_{BN}^{RF}$	67.73	0.58 (↙)	67.63	−2.37	71.94	−2.11	73.95	−3.1	66.54	−2.3	68.25	0.6 (↙)
$CLIF_{BN}^{SVM}$	62.11	−5.04	63.42	−6.58	62.23	−11.82	67.03	−10.	55.82	−13.01	62.13	−5.49



(a) Average improvement in LF using $CLIF_{NB}$ classifiers (b) Average improvement in RA using $CLIF_{NB}$ classifiers

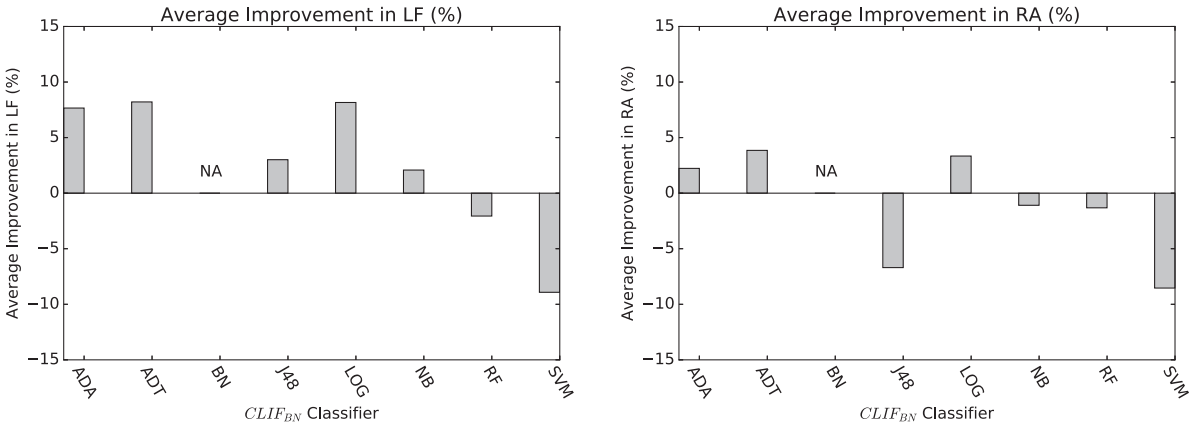


Fig. 4. Average improvement obtained using $CLIF_{NB}$ and $CLIF_{BN}$ classifiers, NA = Not Available.

Approach: To answer RQ 5, we compute the average LF and average RA of all the LogOptPlus classifiers. We select the LogOptPlus classifier giving the highest average LF as a baseline classifiers. We then compute average LF and average RA of all the $CLIF_{NB}$ and $CLIF_{BN}$ classifiers. We, finally compute average improvement obtained by each $CLIF_{NB}$ and $CLIF_{BN}$ classifier over the baseline classifier.

Results: LogOptPlusBN classifier gives the highest average LF among all the LogOpt-Plus classifiers, hence, we consider it as our baseline classifier for this experiment. LogOptPlusBN give average LF of 33.26% and average RA of 70.66%. Fig. 4 presents the improvement obtained in average LF and average RA using CLIFNB and CLIFBN classifiers as compared to the baseline classifiers. Fig. 4a presents that $CLIF_{NB}^{ADA}$, $CLIF_{NB}^{ADT}$, $CLIF_{NB}^{BN}$, $CLIF_{NB}^{J48}$, $CLIF_{NB}^{LOG}$ and $CLIF_{NB}^{NB}$ classifiers give improvement of 8.15%, 7.16%, 6.29%, 2.8%, 8.18% and 4.34%, respectively, in average LF. $CLIF_{NB}^{ADA}$, $CLIF_{NB}^{ADT}$, and $CLIF_{NB}^{LOG}$ also give improvement of 4.89%, 4.11%, and 4.15%, respectively, in average RA. Fig. 4c and d presents that $CLIF_{BN}^{ADA}$, $CLIF_{BN}^{ADT}$, $CLIF_{BN}^{J48}$, $CLIF_{BN}^{LOG}$ and $CLIF_{BN}^{NB}$ give improvement of 7.66%, 8.21%, 3.01%, 8.16% and 2.08%, respectively, in average LF. $CLIF_{BN}^{ADA}$, $CLIF_{BN}^{ADT}$, and $CLIF_{BN}^{LOG}$ give improvement of 2.23%, 3.85% and 3.34%, respectively, in average RA. Fig. 4c and d have Not Available ('NA') entries for $CLIF_{BN}^{BN}$ classifier because we are not able to train this classifier for CloudStack → Tomcat, CloudStack → Hadoop project pairs. $CLIF_{NB}^{RF}$, $CLIF_{NB}^{SVM}$, $CLIF_{BN}^{RF}$ And $CLIF_{BN}^{SVM}$ classifier did not give any improvement in average LF or average RA as com- BN pared to the baseline classifier.

> Several $CLIF_{NB}$ and $CLIF_{BN}$ classifiers outperform the baseline classifier. $CLIF_{NB}^{ADT}$, $CLIF_{NB}^{LOG}$, $CLIF_{BN}^{LOG}$, $CLIF_{NB}^{ADA}$, $CLIF_{BN}^{ADA}$, $CLIF_{NB}^{ADT}$, give improvement of 8.21%, 8.18%, 8.16%, 8.15%, 7.66%, and 7.16%, respectively, in average LF, and improvement of 3.85%, 4.15%, 3.34%, 4.89%, 2.23%, and 4.12%, respectively, in average RA.

6. Discussion

6.1. Performance of the classifiers

LOG, ADT, and ADA based CLIF classifiers give good results for cross-project if-blocks logging prediction. We believe that ADT based classifiers performed well as it is a generalization of decision trees and uses a concept of voted decision tree along with combining the advantages of boosting for improving accuracy (Freund and Mason, 1999). Previous research shows that ADT has performed well on Software Engineering dataset and problems and is suitable for imbalanced dataset cases (Antoniol et al., 2008; Panichella et al., 2015; Tan et al., 2015). Antoniol et al. (2008) apply ADT to classify if an issue is a bug or enhancement. Panichella et al. (2015) apply ADT for the purpose of categorizing mobile app user reviews for software maintenance and evolution. Tan et al. (2015) use ADT for defect prediction and mention that ADT has performed best in previous work. In CLIF, as we combine text mining classifiers output with Boolean and numeric features and then learning a classifier on it. Hence, the upper classifier is basically working as a meta classifier. It is found that LOG regression give good performance when used as a meta classifier in ensemble techniques like stacking (Jonsson et al., 2016; Witten et al., 2011). BN and NB classifiers give good performance for cross-project if-blocks logging prediction with LogOptPlus classifier. NB give good results in other text classification tasks (Shivaji et al., 2013). We observe that among all the classifiers RBF give the worst performance for cross-project if-blocks logging prediction. RBF is an artificial neural network based classifier which requires parameter tuning for better performance. Lal et al. (2017) also achieved the same results for cross-project catch-blocks logging prediction with RBF classifier.

6.2. CLIF performance on different projects

We evaluated performance of the CLIF classifiers on six source and target project pairs. CLIF classifiers give considerable improvement in RA metric on all the source and target project pairs. CLIF classifiers also give considerable improvement in LF metric on four source and target project pairs (Tomcat → CloudStack, Tomcat → Hadoop, Hadoop → CloudStack, Hadoop → Tomcat). CLIF classifiers did not give much improvement in LF metric in two cases when CloudStack project is used for model building. We believe that this happens because in this work, we use NB and BN classifiers as text mining classifiers for generating numerical textual features. We use NB and BN because these two classifiers give the best results on most of the source and target project pairs in our initial investigation (referto Section 5.1). However, on CloudStack → Tomcat, CloudStack → Hadoop project pair ADA classifier give the highest LF value. We believe that this is one of the reason CLIF classifiers give poor performance (in any case no worst than baseline classifier) when CloudStack is used as source project. In future, we plan to create one more CLIF variant using ADA as a text mining classifiers for numerical textual feature generation.

6.3. Effectiveness of threshold learning

We use threshold learning to address the class imbalance problem of logging dataset. We use threshold value learned on source project (training dataset) for if-block logging prediction of target project (testing dataset). Tables 5 and 6 presents the threshold value of CLIF classifiers. Tables 5 and 6 shows that CLIF classifiers give best results on source project for threshold value other than 0.5. This shows that threshold learning can be beneficial in improving the logging prediction performance in case of imbalanced logging datatset.

6.4. Effectiveness of numeric textual features

In this work, we use two metrics, LF and RA, for comparing the prediction performances of the baseline classifier and CLIF classifiers. CLIF uses numerical textual features generation and threshold learning for improving the cross-project if-blocks logging prediction performance. We observe that several CLIF classifiers give improvement in RA metric as compared to baseline LogOpt-Plus Classifier on all source and target project pairs. RA is not affected by the decision threshold used for prediction. Hence, all the improvements obtained in RA metric is because of the uses of numeric textual features. This shows that numeric textual features can be very useful in improving the performance of machine learning classifiers.

7. Threats to validity

In this section, we discuss threats to validity related to the work presented in this paper.

7.1. Threat to external validity

The external threat to validity is concerned with the generalization of the results. In our study, we conduct all our experiments on three open source Java projects. These projects have different

domains, sizes, and development history. Based on our study, we have found that the proposed approach i.e., CLIF is effective in improving cross-project if-blocks logging prediction performance. However, our results may not be generalizable to all the Java projects, because we have only studied three projects from the ASF. Additionally, studied are required on Java project from ASF as well on other Java ecosystem such as Android, Eclipse. In addition to this, more research is required for other types of projects such as closed source projects or projects written in other programming languages (e.g., C#, Python etc.) may have different logging practices. However, to mitigate this threat we conduct our experiments on three large and standard projects having good logging practices. In future, we plan to conduct experiments on more projects from different languages and domains.

7.2. Threat to construct validity

It is concerned with how we identify log statements. We create 26 regular expressions to find all the log statements. The manual inspection reveals that we identify all the types of log statements. However, there is still a possibility that we missed some types of log statements.

We consider logging statements inserted by the software developers in the source code as gold standard or optimal. However, there is a possibility of error or non-optimal logging in the source code by the software developer which can affect the performance of the prediction model. However, all the three projects are long lived and actively maintained, hence, we can assume good logging in all the three projects.

We predict cross-project if-blocks logging prediction on the basis of Boolean, numeric and numeric textual features that we extracted in model building phase. There is a possibility that there exists a better set of predictors for cross-project if-blocks logging prediction. In future, we plan to extend this study with more predictors.

7.3. Threat to Internal validity

At the threshold learning step we randomly divide the training dataset into two part. This random division can lead to biases in the threshold learned. To mitigate this threat we perform this split 10 times and report the threshold value that performed the best on average.

We use default WEKA parameters for all the classifiers. Tuning classifier parameter is important and can be beneficial in improving the prediction performance. However, as a first step towards cross-project if-blocks logging prediction we use default classifier parameters. In future, we plan to extend this study and notice the effect of parameter tuning on classifier performance.

We mainly consider LF metric for comparing results of different classifiers. LF (i.e., F1-score) is a well known and a widely used metric for classifier performance evaluation on imbalanced dataset. However, depending on different application needs other evaluation metrics might be desirable. In such cases, at threshold learning step we can select the threshold value that is maximizing the desirable metric. In future, we plan to consider more metrics to compare the performances of different classifiers.

8. Conclusion and future work

In this work, we propose CLIF, a three level, machine learning based model for cross-project if-blocks logging prediction. CLIF uses three types of features (Boolean, numeric, and textual) for model building. At the level 1, CLIF coverts textual features to numeric textual features. At level 2, CLIF combines numeric textual features with Boolean and numeric features and forms a final features vector. CLIF then learns a machine learning classifier on final feature vector. At level 3, CLIF learns an effective decision threshold boundary for prediction on imbalanced dataset. At level 1, we use two text mining classifiers NB and BN. We use NB and BN because they give better results as compared to other classifiers in our initial investigation, as inferred in Section 5.1. At level 2, we use eight machine learning classifiers i.e., ADA, ADT, BN, J48, LOG, NB, RF, SVM. In total, we generate sixteen CLIF classifies i.e., eight $CLIF_{NB}$ and $CLIF_{BN}$ classifiers. Experimental results on three open source projects show that CLIF is effective in improving cross-project if-blocks logging prediction. Several $CLIF_{NB}$ and $CLIF_{BN}$ classifiers outperform the baseline *LogOptPlus* classifier. Experimental results in Section 5 show that among $CLIF_{NB}$ and $CLIF_{BN}$ classifiers, $CLIF_{NB}^{ADT}$, $CLIF_{NB}^{LOG}$, $CLIF_{NB}^{ADA}$, $CLIF_{NB}^{ADT}$, $CLIF_{BN}^{LOG}$, and $CLIF_{BN}^{ADA}$ performs the best. $CLIF_{BN}^{ADT}$, $CLIF_{NB}^{LOG}$, $CLIF_{BN}^{LOG}$, $CLIF_{NB}^{ADA}$, $CLIF_{BN}^{ADA}$, and $CLIF_{NB}^{ADT}$ give improvement of 8.21%, 8.18%, 8.16%, 8.15%, 7.66%, and 7.16%, NB respectively, in average LF, and improvement of 3.85%, 4.15%, 3.34%, 4.89%, 2.23%, and 4.12%, respectively, in average RA.

The work presented in this paper, is the first in-depth study of cross-project if-blocks logging prediction. It opens the door to many interesting future directions. At present, we use the default parameters for all the classifiers. In future, we plan to work on parameter tuning to find the optimal classification parameters to further improve the performance of CLIF classifier. At present, we use only one numeric textual feature. In future, we plan to combine multiple numeric textual features and add one more level to CLIF. We plan to extend the functionality of CLIF for other types of code constructs such as catch-blocks. We also plan to develop a CLIF plug-in for Eclipse IDE that can be used to give logging suggestions to the software developers.

References

Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.-G., 2008. Is it a bug or an enhancement?: a text-based approach to classify change requests. Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds. ACM, p. 23.

Apache. Apache project homepage. https://commons.apache.org/proper/commons-logging/. [Online; accessed 18-March-2016].

Blackberry enterprise server logs submission. https://salesforce.services.blackberry.com/webforms/beslogs. (Online; accessed 4-June-2016).

Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P., 2002. Smote: synthetic minority over-sampling technique. J. Artif. Intell. Res. 16, 321–357.

Chawla, N.V., Lazarevic, A., Hall, L.O., Bowyer, K.W., 2003. Smoteboost: Improving prediction of the minority class in boosting. European Conference on Principles of Data Mining and Knowledge Discovery. Springer, pp. 107–119.

Cloudstack. Cloudstack project homepage. https://cloudstack.apache.org/. (Online; Accessed 18-March-2016).

Dai, W., Yang, Q., Xue, G.-R., Yu, Y., 2007. Boosting for transfer learning. Proceedings of the 24th International Conference on Machine Learning, ICML '07. ACM, New York, NY, USA, pp. 193–200.

Ding, R., Zhou, H., Lou, J.-G., Zhang, H., Lin, Q., Fu, Q., Zhang, D., Xie, T., 2015. Log2: a cost-aware logging mechanism for performance diagnosis. USENIX Annual Technical Conference, pp. 139–150.

Freund, Y., Mason, L., 1999. The alternating decision tree learning algorithm. In Icml 99, 124–133.

Fu, Q., Lou, J.-G., Wang, Y., Li, J., 2009. Execution anomaly detection in distributed systems through unstructured log analysis. Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, ICDM '09. IEEE Computer Society, Washington, DC, USA.

Fu, Q., Zhu, J., Hu, W., Lou, J.-G., Ding, R., Lin, Q., Zhang, D., Xie, T., 2014. Where do developers log? an empirical study on logging practices in industry. Companion Proceedings of the 36th International Conference on Software Engineering, ICSE Companion, pp. 24–33.

Guo, H., Viktor, H.L., 2004. Learning from imbalanced data sets with boosting and data generation: the databoost-im approach. ACM SIGKDD Explor. Newsletter 6 (1), 30–39.

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

Hadoop. Hadoop bug. https://issues.apache.org/jira/browse/MAPREDUCE-5501. (Online; Accessed 18-may-2016).

Hadoop. Hadoopt project homepage. http://hadoop.apache.org/. (Online; accessed 18-March-2016).

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., Nov. 2009. The weka data mining software: an update. SIGKDD Explor. Newsl. 11 (1), 10–18.

Han, J., Kamber, M., Pei, J., 2011. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

He, H., Garcia, E.A., 2009. Learning from imbalanced data. IEEE Trans. Knowl. Data Eng. 21 (9), 1263–1284.

Jonsson, L., Borg, M., Broman, D., Sandahl, K., Eldh, S., Runeson, P., 2016. Automated bug assignment: ensemble-based machine learning in large scale industrial contexts. Empirical Softw. Eng. 21 (4), 1533–1578.

Kim, S., Whitehead Jr., E.J., Zhang, Y., 2008. Classifying software changes: clean or buggy? IEEE Trans. Software Eng. 34 (2), 181–196.

Kubat, M., Matwin S., et al., 1997. Addressing the curse of imbalanced training sets: one-sided selection. In: ICML, vol. 97. Nashville, USA. pp. 179–186.

Lal, S., Sureka, A., 2016. Logopt: Static feature extraction from source code for automated catch block logging prediction. 9th India Software Engineering Conference (ISEC), pp. 151–155.

Lal, S., Sardana, N., Sureka, A., 2016a. Improving logging prediction on imbalanced datasets: a case study on open source java projects. Int. J. Open Source Softw. Processes (IJOSSP) 7 (2), 43–71.

Lal, S., Sardana, N., Sureka, A., 2016b. Logoptplus: Learning to optimize logging in catch and if programming constructs. 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), vol. 1, pp. 215–220.

Lal, S., Sardana, N., Sureka, A., 2017. Eclogger: cross-project catch-block logging prediction using ensemble of classifiers. E-inf. Softw. Eng. 11.

Liu, H., Yu, L., 2005. Toward integrating feature selection algorithms for classification and clustering. IEEE Trans. Knowl. Data Eng. 17 (4), 491–502.

Manning, C.D., Raghavan, P., Schütze, H., 2008. Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA.

Nagaraj, K., Killian, C., Neville, J., 2012. Structured comparative analysis of systems logs to diagnose performance problems. Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, p. 26.

Nam, J., Pan, S.J., Kim, S., 2013. Transfer defect learning. 2013 35th International Conference on Software Engineering (ICSE), pp. 382–391.

Pan, S.J., Yang, Q., 2010. A survey on transfer learning. IEEE Trans. Knowl. Data Eng. 22 (10), 1345–1359.

Pan, S.J., Tsang, I.W., Kwok, J.T., Yang, Q., 2011. Domain adaptation via transfer component analysis. IEEE Trans. Neural Networks 22 (2), 199–210.

Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C.A., Canfora, G., Gall, H.C., 2015. How can i improve my app? classifying user reviews for software maintenance and evolution. 2015 IEEE international conference on Software maintenance and evolution (ICSME). IEEE, pp. 281–290.

Shivaji, S., Whitehead, E.J., Akella, R., Kim, S., 2013. Reducing features to improve code change-based bug prediction. IEEE Trans. Software Eng. 39 (4), 552–569.

Sigelman, B.H., Barroso, L.A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S., Shanbhag, C., 2010. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Technical report. Google.

Tan, M., Tan, L., Dara, S., Mayeux, C., 2015. Online defect prediction for imbalanced data. Proceedings of the 37th International Conference on Software Engineering, vol. 2. IEEE Press, pp. 99–108.

Tomcat. Tomcate project homepage. http://tomcat.apache.org//. (Online; Accessed 18-March-2016).

Valdivia Garcia, H., Shihab, E., 2014. Characterizing and predicting blocking bugs in open source projects. Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014. ACM, New York, NY, USA, pp. 72–81.

Witten, I.H., Frank, E., Hall, M.A. Data mining: Practical machine learning tools and techniques. 2011.

Xia, X., Lo, D., McIntosh, S., Shihab, E., Hassan, A.E., 2015. Cross-project build co-change prediction. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). March 2015. pp. 311–320.

Xia, X., Lo, D., Shihab, E., Wang, X., Yang, X., 2015b. Elblocker: predicting blocking bugs with ensemble imbalance learning. Inf. Softw. Technol. 61, 93–106.

Xia, X., Lo, D., Correa, D., Sureka, A., Shihab, E., 2016. It takes two to tango: Deleted stack overflow question prediction with text and meta features. 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), vol. 1, pp. 73–82.

Yu, L., Liu, H., 2004. Efficient feature selection via analysis of relevance and redundancy. J. Mach. Learn. Res. 5, 1205–1224.

Zhang, Y., Lo, D., Xia, X., Sun, J., 2015. An empirical study of classifier combination for cross-project defect prediction. Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual, vol. 2, pp. 264–269.

Zhu, J., He, P., Fu, Q., Zhang, H., Lyu, M., Zhang, D., 2015. Learning to log: helping developers make informed logging decisions. 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), vol. 1, pp. 415–425.