EFSPredictor: Predicting Configuration Bugs With Ensemble Feature Selection

Bowen Xu, David Lo, Xin Xia, Ashish Sureka, and Shanping Li
max_xbw@zju.edu.cn

Zhejiang University, Hangzhou, China
SMU, Singapore
IIIT Delhi, India
**Introduction-News1**

Airbus confirms software configuration error caused plane crash

Airbus A400M flight recorder data confirms "quality issue" in setup caused failure.

by Sean Gallagher - Jun 2, 2015 12:36am CST

An executive of Airbus Group has confirmed that the crash of an Airbus A400M military transport was caused by a faulty software configuration. Marwan Lahoud, chief marketing and strategy officer for Airbus, told the German newspaper *Handelsblatt on Friday* that there was a "quality issue in the final assembly" of the components of the aircraft engine.

As *Ars reported on May 19*, Airbus had issued a warning to its military customers about a potential software problem in the engine control software for the A400M. The release of the exact cause of the crash, however, had been delayed because a Spanish magistrate placed the flight data recorders from the aircraft under seal. Airbus has since been able to obtain the flight data, which Lahoud said confirms that the engine control software had been improperly configured during the installation of the engines on the ill-fated aircraft.

"The black boxes attest to that," Lahoud told Handelsblatt. "There are no structural defects, but we have a serious quality problem in the final assembly." The error was not in the code itself, but in configuration settings programmed into the electronic control unit (ECU) of the engines.

If it holds up, the finding means that Airbus will be able to avoid a complete redesign of the A400M's engines—the largest turboprop engines ever manufactured in Europe. The Spanish aircraft was being used by the military there to conduct pre-acceptance testing on the aircraft, which has already suffered from numerous delays and cost overruns.

---

**Fig.1. Example1 : Airbus confirms software configuration error caused plane crash**
Introduction - News2

Configuration errors are one of the dominant causes of software failures and it could cause huge financial losses in a software system!

Fig.2. Example2: ‘Configuration Error’ Blamed for AWS Outage
Introduction-Configuration Bugs

-What is Configuration Bugs?
Wrong configuration settings in software system.

-Root caused:
An incompatibility between different architectures and platforms.

-Fixing:
Modify its maven configuration file, i.e., pom.xml, by changing the value of the “needarch” parameter from “true” to “x86 64”.

Fig.3. Bug Report of accumulo with BugID=1560.
Fig.4. The Patch File for the Bug in Fig 1.
Introduction-Configuration Bugs

-What is Configuration Bugs?
Wrong configuration settings in software system.

-Observations:
• The natural-language description of a bug report provides information to indicate whether a bug is a configuration bug.
• Some terms in a bug report are good indicators to identify whether it is a configuration bug, while some other terms are noise.

The above observations tell us:
We can use the natural-language description of bug reports to identify configuration bugs, and selecting good terms (indicators) from bug reports could help to improve classification performance.

Fig.5. Bug Report of accumulo with BugID=1560.
Motivation

To improve performance of prediction, many previous works focus on using different classifiers. They use single feature selection method or even without feature selection!
Motivation

To improve performance of prediction, many previous works focus on using different classifiers. They use single feature selection method or even without feature selection!
Motivation

Based on our observations(slide5) - **Some terms in a bug report are good indicators to identify whether it is a configuration bug, while some other terms are noise**, thus we focus our work on a different insights that how to select the most representative feature to reduce noise to improve performance of configuration bug prediction.
Motivation - Ensemble Feature Selection

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. 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.
Our Proposed Approach-Framework

Fig.8. Overall Framework of EFSPredictor.
## Our Proposed Approach

- Feature Selection Method 1 - ChiSquare

<table>
<thead>
<tr>
<th>Term(Feature) \ class</th>
<th>Configuration Bug</th>
<th>Non-Configuration Bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include “Term_i”</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Exclude “Term_i”</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Measure a term \( t \) is associated with the document type \( c \) :

\[
\chi^2(t, c) = \frac{N \times (A \times D - B \times C)^2}{(A + C) \times (B + D \times (A + B) \times (C + D))}
\]  

(1)

\( N \) is the total number of training bug reports.
Our Proposed Approach

- Feature Selection Method 2 - Filter

Training Dataset → Resample → Class Balance Dataset → ChiSquare Feature Selection → Feature Ranklist

randomly duplicating some data instances (in our case: bug reports) in the minority class (i.e., class with the least members)
Our Proposed Approach

- Feature Selection Method 3 - GainRatio

Information Gain

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

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

Intrinsic Value

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

Information gain ratio is a ratio of information gain to the intrinsic information. It is used to estimate the importance of a feature is to measure how much information the feature can bring to classify a bug report following information theory principles.
Our Proposed Approach

-Feature Selection Method 4 - OneR (OneRule)

OneR is the abbreviation of “One Rule” which is proposed by Holte. The basic idea of OneR is to build one rule for each feature in a training set of bug reports, and measure the prediction accuracies of the rules. The features are ranked based on the prediction accuracies of their associated rules.

pseudo-code:
1. FOR (each $Feature_i$ : Features)
2.  FOR (each $Value_j$ of $Feature_i$){
3.    calc_frequency(each class);
4.    search most frequent class;
5.    build_rule();
6.  }
7. calc Error_Rate;
8. Return minErrorRate;
Our Proposed Approach

-Feature Selection Method 5 - Relief

Relief is a feature selection algorithm for binary classification which performs $n$ iterations [18]. In the $i$th iteration, it randomly takes a bug report $x$, and finds other bug reports 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.

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

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.
Our Proposed Approach

- Feature List Composition

**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 \( L^i \) \( \in RL \) do
10: for all feature \( f \) \( \in FS \) do
11: \( pos = \text{position of } f \text{ in } L^i \);
12: \( w_f^i = \frac{1}{\text{pos}} \);
13: end for
14: end for
15: for all feature \( f \) \( \in FS \) do
16: \( s_f = \sum_i w_f^i \)
17: end for
18: \( EFS = \text{Top } n \text{ features with highest } s_f \text{ scores} \)
19: Output \( EFS \)

**Table. Example for Ensemble Feature (n=1)**

<table>
<thead>
<tr>
<th>No.</th>
<th>Feature Selection Method</th>
<th>( L^1 )</th>
<th>( L^2 )</th>
<th>( L^3 )</th>
<th>( L^4 )</th>
<th>( L^5 )</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ChiSquare</td>
<td>( f_1 )</td>
<td>( f_2 )</td>
<td>( f_3 )</td>
<td>( f_4 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Filter</td>
<td>( f_3 )</td>
<td>( f_2 )</td>
<td>( f_4 )</td>
<td>( f_1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>GainRatio</td>
<td>( f_4 )</td>
<td>( f_2 )</td>
<td>( f_1 )</td>
<td>( f_3 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>OneR</td>
<td>( f_1 )</td>
<td>( f_3 )</td>
<td>( f_2 )</td>
<td>( f_4 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Relief</td>
<td>( f_1 )</td>
<td>( f_2 )</td>
<td>( f_4 )</td>
<td>( f_3 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>Ensemble</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>1 1/4 1/3 1 1</td>
<td>43/12</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>1/2 1/2 1/2 1/3 1/2</td>
<td>28/12</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>1/3 1 1/4 1/2 1/4</td>
<td>28/12</td>
</tr>
<tr>
<td>( f_4 )</td>
<td>1/4 1/3 1 1/4 1/3</td>
<td>22/12</td>
</tr>
</tbody>
</table>
Experiments and Results

- Dataset (same as previous work)

<table>
<thead>
<tr>
<th>Project</th>
<th>#Bugs</th>
<th>Time</th>
<th>#Conf</th>
<th>#Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>accumulo</td>
<td>181</td>
<td>2011.10 - 2013.06</td>
<td>33</td>
<td>227</td>
</tr>
<tr>
<td>activemq</td>
<td>175</td>
<td>2005.12 - 2007.12</td>
<td>29</td>
<td>327</td>
</tr>
<tr>
<td>camel</td>
<td>1,189</td>
<td>2007.07 - 2013.09</td>
<td>333</td>
<td>1,261</td>
</tr>
<tr>
<td>flume</td>
<td>279</td>
<td>2010.07 - 2013.05</td>
<td>83</td>
<td>341</td>
</tr>
<tr>
<td>wicket</td>
<td>1,379</td>
<td>2006.11 - 2013.09</td>
<td>46</td>
<td>1,340</td>
</tr>
</tbody>
</table>
Experiments and Results

-Metrics (same as previous work)

<table>
<thead>
<tr>
<th></th>
<th>Configuration Bug</th>
<th>Non-configuration Bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classified as ‘Configuration Bug’</td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>Classified as ‘Non-Configuration Bug’</td>
<td>FN</td>
<td>TN</td>
</tr>
</tbody>
</table>
Experiments and Results

-Metrics (same as previous work)

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

\[
P = \frac{TP}{TP + FP}
\]

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

\[
R = \frac{TP}{TP + FN}
\]

• **F1-score (main evaluation metric).** A harmonic mean of precision and recall. It is defined as:

\[
F1 = \frac{2 \times P \times R}{P + R}
\]
Experiments and Results

- Research Questions 1:

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

<table>
<thead>
<tr>
<th>TABLE III. PRECISION, RECALL AND F1-SCORE OF EFSPREDICTOR AND XIA ET AL.’S APPROACH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
</tr>
<tr>
<td>Xia et al.'s Approach</td>
</tr>
<tr>
<td>EFSPredictor</td>
</tr>
<tr>
<td>Recall</td>
</tr>
<tr>
<td>Xia et al.'s Approach</td>
</tr>
<tr>
<td>EFSPredictor</td>
</tr>
<tr>
<td>F1-score</td>
</tr>
<tr>
<td>Xia et al.'s Approach</td>
</tr>
<tr>
<td>EFSPredictor</td>
</tr>
<tr>
<td>Improvement</td>
</tr>
</tbody>
</table>

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

- Research Questions 2:

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

<table>
<thead>
<tr>
<th>TABLE IV.</th>
<th>F1-score of EFSPredictor and Its 5 Variants</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relief</td>
</tr>
<tr>
<td>accumulo</td>
<td>0.56</td>
</tr>
<tr>
<td>activemq</td>
<td>0.37</td>
</tr>
<tr>
<td>camel</td>
<td>0.49</td>
</tr>
<tr>
<td>flume</td>
<td>0.57</td>
</tr>
<tr>
<td>wicker</td>
<td>0.35</td>
</tr>
</tbody>
</table>

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

- Research Questions 3:

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

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.

Fig. 2. Effectiveness of EFSPredictor for Different Numbers of Features
Conclusion and Future Work

• Conclusion
  - 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.

• Future Work
  - 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.
  - Our proposed approach is general and we have proved that it is able to achieve better performance than previous work on predict configuration bug. Thus, we believe that it is a good starting to apply our approach on predicting other types of bug to make a deeper exploration.
Thank you and QA!