2015 Asia-Pacific Software Engineering Conference

EFSPredictor: Predicting Configuration Bugs With Ensemble Feature Selection

Bowen Xu*, David Lo†, Xin Xia*♮, Ashish Sureka‡, and Shanping Li*
*College of Computer Science and Technology, Zhejiang University, Hangzhou, China
†School of Information Systems, Singapore Management University, Singapore
‡Indraprastha Institute of Information Technology, IIIT-Delhi, India
max_xbw@zju.edu.cn, davidlo@smu.edu.sg, and xxia@zju.edu.cn, ashish @iiitd.ac.in, shan@zju.edu.cn

***Abstract*—The configuration of a system determines the system behavior and wrong configuration settings can adversely impact system's availability, performance, and correctness. We refer to these wrong configuration settings as *configuration bugs*. The importance of configuration bugs has prompted many researchers to study it, and past studies can be grouped into three categories: detection, localization, and fixing of configuration bugs. In the work, we focus on the detection of configuration bugs, in particular, we follow the line-of-work that tries to predict if a bug report is caused by a wrong configuration setting. Automatically prediction of whether a bug is a configuration bug can help developers reduce debugging effort. We propose a novel approach named *EFSPredictor* which applies ensemble feature selection on the natural-language description of a bug report. It uses different feature selection approaches (e.g., ChiSquare, GainRatio and Relief) which output different ranked lists of textual features. Next, to obtain a set of representative textual features, *EFSPredictor* first assigns different scores to the features outputted by these feature selection approaches. Next, for each feature, *EFSPredictor* sums up the scores outputted by the multiple ranked lists, and outputs the top features (e.g., 25% of the total number of features) as the selected features. Finally, *EFSPredictor* builds a prediction model based on the selected features. We conduct experiments on 5 bug report datasets (i.e., accumulo, activemq, camel, flume, and wicket) containing a total of 3,203 bugs. The experiment results show that, on average across the 5 projects, *EFSPredictor* achieves an F1-score to 0.57, which improves the state-of-the-art approach proposed by Xia et al. by 14%.**

***Keywords*—Ensemble Feature Selection, Configuration Bugs, Data Mining**

I. Introduction

Developers often allow users to customize system behaviors flexibly via configuration options. Furthermore, through sharing of libraries, registry entries, environment variables and configuration files, applications can interact with one another conveniently. Unfortunately, this flexibility incurs a cost, software systems become complex and hard to set up resulting in various configuration problems.

There have been a number of studies which show that configuration bugs (i.e., misconfiguration) significantly impact the availability, performance, and correct working of software systems. For example, a substantial amount of technical support time, which contributes 17% of the total cost of ownership of today's desktop PCs, is spent on troubleshooting, and most of the problems are due to misconfigurations [35]. As Gallagher reported on June 2015, an executive of Airbus Group has confirmed that the crash of an Airbus A400M military transport was caused by a faulty software configuration [2]. Recent research has even found that operators frequently misconfigure Internet services, causing various availability and performance problems [49]. Rabkin and Katz's study indicates that misconfigurations are the dominant cause of Hadoop clusters breakdown [26]. An empirical study of issues in Amazon EC2 APIs points out that misconfigurations by API users are the main factor leading to interaction faults [21]. A significant portion of misconfigurations can cause hard-to-diagnose failures (e.g., crashes, hangs, severe performance degradation), indicating that systems should be better-equipped to handle misconfigurations [43]. Xu et al. [41] advocate software developers to play an active role in handling configuration bugs rather than blindly blame users for misconfigurations.

Many research studies have been carried out on detecting, locating and fixing configuration bugs. In this work, we focus on the problem of detecting configuration bugs. In particular, we extend the line-of-work that identifies configuration bugs among the many bug reports that are submitted to an issue tracking system (aka. configuration bug reports prediction). Given a bug report, identifying whether it is a configuration bug or not can help developers reduce their debugging effort. For those that are configuration bugs, developers can focus their effort on checking configuration files rather than source code. Two most recent works in this direction are by Arshad et al. [8] and Xia et al. [38]. Xia et al. use Arshad et al.'s method as a baseline and their experiment results show that their proposed approach improves the F1-score of Arshad et al.'s method. Despite this improvement, the F1-score achieved by Xia et al.'s approach is not optimal, and in this work, we aim to improve it further.

We present a new approach named *EFSPredictor* which combines multiple feature selection technologies to obtain a set of representative features, and then builds a statistical prediction model on these representative features extracted from historical bug reports with known labels (i.e., non-configuration or configuration). This statistical model can then be used to predict a new bug report as either a configuration bug report or not. *EFSPredictor* first uses different feature selection approaches (e.g., ChiSquare, GainRatio and Relief) to output different ranked lists of textual features. Then, *EFSPredictor* assigns different scores to the features outputted by these feature selection approaches. Next, for each feature,

♮Corresponding author.

1530-1362/15 $31.00 © 2015 IEEE
DOI 10.1109/APSEC.2015.38

TABLE I. THE F1-SCORE FOR 5 SINGLE FEATURE SELECTION ALGORITHMS BUILT ON 5 DATASETS

Projects	Relief	ChiSquare	Filtered	GainRatio	OneR
accumulo	0.55	0.60	0.61	0.61	0.58
activemq	0.37	0.44	0.44	0.44	0.40
camel	0.49	0.52	0.52	0.52	0.53
flume	0.57	0.55	0.55	0.54	0.53
wicker	0.35	0.44	0.44	0.44	0.34

EFSPredictor sums up the scores for each of the ranked lists, and outputs the top features with the highest scores (e.g., top 25% features). Finally, *EFSPredictor* builds a prediction model based on the selected top features.

We evaluate our approach on 5 datasets consisting of 3,203 bug reports from different open source software projects: accumulo [1], activemq [3], camel [4], flume [5], and wicket [6]. The experiment results show that, on average across the 5 projects, *EFSPredictor* achieves an F1-score of 0.57, which improves the F1-score achieved by the state-of-the-art approach proposed by Xia et al. by 14%.

The main contributions of this paper are as follows:

1) We introduce a new approach which uses an ensemble of feature selection technologies on natural-language descriptions in bug reports to identify configuration bugs.
2) Through an experiment using 5 datasets containing a total of 3,203 bugs, we demonstrate the effectiveness of our approach. Our approach *EFSPredictor* outperforms the state-of-the-art method proposed by Xia et al. by a substantial margin.

The remainder of this paper is structured as follows. We elaborate the motivation of our work in Section II. We describe the overall framework and the details of *EFSPredictor* in Section III. We present our experiments and the results in Section IV. We review related work in Section V. We conclude and mention future work in Section VI.

II. MOTIVATION

In this section, we elaborate the intuition of our proposed tool *EFSPredictor* which is based on an ensemble of feature selection algorithms. We do so by describing why an ensemble of feature selection algorithms can boost the effectiveness of predicting configuration bugs.

Many feature selection algorithms have been proposed in the machine learning field. Each of these algorithms is based on a different heuristic to evaluate the importance of a feature. Since they are only heuristics, some feature selection algorithms may perform better than others on some cases, but worse on other cases. One way to address the weaknesses of these feature selection algorithms is to integrate them together in order to make a comprehensive judgement. Many studies, e.g., [23], [28], [32] have argued that using an ensemble of feature selection techniques has a great promise for high-dimensional problems with small sample sizes. By using an ensemble of feature selection techniques, we can utilize the advantages of the various feature selection techniques to generate a set of most representative features to improve prediction results.

To demonstrate that the above hypothesis applies in our setting (i.e., configuration bug prediction), we perform an initial experiment using 5 feature selection algorithms, i.e., ChiSquare, Filter, GainRatio, OneR, Relief, on 5 datasets, i.e., accumulo, activemq, camel, flume, and wicket. We use naive Bayes multinomial, which was also used by Xia et al. [38], to build statistical prediction models from selected features. We use the standard 10-fold cross validation to evaluate the effectiveness of the models learned using features selected by the 5 algorithms, and utilize F1-score as a yardstick to evaluate the models' effectiveness.

Table I presents the F1-scores achieved using each of the 5 feature selection algorithms for the 5 datasets. We notice that none of these 5 feature selection algorithms can defeat all the other algorithms for all datasets. For example, for accumulo, Filter and GainRatio achieve the best performance. But OneR achieves the best performance on camel, and Relief achieves the best performance on flume. Since the relative effectiveness of different feature selection algorithms differs for different cases, in this paper, we aim to utilize the advantages of multiple feature selection algorithms that are ensembled together.

III. OUR PROPOSED APPROACH

In this section, we first present the overall framework of our proposed *EFSPredictor* in Section III-A. Then, we introduce the 5 feature selection techniques that we ensemble together in Section III-B. Next, we describe the details of a core computation step performed by *EFSPredictor* in Section III-C.

A. Overall Framework

Figure 1 presents the overall framework of *EFSPredictor*. The framework contains two phases: model building phase and prediction phase. In the model building phase, we build a statistical prediction model that is able to predict if a bug report corresponds to a configuration bug or not. This model is used in the prediction phase to predict if a new bug report is a configuration bug or not.

Given a training set of bug reports with known labels, *EFSPredictor* first extracts and preprocesses the natural language descriptions in the bug reports by tokenizing them, removing stop words, and stemming the tokens (Step 1). In the tokenization process, we extract words in bug reports. In the stop word removal process, we remove frequently appearing words that provide little help to differentiate one bug report from another. We use a list of stop words that is available at http://snowball.tartarus.org/algorithms/english/stop.txt. In the stemming process, we reduce each word to its root form, for example, words "write" and "written" are both reduced to "writ". We use a popular stemming algorithm namely the Porter stemmer [24]. After the preprocessing step, we represent the natural language description sof bug reports in the form of "bags of words" [10], and each pre-processed word is considered as a textual feature (Step 2). Next, we apply 5 different feature selection algorithms (i.e., ChiSquare, Filter, GainRatio, OneR, Relief) on the features extracted from the training bug reports (Step 3). This process produces five ranked list of features. *EFSPredictor* then selects a set of the most representative features by merging the five ranked lists into a single one and outputs the topmost features (Step 4). We refer

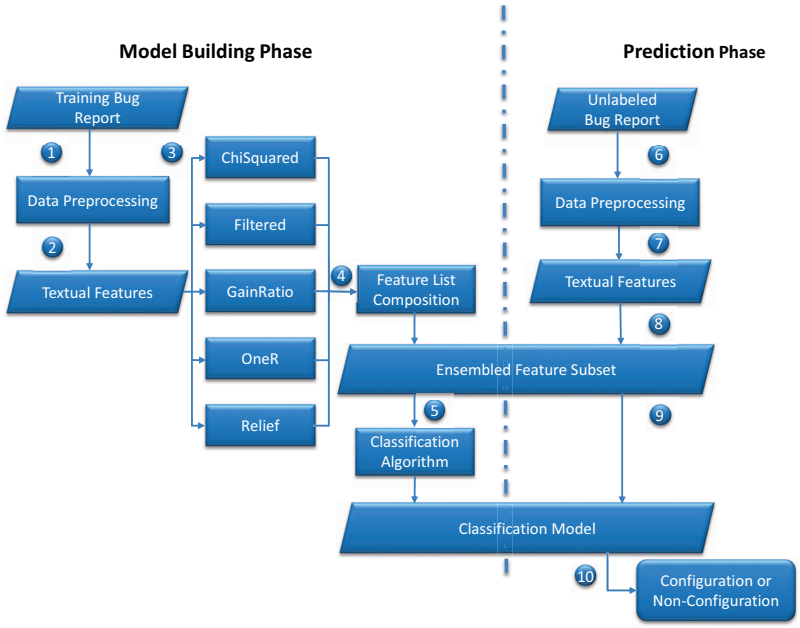

Fig. 1. Overall Framework of *EFSPredictor*

to this set as the *ensembled feature subset*. We only keep this ensembled feature subset from the training set of bug reports and build a statistical prediction model using a classification algorithm (Step 5). By default, we use naive Bayes multinomial as the classification algorithm.

After the statistical prediction model is built, in the prediction phase, for each unlabeled/new bug report, *EFSPredictor* first preprocesses it using the same preprocessing strategy as the model building phase (Step 6), which results in a set of textual features (Step 7). Next, we only keep the features that are in the *ensembled feature subset* (Step 8). Then, we input these features into the prediction model (Step 9), which outputs the prediction results: configuration bug or not (Step 10).

B. Feature Selection Techniques

Feature selection techniques try to identify features which are the most helpful in differentiating different classes (in our case: configuration bugs or not). Many studies show that feature selection can improve predictive accuracy, help learn model that is more compact, and reduce execution time [13]. In this paper, we consider 5 state-of-the-art feature selection techniques, namely ChiSquare, Filter, GainRatio, OneR, Relief to generate 5 ranked list of features that are then composed together. We use these 5 techniques since they were successfully used in many previous works [48], [33]. We describe these 5 feature selection techniques in a nutshell in the following paragraphs:

ChiSquare. ChiSquare is a popular feature selection method that uses chi-square test (a famous discrete data hypothesis testing method from statistics), to evaluates the correlation between each feature and the class label (in our case: configuration bug or not) and determine whether they are independent or correlated [17]. The chi-square (χ^2) value for a feature t considering a class c can be computed using Equation 1.

$$\chi^2(t,c) = \frac{N * (A * D - B * C)^2}{(A + C) * (B + D * (A + B) * (C + D)} \quad (1)$$

In the above equation, N is the total number of training bug reports, A is the number of bug reports with class c that contains feature (i.e., word) t, B is the number of bug reports not in class c that contains feature t, C is the number of bug reports in class c that does not contain feature t, and D is the number of bug reports not in class c that does not contain feature t. After the χ^2 scores of the features are computed, ChiSquare feature selection technique returns a ranked list of features sorted in descending order of their χ^2 scores. ChiSquare was used before to solve many text classification problems and showed promising results [11], [42].

Filter. Filter feature selection technique first resamples a training dataset by randomly duplicating some data instances (in our case: bug reports) in the minority class (i.e., class with the least members) such that the numbers of instances in both classes (in our case: configuration bugs or not) are equal. After the resampling step, it runs ChiSquare technique on the balanced training dataset to obtain a ranked list of features.

GainRatio. Another way to estimate the importance of a feature is to measure how much information the feature can bring to classify a data instance (in our case: a bug report) following information theory principles. GainRatio measures the importance of a feature f in a training dataset D by following Equation 2.

$$GainRatio(f, D) = \frac{IG(f, D)}{IV(f, D)} \quad (2)$$

$$IG(f, D) = H(D) - \sum_{v \in VAL(f,D)} \frac{|D_{f=v}|}{|D|} \times H(D_{f=v}) \quad (3)$$

$$IV(f,D) = -\sum_{v \in VAL(f,D)} \frac{|D_{f=v}|}{|D|} \times log_2 \frac{|D_{f=v}|}{|D|} \quad (4)$$

$$H(D) = -\sum_{c \in \{+,-\}} \frac{D_c}{D} \times log_2 \frac{D_c}{D} \quad (5)$$

The GainRatio score of a feature f given a dataset D is defined by normalizing the information gain (IG) score of the feature f (defined in Equation 3) with intrinsic value (IV) score of the feature f (defined in Equation 4) considering dataset D. Information gain is defined based on entropy (H) defined in Equation 3. In the equations, $VAL(f,D)$ corresponds to a set of values of attribute f in dataset D, $D_{f=v}$ corresponds to a subset of D whose attribute f is of value v, and D_c corresponds to a subset of D which belongs to class c (in our setting: configuration bugs or not). GainRatio feature selection technique sorts features based on their gain ratio scores and outputs the ranked list of features.

OneR. OneR is the abbreviation of "One Rule" which is proposed by Holte [15]. The basic idea of OneR is to build one rule for each feature in a training set of data instances (in our case: bug reports), and measure the prediction accuracies of the rules. The features are ranked based on the prediction accuracies of their associated rules. For more details of OneR, please refer to [15].

Relief. Relief is a feature selection algorithm for binary classification which performs n iterations [18]. In the i^{th} iteration, it randomly takes a data instance (i.e., a bug report) x, and finds instances which are closest to that of x (measured using Euclidean distance). We refer to the closest same-class instance as "near-hit" (denoted as hit), and the closest different-class instance as "near-miss" (denoted as $miss$). For each feature f, we update its feature score f_{score} by following Equation 6.

$$f_{score} = f_{score} - (x_f - hit_f)^2 + (x_f - miss_f)^2 \quad (6)$$

In the above equation, x_f, hit_f, and $miss_f$ correspond to the values of feature f for x, hit, and $miss$ data instances respectively. At the end of n iterations, each feature will have a feature score f_{score}, and Relief ranks the features according to their feature scores.

C. Feature List Composition

Algorithm 1 presents the details of the *feature list composition* step that is performed during the model construction phase of *EFSPredictor*. This step takes as input five ranked lists of features that are outputted by the five feature selection techniques ($RL = \{L^1, L^2, L^3, L^4, L^5\}$), a set of all features (FS), and the number of representative features to output. It outputs a set of representative features referred to as *ensembled feature subset* (EFS).

The algorithm goes through each ranked list, and for each feature, it assigns a weight to it based on its position in the list (Lines 9-14). The weight of a feature f in the i^{th} list (w_f^i) is computed by taking the reciprocal of its position in the list (Line 12). Next, for each feature f, the algorithm computes a score for it (s_f) by summing up the weights assigned to it for each of the ranked lists (Lines 15-17). Finally, the algorithm outputs the top n of the features with the highest scores (Lines 18-19). By default, we set n as 25% of the total number of features.

Algorithm 1 Feature List Composition

```
1: ComposeList
2: Input:
3: RL = {L^1, L^2, L^3, L^4, L^5}: Five ranked lists of features
4: FS: Set of all features
5: n: Output size
6: Output:
7: EFS: Ensembled feature subset
8: Method:
9: for all ranked list L^i ∈ RL do
10:    for all feature f ∈ FS do
11:       pos = position of f in L^i;
12:       w_f^i = 1/pos;
13:    end for
14: end for
15: for all feature f ∈ FS do
16:    s_f = Σ_i w_f^i
17: end for
18: EFS = Top n features with highest s_f scores
19: Output EFS
```

IV. Experiments And Results

In this section, we measure the effectiveness of *EFSPredictor* and compare it with the state-of-the-art approach by Xia et al. [38]. The experimental environment is an Intel(R) Core(TM) T6570 2.10 GHz desktop with 4GB RAM running Windows 7 (32-bit).

A. Experiment Setup

We evaluate *EFSPredictor* on 5 datasets containing a total of 3,203 bug reports from open source software projects (i.e., accumulo, activemq, camel, flume, and wicket), which are the same datasets used by Xia et al. [38] to evaluate their work. Each of the datasets contains a number of bug reports labeled as configuration bugs or not. Table II provides a summary information about the 5 datasets, including the project name (Project), the number of bug reports collected (# Bugs), the time period of the collected bug reports (Time), the number of configuration bug reports (# Confs), and the number of unique terms (i.e., words) in the bug reports (# Terms). To extract words from these bug reports, we use WVTool. WVTool is a Java library for statistical language modeling, which is used to create word vector representations of text documents. To reduce noise due to rarely used words, we remove words which appear less than 5 times.

To reduce training set selection bias, we perform 10-fold cross-validation. We randomly split each dataset into ten subsets (each with almost equal numbers of configuration and non-configuration bugs) and use 9 subsets for training and one subset for evaluation. The process is repeated ten times with a different subset used for evaluation in each iteration. In the end, an aggregate evaluation score computed across the ten iterations is reported. Cross validation is a standard and useful

evaluation setting, which has been widely used in past software engineering studies [22], [30], [37].

TABLE II. STATISTICS OF COLLECTED DATASETS

Project	#Bugs	Time	#Confs	#Terms
accumulo	181	2011. 10 - 2013. 06	33	227
activemq	175	2005. 12 - 2007. 12	29	327
camel	1,189	2007. 07 - 2013. 09	333	1,261
flume	279	2010. 07 - 2013. 05	83	341
wicket	1,379	2006. 11 - 2013. 09	46	1,340

B. Evaluation Metrics

The effectiveness of prediction models for two-class classification problems (e.g., predicting configuration bugs vs. non-configuration bugs) is typically evaluated using precision, recall and F1-score. They are defined based on four basic statistics: *TP* (true positive) represents the number of bug reports that are classified as configuration bug reports when they truly are configuration bug reports; *FP* (false positive) represents the number of bug reports that are classified as configuration bug reports when they actually are non-configuration bug reports; *FN* (false negative) represents the number of bug reports that are classified as non-configuration bug reports when they actually are configuration bug reports; *TN* (true negative) represents the number of bug reports that are classified as non-configuration bug reports and they truly are non-configuration bug reports. Precision, recall, and F1-score can be computed from these four statistics as follows:

- **Precision.** The proportion of bug reports correctly classified as configuration bug reports among those classified as configuration bug reports. It is defined as:

$$P = \frac{TP}{TP + FP} \tag{7}$$

- **Recall.** The proportion of bug reports correctly classified as configuration bug reports among all configuration bug reports. It is defined as:

$$R = \frac{TP}{TP + FN} \tag{8}$$

- **F1-score.** A harmonic mean of precision and recall. It is defined as:

$$F1 = \frac{2 \times P \times R}{P + R} \tag{9}$$

F1-score is a summary measure that evaluates if an increase in precision (recall) outweighs a loss in recall (precision). Since often there is a trade-off between precision and recall, in many past software engineering papers, F1-score is often used as the main evaluation metric – c.f., [27], [45]. In this paper, we also choose F1-score as the main evaluation metric.

C. Research Questions and Findings

We are interested to answer the following research questions:

RQ1: ***How effective is EFSPredictor? How much improvement can it achieve over the method proposed by Xia et al.?***

Motivation. We need to investigate the effectiveness of *EFSPredictor*, and compare it with the best performing variant of Xia et al.'s approach [38]. Answer to this research question would shed light to whether and to what extent *EFSPredictor* improves over the state-of-the-art approach.

Approach. To answer this research question, we compute precision, recall, and F1-score of *EFSPredictor* and the approach proposed by Xia et al. when they are applied to the 5 datasets listed in Table II. We perform 10-fold cross-validation and report the average evaluation scores across the ten iterations.

Result. Table III presents the precision, recall and F1-score of *EFSPredictor* and Xia et al.'s approach. From the table, we can note that the F1-scores of *EFSPredictor* for each of the five datasets are 0.73, 0.48, 0.54, 0.59, 0.50, which outperform Xia et al.'s approach by 20%, 7%, 10%, 10% and 20%, respectively. On average, *EFSPredictor* outperforms Xia et al.'s approach in terms of F1-score by 14%. The results show that the improvement that *EFSPredictor* achieves over Xia et al.'s approach is substantial. We can draw the conclusion that *EFSPredictor* performs better than Xia et al.'s approach.

EFSPredictor outperforms Xia et al.'s approach in predicting configuration bugs. The improvement in F1-score is 7-20% with an average of 14%.

RQ2: ***Does combining multiple feature selection techniques helps improve the effectiveness of EFSPredictor?***

Motivation. *EFSPredictor* is built on top of 5 feature selection techniques. We want to investigate whether *EFSPredictor* achieves better performance compared to its variant that only uses one feature selection method.

Approach. To answer this research question, we compute the F1-scores of standard *EFSPredictor* and five variants that only uses one of the 5 feature selection techniques, when they are applied to the 5 datasets listed in Table II.

Result. Table IV presents the F1-scores of *EFSPredictor* and its five variants when they are applied to the 5 datasets. From the table, we can see that *EFSPredictor* achieves the best F-measure for almost all the datasets with one exception. It losses to its variant that only uses Relief for one dataset (i.e., flume). Averaging across the five datasets, *EFSPredictor* outperforms its five variants. The results show that *EFSPredictor* approach to ensemble multiple features selection algorithms is effective.

In almost all cases and on average, composing multiple feature selection techniques improves the effectiveness of predicting configuration bugs.

RQ3: ***What is the impact of using different numbers of selected features on the effectiveness of EFSPredictor?***

Motivation. By default, we set the number of selected features (i.e., parameter n in Algorithm 1) as 25% of the total number of features. We investigate whether different number of selected features impacts the effectiveness of *EFSPredictor*.

Approach. To answer this research question, we vary the number of selected features from 5% to 100% (with a step of 5%) of the total number of features and compute the corresponding F1-scores of *EFSPredictor* for the different

TABLE III. PRECISION, RECALL AND F1-SCORE OF EFSPREDICTOR AND XIA ET AL.'S APPROACH

		accumulo	activemq	camel	flume	wicker
Precision	Xia et al.'s Approach	0.52	0.35	0.58	0.49	0.32
	EFSPredictor	0.66	0.48	0.61	0.57	0.42
Recall	Xia et al.'s Approach	0.73	0.62	0.44	0.60	0.65
	EFSPredictor	0.82	0.48	0.48	0.63	0.63
F1-score	Xia et al.'s Approach	0.61	0.45	0.49	0.54	0.42
	EFSPredictor	0.73	0.48	0.54	0.59	0.50
	Improvement	0.20	0.07	0.10	0.10	0.20

TABLE IV. F1-SCORE OF *EFSPredictor* AND ITS 5 VARIANTS

	Relief	ChiSquare	Filtered	GainRatio	OneR	*EFSPredictor*
accumulo	0.56	0.61	0.61	0.61	0.58	0.62
activemq	0.37	0.44	0.44	0.44	0.40	0.44
camel	0.49	0.52	0.52	0.52	0.52	0.52
flume	0.57	0.55	0.55	0.55	0.54	0.56
wicker	0.35	0.44	0.44	0.44	0.34	0.45

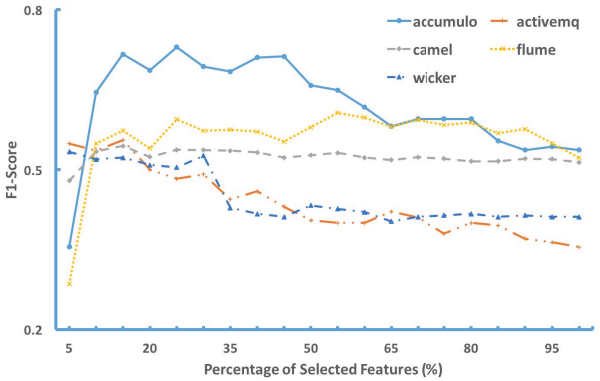

Fig. 2. Effectiveness of *EFSPredictor* for Different Numbers of Features

numbers of features. We plot the resultant F1-scores for each of the 5 datasets in Figure 2.

Result. From Figure 2, we can notice a general trend: F1-score increases when we increase the number of features until a certain point, after which, the F1-score starts to decrease. When we use too few features, these features are not able to fully characterize the difference between configuration and non-configuration bugs. On the other hand, when we use too many features, some features capture noise reduces the effectiveness of *EFSPredictor*. For most datasets, *EFSPredictor* achieves the best F1-score when the number of features are between 15% to 30% of the total number of features.

The effectiveness of EFSPredictor initially improves when we increase the number of features until a certain point, beyond which the effectiveness of EFSPredictor starts to decrease. For most datasets, the best number of features is between 15% to 30% of the total number of features.

D. Threats to Validity

Threats to internal validity relate to common errors in our experiments. We have checked our experiments and implementations, still there could be errors that we did not notice.

Threats to external validity relate to the generalizability of our results. We have analyzed a total of 3,203 bug reports from 5 open source software projects. In the future, we plan to reduce this threat further by analyzing more bug reports from additional open source software projects.

Threats to construct validity refer to the suitability of our evaluation metrics. We use precision, recall, and F1-scores as the evaluation metrics. These metrics are standard and widely used to evaluate the effectiveness of a prediction technique, c.f., [25], [36]. Thus, we believe there is little threat to construct validity.

V. RELATED WORK

In this section, we briefly review some previous studies on configuration bug reports, bug report categorization, and classification in software engineering. Due to space limitation, the survey here is by no means complete.

A. Studies on Configuration Bugs

Xia et al. propose a text mining technique to predict if a bug report is due to a misconfiguration [38]. Our work builds upon their work by integrating multiple feature selection algorithms, i.e., ChiSquare, Filter, GainRatio, OneR, and Relief, to produce a set of representative features. The experiments results show that our proposed approach improves Xia et al.'s approach by a substantial margin.

Aside from the closest related work by Xia et al., there are a number of other studies on configuration bugs. Wang et al. present PeerPressure, which leverages statistical methods to diagnose the root-cause of configuration errors [34]. Yin et al. perform an empirical study on configuration bugs in five software systems, and they find that most configuration bugs are due to incorrect parameter setting [43]. Zhang and Ernst use static analysis, dynamic profiling, and statistical analysis to detect configuration issues [46]. Arshad et al. study configuration bugs from two open source projects, i.e., GlassFish and JBoss, and they characterize the configuration bugs from four dimensions, i.e., problem-type, problem-time, problem-manifestation, and problem-culprit [8]. Xu et al. propose a tool SPEX to expose misconfiguration vulnerabilities, and detect

error-prone configuration design in software systems [41]. Attariyan and Flinn propose AutoBash to diagnose configuration bugs by leveraging the causal dependencies of test case executions [9]. Our study complements the above studies and consider a different problem – we predict whether a bug report is caused due to misconfigurations.

B. Studies on Bug Categorization

There have been a number of studies that automatically group bugs into various categories. Zanetti et al. propose the usage of social network to predict valid bugs in open source projects [44]. They construct a social network by considering the reporters, the assigned developers, and the developers in the CC list of bug reports, and extract a number of features from the network, such as closeness centrality, betweenness centrality, and LCC, to differentiate valid from invalid bug reports. Antoniol et al. propose a classification approach that can predict if an issue report is a bug report or a feature request [7]. Their approach is extended by Zhou et al. [50]; different from Antoniol et al.'s work that only uses text features from issue reports, Zhou et al. use both text features and non-text features (i.e., values of reporter, assignee, priority, severity, and component fields) extracted from bug reports to classify issue reports into bug reports and feature requests more accurately. Herzig et al. perform an empirical study on the impact of misclassification on bug prediction [14]. They propose six fine-grained categories of bug reports, i.e., bug, enhancement, improvement, documentation, refactoring, and others. Kochhar et al. extend Herzirg et al.'s work by proposing an automated approach to predicts if a bug report needs to be reclassified and its reclassified category [19].

Gegick et al. use text mining techniques to identify security bug reports [12]. Thung et al. propose an approach to automatically categorize bug reports into 3 labels: a control bug, data flow bug, or a structural bug [29]. Huang et al. predict the impact of bugs by analyzing the textual contents extracted from bug reports [16]. They classify a bug into five categories: reliability, capability, integrity, usability, and requirements. Xia et al. propose a technique to categorize bugs based on their fault triggering conditions [40]. Xia et al. also propose *ELBlocker* to identify blocking bugs [39]. *ELBlocker* first randomly divides training data into multiple disjoint sets, and for each disjoint set, it builds a classifier; next, it combines these multiple classifiers, and automatically determines an appropriate imbalance decision boundary to differentiate blocking bugs from non-blocking bugs.

Our work is orthogonal to the above studies; we focus on identifying configuration bugs, which is a different problem than those investigated in the above studies.

C. Studies on Classification in Software Engineering

There have been many other studies that employ classification algorithms to solve various software engineering problems. We highlight some of them below. Tian et al. propose an automated approach based on machine learning that recommends a priority level to a bug report based on multiple influencing factors [31]. Menzies et al. presents an automated method named SEVERIS (SEVERity ISsue assessment), which assists test engineers in assigning severity levels to defect reports [22]. Lamkanfi et al. extend Menzies et al.'s work by proposing another text mining approach which analyzes textual descriptions of bug reports and predict *coarse-grained* bug severity levels with more accuracy [20]. Zhang et al. investigate 7 composite algorithms, which integrate multiple machine learning classifiers, to improve cross-project defect prediction [47].

VI. Conclusion and Future Work

In order to predict configuration bugs, we propose an automated tool *EFSPredictor* which combines multiple feature selection techniques and a classification algorithm to build a statistical prediction model from historical bug reports. We investigate 5 feature selection techniques, namely ChiSquare, Filtered, GainRatio, OneR, Relief, and evaluate *EFSPredictor* on 5 open source projects including a total of 3,203 bug reports. The experiment results show that our proposed tool *EFSPredictor* performs much better than the state-of-the-art approach by Xia et al. Across the 5 projects, the F1-scores of *EFSPredictor* outperform those of Xia et al's approach by 20%, 7%, 10%, 10% and 20%, with an average of 14%.

In the future, we intend to investigate more bug reports from more projects to reduce the threats to external validity further. We also plan to design additional solutions that can help boost the effectiveness of *EFSPredictor* further.

Acknowledgment

This research was partially supported by China Knowledge Centre for Engineering Sciences and Technology (No. CKCEST-2014-1-5), National Key Technology R&D Program of the Ministry of Science and Technology of China (No. 2015BAH17F01), and the Fundamental Research Funds for the Central Universities.

References

[1] Accumulo. *https://issues.apache.org/jira/browse/ACCUMULO*.

[2] Airbus confirms software configuration error caused plane crash. *http://arstechnica.com/information-technology/2015/06/airbus-confirms-software-configuration-error-caused-plane-crash/*.

[3] Amq. *https://issues.apache.org/jira/browse/AMQ*.

[4] Camel. *https://issues.apache.org/jira/browse/CAMEL*.

[5] Flume. *https://issues.apache.org/jira/browse/FLUME*.

[6] Wicket. *https://issues.apache.org/jira/browse/WICKET*.

[7] G. Antoniol, K. Ayari, M. D. Penta, F. Khomh, and Y. Guéhéneuc. Is it a bug or an enhancement?: a text-based approach to classify change requests. In *Proceedings of the 2008 conference of the Centre for Advanced Studies on Collaborative Research, October 27-30, 2008, Richmond Hill, Ontario, Canada*, page 23, 2008.

[8] F. Arshad, R. J. Krause, S. Bagchi, et al. Characterizing configuration problems in java ee application servers: An empirical study with glassfish and jboss. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pages 198–207. IEEE, 2013.

[9] M. Attariyan and J. Flinn. Using causality to diagnose configuration bugs. In *USENIX Annual Technical Conference*, pages 281–286, 2008.

[10] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[11] Y.-T. Chen and M. C. Chen. Using chi-square statistics to measure similarities for text categorization. *Expert Systems with Applications*, 38(4):3085–3090, 2011.

[12] M. Gegick, P. Rotella, and T. Xie. Identifying security bug reports via text mining: An industrial case study. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 11–20. IEEE, 2010.

[13] M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[14] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401. IEEE Press, 2013.

[15] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

[16] L. Huang, V. Ng, I. Persing, R. Geng, X. Bai, and J. Tian. Autoodc: Automated generation of orthogonal defect classifications. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 412–415. IEEE, 2011.

[17] X. Jin, A. Xu, R. Bie, and P. Guo. Machine learning techniques and chi-square feature selection for cancer classification using sage gene expression profiles. In *Data Mining for Biomedical Applications*, pages 106–115. Springer, 2006.

[18] K. Kira and L. A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *AAAI*, volume 2, pages 129–134, 1992.

[19] P. S. Kochhar, F. Thung, and D. Lo. Automatic fine-grained issue report reclassification. In *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*, pages 126–135. IEEE, 2014.

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

[21] Q. Lu, L. Zhu, L. Bass, X. Xu, Z. Li, and H. Wada. Cloud api issues: an empirical study and impact. In *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, pages 23–32. ACM, 2013.

[22] T. Menzies and A. Marcus. Automated severity assessment of software defect reports. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 346–355. IEEE, 2008.

[23] D. W. Opitz. Feature selection for ensembles. In *AAAI/IAAI*, pages 379–384, 1999.

[24] M. Porter. An algorithm for suffix stripping. *Program*, 1980.

[25] N. Prasasti, M. Okada, K. Kanamori, and H. Ohwada. Customer lifetime value and defection possibility prediction model using machine learning: An application to a cloud-based software company. In *Intelligent Information and Database Systems*, pages 62–71. Springer, 2014.

[26] A. Rabkin and R. H. Katz. How hadoop clusters break. *Software, IEEE*, 30(4):88–94, 2013.

[27] F. Rahman, D. Posnett, and P. Devanbu. Recalling the imprecision of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 61. ACM, 2012.

[28] Y. Saeys, T. Abeel, and Y. Van de Peer. Robust feature selection using ensemble feature selection techniques. In *Machine learning and knowledge discovery in databases*, pages 313–325. Springer, 2008.

[29] F. Thung, D. Lo, and L. Jiang. Automatic defect categorization. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 205–214. IEEE, 2012.

[30] Y. Tian, J. Lawall, and D. Lo. Identifying linux bug fixing patches. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 386–396. IEEE, 2012.

[31] Y. Tian, D. Lo, X. Xia, and C. Sun. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering*, pages 1–30, 2014.

[32] A. Tsymbal, S. Puuronen, and D. W. Patterson. Ensemble feature selection with the simple bayesian classification. *Information Fusion*, 4(2):87–100, 2003.

[33] A. K. Uysal and S. Gunal. A novel probabilistic feature selection method for text classification. *Knowledge-Based Systems*, 36:226–235, 2012.

[34] H. J. Wang, J. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Peerpressure: A statistical method for automatic misconfiguration troubleshooting, 2003.

[35] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Automatic misconfiguration troubleshooting with peerpressure. In *OSDI*, volume 4, pages 245–257, 2004.

[36] B.-J. M. Webb-Robertson, M. M. Matzke, S. Datta, S. H. Payne, J. Kang, L. M. Bramer, C. D. Nicora, A. K. Shukla, T. O. Metz, K. D. Rodland, et al. Bayesian proteoform modeling improves protein quantification of global proteomic measurements. *Molecular & Cellular Proteomics*, 13(12):3639–3646, 2014.

[37] X. Xia, Y. Feng, D. Lo, Z. Chen, and X. Wang. Towards more accurate multi-label software behavior learning. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 134–143. IEEE, 2014.

[38] X. Xia, D. Lo, W. Qiu, X. Wang, and B. Zhou. Automated configuration bug report prediction using text mining. In *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, pages 107–116. IEEE, 2014.

[39] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang. Elblocker: Predicting blocking bugs with ensemble imbalance learning. *Information and Software Technology*, 61:93–106, 2015.

[40] X. Xia, D. Lo, X. Wang, and B. Zhou. Automatic defect categorization based on fault triggering conditions. In *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*, pages 39–48. IEEE, 2014.

[41] T. Xu, J. Zhang, P. Huang, J. Zheng, T. Sheng, D. Yuan, Y. Zhou, and S. Pasupathy. Do not blame users for misconfigurations. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 244–259. ACM, 2013.

[42] J. Yang, Y. Liu, X. Zhu, Z. Liu, and X. Zhang. A new feature selection based on comprehensive measurement both in inter-category and intra-category for text categorization. *Information Processing & Management*, 48(4):741–754, 2012.

[43] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy. An empirical study on configuration errors in commercial and open source systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 159–172. ACM, 2011.

[44] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer. Categorizing bugs with social networks: a case study on four open source software communities. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1032–1041. IEEE Press, 2013.

[45] H. Zhang, L. Gong, and S. Versteeg. Predicting bug-fixing time: an empirical study of commercial software projects. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1042–1051. IEEE Press, 2013.

[46] S. Zhang and M. D. Ernst. Automated diagnosis of software configuration errors. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 312–321. IEEE Press, 2013.

[47] Y. Zhang, D. Lo, X. Xia, and J. Sun. An empirical study of classifier combination for cross-project defect prediction. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*. IEEE, 2015.

[48] Z. Zhao, L. Wang, H. Liu, and J. Ye. On similarity preserving feature selection. *Knowledge and Data Engineering, IEEE Transactions on*, 25(3):619–632, 2013.

[49] W. Zheng, R. Bianchini, and T. D. Nguyen. Automatic configuration of internet services. *ACM SIGOPS Operating Systems Review*, 41(3):219–229, 2007.

[50] Y. Zhou, Y. Tong, R. Gu, and H. C. Gall. Combining text mining and data mining for bug report classification. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*, pages 311–320, 2014.