A linear classifier based approach for identifying security requirements in open source software development

Wentao Wang \textsuperscript{a,1}, Kavya Reddy Mahakala \textsuperscript{a}, Arushi Gupta \textsuperscript{a}, Nesrin Hussein \textsuperscript{a}, Yinglin Wang \textsuperscript{b}

\textsuperscript{a} Department of Electrical Engineering and Computer Science, University of Cincinnati, USA
\textsuperscript{b} Department of Computer Science and Technology, Shanghai University of Finance and Economics, China

\textbf{ARTICLE INFO}

MSC:
00-00 Just-in-time requirements engineering
99-00 Security requirements identification
Linear classifier
Regression model,

\textbf{ABSTRACT}

There are several security requirements identification methods proposed by researchers in up-front requirements engineering (RE). However, in open source software (OSS) projects, developers use lightweight representation and refine requirements frequently by writing comments. They also tend to discuss security aspect in comments by providing code snippets, attachments, and external resource links. Since most security requirements identification methods in up-front RE are based on textual information retrieval techniques, these methods are not suitable for OSS projects or just-in-time RE. In this study, we proposed a linear based approach to identify security requirements. It first uses logistic regression models (RMs) to calculate feature values for requirements in OSS project. Then it uses the linear combination of all feature values to classify security and non-security requirements. Our results show that compares to single RMs, our approach can achieve higher recall and precision.

1. Introduction

Security refers to a class of non-functional requirements (NFRs) related to system confidentiality, integrity, and availability [1]. It is a fundamental requirement and research direction for the Internet of Things (IoT) [2,3], a key component of Industry 4.0 [4,5].

Experience indicates that thinking about security early in the software life cycle can help address security problems, such as reducing the defect density ratio (that is, number of bugs per thousand lines of code) [6]. Early detection of security requirements enables engineers to incorporate security into the initial architectural design [7]. However, differentiating security requirements from non-security requirements can be nebulous even to the humans, much less the automated detection methods. Additionally, the trend of agile software engineering and Just-In-Time requirements engineering (JIT RE) [8,9] increases the complexity and difficulty of this problem.

A considerable number of studies have been done on detecting security requirements [10,11]. However, they are labor intensive. Clelland-Huang et al. [7] proposed NFR-classifier, an automated approach based on information retrieval methods for detecting and classifying NFRs. Mahmoud and Williams [12] proposed another automated approach which exploits the textual semantics of software functional requirements (FRs) to infer potential quality constraints enforced in the system. Their research demonstrated that those methods can achieve high accuracy in traditional or up-front RE. As a precursor to our work, we applied these methods to three open source software (OSS) projects, and the results show that none of them achieve similar performance as in up-front RE projects. Furthermore, the goal of those research is providing general methods for all NFRs, not specific for security requirements. Therefore, security specific semantic information, such as CVE (Common Vulnerabilities and Exposures)\textsuperscript{1} are not included in their approaches.

Motivated by the above observations this paper, we propose a novel and efficient approach for identifying security requirements in OSS development. In this paper, we have following contributions: (1) building models with metrics related to requirements complexity and external resources; and (2) finding an optimal way to integrate all models with NFR-classifier. Our results show that the enhanced approach can achieve average 92.31\% recall and 62.94\% precision in three OSS projects.

The remainder of this paper is organized as follows. Section 2 reviews related work in security requirements identification. In Section 3, we describe the rationale of datasets and metrics selection. Section 4 introduces the linear classifier based approach and Section 5 assesses the performance of our approach. Section 6 concludes the paper and discusses prospects for future work.

\footnotesize{\textsuperscript{1}https://cve.mitre.org

https://doi.org/10.1016/j.jii.2018.11.001

Received 11 November 2017; Received in revised form 17 August 2018; Accepted 1 November 2018

Available online 05 November 2018

2452-414X/ © 2018 Elsevier Inc. All rights reserved.
2. Background and related work

Existing researches applied different machine learning algorithms to identify and classify NFRs. In this section, we introduce previous researches based on different machine learning algorithms and their performance on security requirements identification.

Cleland-Huang et al. [7] described NFR classifier, an automated method based on the assumption that certain keywords can be used to distinguish different types of NFRs. A set of pre-defined requirements was used to derive different sets of weighted keywords called indicator terms for different NFRs. Indicator terms were then used as queries to retrieve NFRs from various software artifacts. An industrial case study was conducted to evaluate the proposed approach. The result showed that NFR classifier can recall 82.8% security requirements. At the same time, the result also showed a very high ratio of false positive (7.1% precision). The proposed approach benefit from the textual features of requirements, and ignore other requirements features, e.g., stakeholders information and logical information. We conjecture that contextual and lexical features are complementary with the textual feature on indicating security requirements and thus classifiers build with contextual information may be possible to integrate with NFR classifier and improve the performance of the proposed approach.

Unlike NFR classifier which derives indicator terms from a pre-labeled set of requirements, Mahmoud and Williams [12] proposed a method which discovers and assesses the query terms by using un-labeled datasets. The proposed approach first grouped terms of requirements into different clusters based on their semantic similarity. Then average pairwise similarity between clusters and NFR categories (e.g., accessibility, interoperability, and security) were calculated. A cluster was only assigned to the NFR category with highest average pairwise similarity. Requirements then were automatically classified into different NFR categories based on their semantic similarity to the clusters of terms representing individual NFRs. This approach adopts crisp clustering to ensure clearcut results, i.e., one term can belong to one NFR only. However, the same term can indicate different things. For instance, “denial-of-service” is related to both security and availability. Thus, crisp clustering may cause the in-completed set of representative terms. Furthermore, it may harm the performance of NFR identification.

Kurtanović and Maalej [13] applied Support Vector Machine (SVM) with 11 different lexical features, such as text length, fractions of nouns and verbs, and sentence syntax tree height to build two classifiers. The binary classifier was used to distinguish functional requirements (FRs) and NFRs. The multi-class NFRs classifier is worked on the result of the binary classifier to identify usability, security, operational and performance NFRs. The proposed approach was applied to two datasets, and results showed that this method can find 90.1% security requirements and filter out most non-security requirements (precision 90.0%). However, unlike datasets used in this research, requirements specifications in OSS projects tend to be organized by functionality, with NFR scattered widely across multiple documents [7]. Therefore, there is no clear boundary between FRs and NFRs. Moreover, the requirements stored in issue tracking systems are unstructured and seldom obey grammar and punctuation rules [14]. Thus, this approach is not suitable for OSS projects.

Munaiah et al. [15] proposed a un-supervised approach for detecting security requirements. They trained their One-Class Support Vector Machine with the Common Weakness Enumeration (CWE)², a formal list of software weakness types intended to serve as the common language for describing software security weaknesses in architecture, design, or code. The assumption authors conjectured is that the language used to describe security requirements and that used to describe weaknesses overlap thus it could be a good candidate to train the classifier. Like Gibiec et al. [16] pointed out, term mismatch problem is a common occurrence in software artifacts. Additionally, specific terms contained in requirements which indicate concepts like security and dependability are domain/application dependent [Anonymous, 2017]. Therefore, the performance of this approach is lower than other approaches we discussed in the earlier paragraphs of this section (i.e., 70.48% recall and precision level is 67.35%).

To the best of our understanding, there is no approach using features related to requirements complexity and external resources to identify security requirements. However, those features are widely applied in other research domains. Such as, software quality prediction [17], software complexity evaluation [18], and vulnerability detection [19]. Intuitively, these features could be used as metrics to train a binary security requirements classifier.

3. Datasets and metrics selection

The first challenge in building security requirements identification classifier is to find candidate metrics. We manually analyzed three web-based OSS projects, we two types of metrics are suitable to distinguish security requirements from non-security requirements. In this section, we first introduce the three datasets we used to evaluate our approach and then discuss metrics selection.

3.1. Datasets and answer sets

In our study, we analyzed the requirements of three OSS projects³: Apache Axis2/Java (Axis2)², Drools¹, and GeoServer⁶. We selected these projects as the subject systems due to three reasons. First, all of them are successful and long-lived projects. Second, all resources including requirements and source code are available. At last, all three projects are web-based systems and security is one of the core aspects of these projects, so identifying and realizing security requirements are important tasks for developers of these three projects.

These projects come from different application domains, and all of them are written in Java. Axis2 is a web services engine funded by Apache Software Foundation since August 2004. The newest version of Axis2 (1.7.4) was released in December 2016. Drools is a business rule management system developed by Red Hat. The latest stable release of Drools is 6.5.0, which was published in December 2016. GeoServer is a geographic system which allows users to edit and share geospatial information. The current stable version 2.11.0 was published in March 2017. Table 1 shows some basic information about these three projects.

We manually created the answer sets ² via a two-step analysis. In the first step, two pre-defined sets of security indicator terms were used to retrieve security requirements candidates in each project. One is called the “general indicator terms” [] which is extracted from [1] and [7]. For another one which is called “project dependent indicator terms”, we manually went through the official websites of three projects, collected all security related artifacts, and then extracted project dependent indicator terms from those security related artifacts. For instance, in Axis2, developers describe their transport framework as “We have a clean and simple abstraction for integrating and using transports (i.e., senders and listeners for SOAP over various protocols such as SMTP, FTP, message-oriented middleware, etc), and the core of the engine is completely transport-independent.”. According to SOAP (simple object access protocol) definition from W3C⁷, it is security related. Therefore,

³ We share our collected data and analysis results in [20] for validation and replication purpose.
⁴ http://axis.apache.org/axis2/java/core/
⁵ https://www.drools.org
⁶ http://geoserver.org
⁷ Our answer sets can also be found in [20].
⁸ https://www.w3.org/TR/soap/
we added it into Axis2’s project dependent indicator terms set. Table 2 shows “project dependent indicator terms” for three projects.

To evaluate the effectiveness of our indicator terms, we design a pilot study. The research question in this pilot study is “can indicator terms displayed in Table 2 retrieve all security requirements (i.e., achieve 100% recall)?”. First, from each dataset, we randomly choose 10% requirements and ask an analyst who has 1 year research experience on both these three projects and security to manually go through all of selected requirements and classify them into security or non-security requirements. After this step A keywords searching is then applied to the selected requirements. That is, if a requirement contains at least one indicator term, we mark it as security requirements. The searching results are showed in Table 3. Indicator terms can find 100%, 100%, and 96.30% security requirements. Two GeoServer security requirements that do not retrieved by the keywords searching shared words “SQL”. Therefore we add it to Table 2 as a new indicator for GeoServer project (red part). Results of pilot study demonstrated that keywords searching can retrieve all security requirements, we therefore apply it to help us to reduce manual effort.

In the second step, two analysts classified all candidates as security or non-security requirements independently. If two analysts have the same decision on one issue, we mark it as security/non-security requirement directly. Otherwise, two analysts have a discussion and make the final decision together. According to Cohen’s Kappa analysis, two analysts achieved almost perfect agreement [21] (i.e., 0.89, 0.73, and 0.97 on Axis2, Drools, and GeoServer, respectively).

### 3.2. Metrics selection

In this section, we classify our metrics into two categories: complexity and external resources. For each metric, we not only provide the hypotheses for the discriminative power and the predictability, but also provide the rationale for why these metrics can work for security requirements identification.

**Complexity:** Security experts claim that complexity is the enemy of security [22,23]. Complexity in requirements can lead to complex code which contains subtle vulnerabilities that are difficult to test and diagnose, providing more chances for attackers to exploit [Anonymous, 2016]. From this reason, we set up the following hypothesis on complexity in requirements.

- $H_{\text{CommentsComplexity}}$: Security requirements in OSS projects involve more discussions than non-security requirements.

A requirement containing more comments has a higher chance of having interaction with other issues, leading to more code having input from the external source to modules implemented by other developers without properly understanding the security concerns or assumptions of the modules [19]. We first tested four metrics, including number of comments/requirement (# C), number of noun in comments/requirement (# N), number of verbs in comments/requirement (# V), and average length of comments/requirements (ALC), by comparing their distribution difference between security and non-security requirements. For each dataset, we chose subsets of security and non-security requirements via simple random sampling (SRS) [24]. Then applied student $t$-test to test statistic difference between two sets for each metrics. Table 4 shows the testing results. Only ALC has the significant difference between security and non-security requirements. Therefore, only ALC is used to measure comments complexity.

- $H_{\text{StakeholdersComplexity}}$: Security requirements in OSS projects involve more stakeholders than non-security requirements.

We used the number of stakeholders/requirement (# S) to measure the stakeholders’ complexity. Issue tracking system used by three projects (i.e., JIRA) predefined several stakeholders: creators, assignees, comment authors, watchers, and voters. Creators, assignees, and comment authors provide their opinions and refinement of requirements via commenting on requirements. Different people have different opinion and preferences. Therefore, the more stakeholders provide their opinion, the more complicated the requirements are. In addition, human errors are the most frequent type of errors in the requirements [25]. The errors also indicate the high possibility of vulnerability. For watchers and voters, JIRA only shows how many of them, not who are they. That means they may or may not overlap with creators, assignees, and comment authors. If they do not belong to previous three types of stakeholders, that means they do not provide any opinions or refinements. Therefore, we ignore them when we calculate stakeholders complexity. The student $t$-test results (Table 4) shows that there is a significant difference between security and non-security requirements.

**External resources:** Stakeholders tend to provide external resources, e.g., URLs to documents on other websites, to provide the rationale for their refinement and explain their solutions. After analyzed three OSS projects, we found that three external resources have potential to help to distinguish security and non-security requirements.

From our observation, stakeholders add URLs to the requirements

### Table 1

<table>
<thead>
<tr>
<th>Project</th>
<th>Domain</th>
<th># of issues</th>
<th># of security issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis2</td>
<td>Web server</td>
<td>5751</td>
<td>109</td>
</tr>
<tr>
<td>Drools</td>
<td>Business</td>
<td>326</td>
<td>33</td>
</tr>
<tr>
<td>GeoServer</td>
<td>Geo information</td>
<td>8172</td>
<td>651</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Project</th>
<th>Project dependent indicator terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis2</td>
<td>Apache rampart, credentials, MEP (message exchange pattern), stack overflow exception, socket, SOAP (simple object access protocol), threadsafe, WS-Security, WS-Addressing</td>
</tr>
<tr>
<td>Drools</td>
<td>ACL (access control list), security-module-admin.properties, security-policy.properties</td>
</tr>
<tr>
<td>GeoServer</td>
<td>GeoServerSecurityFilterChain, GeoServerSecurityFilterChainProxy, KeyStoreProvider, OAuth2, Resource/AccessManager, SQL</td>
</tr>
</tbody>
</table>

### Table 3

<table>
<thead>
<tr>
<th>Project</th>
<th>Effectiveness: AS</th>
<th>[Retrieved]</th>
<th>Recall</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis2</td>
<td>13</td>
<td>134</td>
<td>1.00</td>
<td>0.10</td>
</tr>
<tr>
<td>Drools</td>
<td>4</td>
<td>14</td>
<td>1.00</td>
<td>0.29</td>
</tr>
<tr>
<td>GeoServer</td>
<td>54</td>
<td>347</td>
<td>0.96</td>
<td>0.15</td>
</tr>
</tbody>
</table>

### Table 4

<table>
<thead>
<tr>
<th>Project</th>
<th>Metric</th>
<th>Sec. Req.s</th>
<th>Non-Sec. Req.s</th>
<th>p value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(mean ± s.d.)</td>
<td>(mean ± s.d.)</td>
<td></td>
</tr>
<tr>
<td>Axis2</td>
<td># C</td>
<td>4.24 ± 4.67</td>
<td>3.97 ± 4.93</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td># N</td>
<td>11.23 ± 8.23</td>
<td>12.65 ± 9.72</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td># V</td>
<td>7.52 ± 6.26</td>
<td>6.71 ± 7.64</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>ALC</td>
<td>33.16 ± 21.37</td>
<td>15.26 ± 22.69</td>
<td>0.02*</td>
</tr>
<tr>
<td></td>
<td># S</td>
<td>7.26 ± 3.21</td>
<td>4.31 ± 4.59</td>
<td>0.04*</td>
</tr>
<tr>
<td>Drools</td>
<td># C</td>
<td>6.23 ± 3.76</td>
<td>5.44 ± 5.24</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td># N</td>
<td>10.32 ± 11.73</td>
<td>9.72 ± 9.27</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td># V</td>
<td>7.43 ± 7.53</td>
<td>6.99 ± 6.17</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>ALC</td>
<td>29.73 ± 32.61</td>
<td>17.29 ± 29.96</td>
<td>0.03*</td>
</tr>
<tr>
<td></td>
<td># S</td>
<td>6.33 ± 5.31</td>
<td>4.11 ± 4.23</td>
<td>0.04*</td>
</tr>
<tr>
<td>GeoServer</td>
<td># C</td>
<td>5.49 ± 7.62</td>
<td>4.24 ± 6.52</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td># N</td>
<td>11.33 ± 7.54</td>
<td>10.58 ± 3.72</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td># V</td>
<td>5.43 ± 5.29</td>
<td>6.34 ± 4.23</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>ALC</td>
<td>24.52 ± 28.31</td>
<td>16.33 ± 23.42</td>
<td>0.03*</td>
</tr>
<tr>
<td></td>
<td># S</td>
<td>8.21 ± 4.27</td>
<td>3.62 ± 3.69</td>
<td>0.02*</td>
</tr>
</tbody>
</table>
while they want to use information provided by the URLs. For instance, solutions for the current requirements, project dependabilities, and detailed description of requirements. However, not every URL can indicate the security issue. After analyzed three OSS project, we found that only two types of URLs indicate the security issues. In the Type I example, the URL points to the document which is mainly about security-related issues (Fig. 1(a)). In this case, the last part of the URL, which is the name of the security document (e.g., “selfadmin.html”), contains security indicator terms. In the Type II URL (Fig. 1(b)) points to a document which contains both security and non-security issues. However, it uses “#” to refers to the title of the specific section/chapter in this document. If the title of this specific section/chapter (e.g., “Character_Encodings”) contains security indicator terms, we classify it as a security related-URL. Therefore, we provide following hypothesis:

\[ H_{\text{SecurityURLs}}: \text{Security requirements in OSS projects involve more security related URLs than non-security requirements.} \]

Another external resource provided by stakeholders is the commits from version control system, e.g., CVS, GitHub. These systems record the diff for each commit. Security experts claim that even software use permission systems to attempt to limit the privileges of the softwares. However, existing permission systems are not sufficient to prevent sensitive data from being leaked [26]. Any outside sources can contaminate the sensitive data [27]. Therefore, we only interested in the commits which contain changes related to data exchange. From our observation, there are two types of code change strongly related to data exchange: (1) the change contains “=m”; and (2) the change contains “get” or “set” methods. Thus, only the commits which contain data exchange related changes are count as security-related commits. We provide following hypothesis for security-related commits:

\[ H_{\text{SecurityCommits}}: \text{Security requirements in OSS projects involve more security related commits than non-security requirements.} \]

Unlike previous external resources, CVE identifiers (CVE IDs) which are assigned by CVE Numbering Authorities (CNAs) and used to discuss and share information about a unique software vulnerability, is a strong indicator of security requirements. That means if a requirement contains a CVE ID, it is a security-related requirement. Therefore, we provide following hypothesis for CVE IDs:

\[ H_{\text{SecurityCommits}}: \text{A requirements in OSS projects involve any CVE IDs shall be considered as security requirements.} \]

4. Linear classifier based approach

In this section, we describe the general architecture of our linear classifier based approach. Our approach contains five parts: Information Preprocessing Component (IPC), NFR Classifier (NFR-C), Regression Models (RMs), CVE ID Detector(CID), and Linear Classifier (LC). Fig. 2 presents the framework of our linear classifier based approach.

Input requirements are processed by IPC to extract information of each metric described in Section 3.2. Then feed these information to four regression models (Comment Complexity Regression Model (CRM), Stakeholder Complexity Regression Model (SRM), Security

URLs Regression Model (URM), Security Commits Regression Model (CRM)), NFR-C and CID. For each requirement, each regression model will generate a weight between 0 to 1, which indicates the likelihood of this requirement is a security requirement. At the same time, CID will detect whether this requirement contains a CVE ID. If there is a CVE ID in this requirement, the weight of it is 1, otherwise, the weight of it is 0. For NFR-C, it directly reuses the approach from [7]. Finally, a linear classifier is used to generate a linear discriminant function which is an optimal way to summarize weights from NFR-C, CID, and all RMs. The result of the LC then is used to classifier whether this requirement is a security or non-security requirement. We briefly describe the regression model and the linear classifier in the following subsections.

4.1. Logistic regression model

Lots of classification techniques, such as J48 decision tree, Random forest, Naïve Bayes, Bayes Network, and Logistic Regression, have been widely applied to build security identification models [19,28]. Lessmann et al. [28] investigated 17 classification techniques, and reported that there is no significant difference in performance between those techniques. Same as [19], we used binary logistic regression to identify security requirements.

Binary logistic regression computes the probability of occurrence of an outcome event from given independent variables by mapping the
linear combination of independent variables to the probability of outcome using the log of odds ratio (logit). An open-source cluster-computing framework Apache Spark\(^9\) was used to build our regression models.

### 4.2. Linear classifier

A linear classifier is a statistical classification technique. The goal of it is to use an object’s feature values to identify which class it belongs to. It achieves this goal by making the classification decision based on the value of the linear combination of feature values\(^{[29]}\).

The discriminant function of linear classification shows in Eq. (1):

\[
f(\omega \cdot \vec{x}) = \Sigma (a_i x_i)
\]

where \(\vec{x}\) is a real vector of feature values, and \(\omega\) is a real vector of weights for each feature value. The weights vector \(\omega\) is learned from a set of labeled training samples by applying linear discriminant analysis (LDA).

Linear classifier is widely used in software engineering field, such as NPR identification\(^{[13]}\) and CVS issue status prediction\(^{[30]}\). The major advantages of the linear classifier are its simplicity and time efficiency. Therefore, we thought linear classifier could be a good candidate for our approach. To our best understanding, this research is the first one which applies linear classifier to solve the security identification problem. Therefore, we tried to answer following research questions:

- **RQ1**: Can the linear combination of feature values achieve better performance?

In addition, the function \(f(\omega \cdot \vec{x})\) maps all values above a certain threshold to one class and all other values to the second class. Therefore, the following question is:

- **RQ2**: What is the optimal threshold for linear classifier when identifying security requirements?

### 5. Measurement and results analysis

#### 5.1. Measurement

For evaluating binary identify modes, we used recall, precision, and \(F\)-measure.

Recall is defined as the ratio of correctly identified security requirements to actual security requirements:

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

(2)

where TP refers to true positives and FN refers to false negatives. Precision is defined as the ratio of correctly identified security requirements to all detected issues:

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

(3)

where FP refers to false positives. \(F\)-measure is a harmonic mean of recall and precision.

\[
F_2 = \frac{(1 + \beta^2) \cdot precision \cdot recall}{\beta^2 \cdot precision + recall}
\]

(4)

Failing to identify and realize security requirements will lead to a number of serious problems in software systems. So, we are more concerned about recall rather than precision. \(F_2\) is a widely used measurement which fits our purpose.

---

\(^9\)https://spark.apache.org
Section 3.1 are completed (Table 3). Therefore, we apply LC to these datasets, results are shown in the latest row of Table 5. Statistic analysis shows that there is no significant difference between results from different datasets.

6. Discussion and conclusion

We summarize the contributions of our study as below:

- **Linear classifier based approach:** To our best understanding, this research is the first one which applied the linear classifier to identify security requirements.

- **Threshold testing:** We applied our approach to three datasets. Through systematic analysis, we suggested that 0.5 is an optimal threshold for our linear classifier based security requirements identification approach.

However, there are two main limitations in our study: First, like we discussed in Section 5.2, metrics used to measure requirements complexity may also relate to other non-functional requirements. In future research, we need to find other complexity metrics with the hope that they can improve the precision without decrease the recall.

Second, in this study, we only tested the linear combination of feature values. The advantages including simplicity and time efficient. However, compared to the non-linear classifier, such as KNN, its disadvantage is also obvious. That is the accuracy of it relatively lower.

Based on our previous discussion, we propose following future work: (1) test more complexity metrics, (2) compare linear classifier with non-linear classifiers to find the most optimal way to combine all feature values, and (3) integrating our models to industry common tools, such as JIRA, to provide real-time support.

**Table 5**

Comparison between NFR-C, RMs, and LC: R refers to recall, P refers to precision, bold values are highest values in their columns.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Axis2</th>
<th>Drools</th>
<th>GeoServer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
<td>P</td>
<td>F₂</td>
</tr>
<tr>
<td>NFR-C</td>
<td>0.62</td>
<td>0.26</td>
<td>0.48</td>
</tr>
<tr>
<td>CRM</td>
<td>0.73</td>
<td>0.21</td>
<td>0.49</td>
</tr>
<tr>
<td>SRM</td>
<td>0.68</td>
<td>0.16</td>
<td>0.41</td>
</tr>
<tr>
<td>URM</td>
<td>0.53</td>
<td>0.20</td>
<td>0.45</td>
</tr>
<tr>
<td>CRM</td>
<td>0.46</td>
<td>0.69</td>
<td>0.49</td>
</tr>
<tr>
<td>LC</td>
<td>0.88</td>
<td>0.67</td>
<td>0.83</td>
</tr>
<tr>
<td>Pilot</td>
<td>0.84</td>
<td>0.71</td>
<td>0.81</td>
</tr>
</tbody>
</table>

Fig. 3. Linear classifier based security requirements identification approach performance under different threshold.

**Supplementary material**

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.jii.2018.11.001.

**References**

[19] Y. Shin, A. Menesly, L. Williams, J.A. Osborne, Evaluating complexity, code churn,


