Empirical studies on the impact of filter-based ranking feature selection on security vulnerability prediction

Xiang Chen¹ ²  |  Zhidan Yuan¹  |  Zhanqi Cui³  |  Dun Zhang¹  |  Xiaolin Ju¹

¹School of Information Science and Technology, Nantong University, Nantong, China
²Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin, China
³Computer School, Beijing Information Science and Technology University, Beijing, China

Abstract
Security vulnerability prediction (SVP) can construct models to identify potentially vulnerable program modules via machine learning. Two kinds of features from different points of view are used to measure the extracted modules in previous studies. One kind considers traditional software metrics as features, and the other kind uses text mining to extract term vectors as features. Therefore, gathered SVP data sets often have numerous features and result in the curse of dimensionality. In this article, we mainly investigate the impact of filter-based ranking feature selection (FRFS) methods on SVP, since other types of feature selection methods have too much computational cost. In empirical studies, we first consider three real-world large-scale web applications. Then we consider seven methods from three FRFS categories for FRFS and use a random forest classifier to construct SVP models. Final results show that given the similar code inspection cost, using FRFS can improve the performance of SVP when compared with state-of-the-art baselines. Moreover, we use McNemar’s test to perform diversity analysis on identified vulnerable modules by using different FRFS methods, and we are surprised to find that almost all the FRFS methods can identify similar vulnerable modules via diversity analysis.

1 | INTRODUCTION

A large number of security vulnerabilities are reported every year, and these security vulnerabilities have imposed significant damages to individuals and companies. However, finding security vulnerabilities is a challenging task and needs an understanding of both the software under test and the attackers’ mindset [1]. Recently, methods using machine learning are increasingly popular in security vulnerability analysis and discovery, and this kind of method has received more interest [2].

Motivated by the previous studies on software defect prediction (SDP) [3,4], security vulnerability prediction (SVP) aims to construct models via machine learning to identify potentially vulnerable program modules similarly. For modules extracted from projects, features used to measure these modules are vital to constructing high-quality SVP models. In a recent study, Walden et al. [5] considered two kinds of features. One kind considered traditional software metrics (SM) (such as lines of code [LOC], cyclomatic complexity) designed from the SDP research domain as features, and the other kind used text mining to extract term vectors as features.

However, considering these two kinds of features can eventually generate a large number of features. Moreover, not all the features are beneficial to the construction of SVP models, and this problem is called the curse of dimensionality. In most cases, data sets with high dimensionality can result in the high computational cost of model construction and performance degradation for constructed models. In our investigated SVP data sets, we find the curse of dimensionality problem is severe after using principal component analysis. Therefore, using feature selection, which is an effective method for dimensionality reduction, to improve the performance of SVP has a critical research significance.

To analyse the impact of feature selection on SVP performance, we consider three real-world large-scale web applications as our empirical subjects. These subjects have 3466 modules (i.e. files) and 223 vulnerabilities (such as code injection vulnerabilities and cross-site request forgery vulnerabilities). By considering traditional SM and term vectors...
extracted by text mining as features, the number of features for these gathered SVP data sets ranges between 3898 and 18,318. Existing feature selection methods can be classified into three categories: filter-based ranking methods, filter-based subset methods, and wrapper-based subset methods [6]. Due to the excessive number of features in gathered SVP data sets, we only consider filter-based ranking feature selection (FRFS) methods in our study. Compared to the other categories of feature selection methods, FRFS methods have much less computational cost because the latter will spend a lot of time on searching for the optimal feature subsets in a large-scale search space [7].

In our empirical studies, we consider seven different feature ranking methods (i.e. rankers) in total for FRFS and use a random forest (RF), which shows competitiveness performance in a previous study [5], as the classifier to construct SVP models. We mainly investigate the following two research questions.

1.1 | RQ1: Can using FRFS methods improve the performance of SVP?

In this RQ, we first find that using FRFS methods can improve the SVP performance no matter what ranker is used on our chosen web applications compared to three state-of-the-art baseline methods. The first two baseline methods adopted by Walden et al. [5] use only SM or only term vectors extracted by text mining to construct SVP models. The last baseline method uses both SM and term vectors extracted by text mining as features to construct SVP models. We then find that previous suggestions on the optimal feature selection ratio for SDP [6,8–10] are not valid for SVP. Therefore, the optimal feature selection ratio should be carefully chosen according to the characteristics of gathered SVP data sets. Later, after analysing the selected features by our FRFS methods, we suggest that traditional SM and features extracted by text mining should be both considered.

1.2 | RQ2: Can different FRFS methods identify the same vulnerable modules?

In this RQ, we want to investigate whether there exists a difference between two SVP models using different FRFS methods in terms of the specific vulnerable modules each FRFS method identifies and does not identify. In particular, we use McNemar's test to perform diversity analysis on identified vulnerable modules. The final results show that the prediction diversity phenomenon on vulnerable modules for different FRFS methods does not exist in most cases. Moreover, we analyse the number of vulnerable modules which cannot be correctly identified by any FRFS methods. Final results show performance bottleneck exists in state-of-the-art SVP methods since any of our considered SVP methods cannot identify some of the vulnerable modules. Therefore, we should design more powerful SVP methods (such as using deep learning to learn semantic features from source codes in the modules automatically) in the future.

The main contributions of this paper can be summarized as follows:

- To the best of our knowledge, we are the first to perform a depth analysis of the impact of FRFS on SVP. We consider seven different feature ranking methods in total from three different FRFS categories and then use a RF to construct SVP models.
- Empirical studies are designed based on three real-world web applications, and final results show the effectiveness of applying FRFS to SVP compared to three state-of-the-art baseline methods. We also use McNemar's test to perform diversity analysis on identified vulnerable modules when comparing different FRFS methods.

1.3 | Paper organization

The rest of this paper is organized as follows: Section 2 analyses the background of SVP and the motivation of our study. Section 3 shows our case study setup. Section 4 gives case study results. Section 5 discusses potential threats to validity analysis. Section 6 summarizes related work for our study. Section 7 concludes this paper and discusses some potential future work.

2 | BACKGROUND AND MOTIVATION

In this section, we first introduce the background of SVP and analyse the difference between SVP and SDP. Then we emphasize the motivation of our study and the design motivation of research questions.

2.1 | Background of SVP

SVP resorts to machine learning and aims to identify vulnerable modules in advance. Therefore, the allocation of limited resources for security audits and code inspection can be optimized. Then more security vulnerabilities may be detected and mitigated as soon as possible.

The brief process of SVP can be summarized as follows [5]. SVP first extracts and labels program modules from software historical repositories, such as version control systems, bug tracking systems, National Vulnerability Database (NVD)s. The granularity of modules can be set to source code file, object-oriented class, binary component as needed. Then it uses features to measure these modules. Later, it labels a program module vulnerable if this module contains at least one vulnerability by analysing commit messages and bug reports in the bug tracking system or NVDS. Finally, it uses a specific classification method (i.e. RF) to train a model based on the gathered SVP data set. For a new program module, we can use the same metrics to measure this module and then use the
trained model to predict whether this module is vulnerable or non-vulnerable.

Based on the SVP process analysis, it is not hard to find that this research topic is mainly motivated by SDP [4,11–17]. Vulnerabilities and defects have a certain similarity [2]. They are both caused by human mistakes, and these mistakes may be related to the code complexity or the developer experience. Therefore, most of SVP studies used metrics designed in SDP to construct SVP models and showed the feasibility of this solution. However, different from defects, vulnerabilities are instances of errors in the specification, development, or configuration of software such that their executions can implicitly or explicitly violate security policies. Therefore, vulnerable modules should have different characteristics when compared to defects, and we should design specific metrics to focus on these characteristics. Moreover, the number of vulnerable modules is far less than the number of defective modules in most projects [18]. Therefore, the problem of class imbalance in SVP is more challenging than SDP.

2.2 | Motivation

2.2.1 | Data set redundancy analysis

In our study, we use principal component analysis (PCA) to analyse data redundancy in gathered SVP data sets (The details of these data sets can be found in Section 3.2) [6]. Figure 1 shows the PCA analysis result for PHPMyAdmin data set. In this figure, we sort principal components (PCs) in the descending order according to their explained variance. For example, c1∼c20 denotes the top 20 PCs, and they can explain 61.543% of the variance. To explain 95% of the variance, we only need the top 2.29% of PCs. The line in Figure 1 shows the cumulative variance explained by the PCs. It is not hard to find that most of the variance is contained in a small number of PCs. The same findings can be found in the remaining two gathered SVP data sets (i.e. Drupal and Moodle). Therefore, all the SVP data sets used in our empirical studies have certain data redundancy.

We consider feature selection methods to reduce the data redundancy in gathered SVP data sets, which can be used to perform dimensionality reduction. Existing feature selection methods can be classified into three categorizes (i.e. filter-based ranking methods, filter-based subset methods and wrapper-based subset methods) [6,19,20]. In particular, filter-based ranking methods select features by estimating their capacity for contributing to the fitness of the constructed models. Filter-based subset methods select features that collectively have a good prediction performance. Wrapper-based subset methods use a predetermined classifier and a performance measure to evaluate the importance of a feature subset for constructed models. As we consider both traditional metrics and term vectors extracted by text mining as features in our study, the final gathered SVP data sets eventually have 3898∼18,318 features. Therefore, we do not consider filter based subset methods and wrapper-based subset methods, since the computational cost of these methods is prohibitive [7].

Therefore, we want to analyse the first research question as follows.

Previous studies on SDP [6,19,21] show that using feature selection can improve the performance of SDP. However, whether using FRFS methods can improve the performance of SVP has not been thoroughly analysed. Then, for SVP, previous studies [6,8,9] using FRFS methods for dimensionality reduction suggested selecting top \( \log_2 N \) features from the list of ranked features, where \( N \) is the total number of features. For SVP, a recent study [10] suggested using top-10 or top-100 features. However, whether these recommendations on optimal feature selection ratio are valid in our empirical studies is still unknown and needs further investigation. Finally, we want to analyse the proportion of different types of features in the selected features since we use both SM and text features (TF) to measure the extracted modules.

In the first research question, we mainly compare and rank the performance of different FRFS methods. In the second research question, we want to conduct an in-depth analysis through the analysis of vulnerable modules identified by SVP models using different FRFS methods. We want to investigate whether there exists a difference between two SVP models using different FRFS methods in terms of the specific vulnerable modules each FRFS method identifies and does not identify. Moreover, we also want to find the vulnerable modules which cannot be predicted correctly by all the FRFS methods.

3 | CASE STUDY SETUP

In this section, we first illustrate our case study approach. Then we show the studied data sets. Later we introduce the details in measuring extracted modules, performing data preprocessing, performing cross-validation. Finally, we analyse the measures in performance evaluation and statistical analysis methods for answering RQ1 and the diversity analysis method on identifying vulnerable modules for answering RQ2.

3.1 | Approach

The overview of our case study approach can be found in Figure 2.

In this approach, we first use cross-validation to generate the training set and the test set (in Section 3.5), respectively. Then we consider two types of metrics to measure the extracted program modules (in Section 3.3). Later, we perform data preprocessing on the training set, including the feature selection method and the class imbalanced method. Finally, we can construct the SVP models by a RF classifier. The details of data preprocessing and model construction can be found in Section 3.4. We use this approach to answer two research questions in Section 2.2 (i.e. performance evaluation and diversity analysis on identifying vulnerable modules).
3.2 | Experimental subjects

In our empirical study, we choose the experimental subjects based on the following criteria: (1) The applications should be open-source projects, which we can access their source code and measure the extracted program modules. (2) The applications should be web applications written in PHP, since PHP is the most popular programming language in Web Development. (3) The applications should contain at least 20 detected vulnerabilities.

Based on the above criteria, we consider three real-world open-source web applications written in PHP programming language. The selected web applications are Drupal, Moodle and PHPMyAdmin. In particular, Drupal is a widely used content management system, Moodle is an open-source learning management system, and PHPMyAdmin is a web-based management tool for MySQL database. Notice, in our study, we only consider these vulnerabilities related to a specific version of these applications (i.e. Drupal 6.0, PHPMyAdmin 3.3, and Moodle 2.0). In these web applications, the granularity of the extracted modules is set to file. Finally, 3466 modules are extracted, and 223 vulnerabilities are identified in total. Security vulnerabilities are mined for each application from vulnerability databases. In particular, for the application Moodle, the data source was the NVD\(^1\), while for the applications Drupal and PHPMyAdmin, the security announcements maintained by those projects are used. The detected vulnerabilities for these web applications have a variety of types, such as code injection vulnerabilities, cross-site request forgery vulnerabilities, cross-site scripting vulnerabilities and path disclosure vulnerabilities.

3.3 | Measuring extracted modules

Extracted modules can be measured in two different ways. One way uses traditional SM inspired by traditional studies in SDP (i.e. SM). The other way uses text mining to extract term vectors as features (i.e. TFs).

3.3.1 | Software metrics

Here 12 different SM are considered: LOC, LOC for non-HTML, Number of functions (NM), cyclomatic complexity, maximum nesting complexity, Halsteads volume, total external calls, fan-in, fan-out, internal functions/methods called, external functions/methods called and external calls to functions/methods. We introduce the details and abbreviation (i.e. corresponding metric name in Weka arff file) of each metric as follows:

**Lines of code**
This metric measures the number of lines in a source file excluding lines, such as blank lines and comments.

**LOC for non-HTML**
This metric is the same as the metric LOC. It measures the number of lines in a source file, except HTML content embedded in from which 3,466 modules (files) (i.e. content outside of php start/end tags).

**Number of functions**
This metric measures the number of function/method definitions in a source file.

**Cyclomatic complexity**
This metric measures the size of a control flow graph after linear chains of nodes are collapsed into one.

**Maximum nesting complexity**
This metric measures the maximum depth to which loops and control structures in the file are nested.

**Halsteads volume**
This metric measures the volume of one source file that can be computed by \((N_1 + N_2)\log(n_1 + n_2)\). Here \(n_1\) and \(n_2\) denote

\(^1\) [http://nvd.nist.gov/](http://nvd.nist.gov/)
the number of distinct operators and the number of distinct operands, respectively. \( N_1 \) and \( N_2 \) denote the total number of operators and the total number of operands, respectively. To measure this metric, operators are method names and PHP language operators, while operands are parameter and variable names.

**Total external calls (NIC)**
This metric measures the number of instances where a statement in the source file being measured invokes a function/method defined in another source file.

**Fan-in (NICU)**
This metric measures the number of source files (excluding the source file being measured), which contain statements that invoke a function/method defined in one source file being measured.

**Fan-out (NOEFCU)**
This metric measures the number of source files (excluding the source file being measured), which contain functions/methods invoked by statements in one source file being measured.

**Internal functions/methods called (NOIC)**
This metric measures the NM/methods defined in the source file being measured which are called at least once by a statement in one same source file.

**External functions/methods called (NOEFC)**
This metric measures the NM/methods defined in other source files which are called at least once by a statement in one source file being measured.

**External calls to functions/methods (NOECU)**
This metric measures the number of source files (excluding the source file being measured) calling a particular function/method defined in the source file being measured, summed across all functions and methods in the source file being measured.

### 3.3.2 | Text features

Term vectors as TFs are extracted by text mining. The tokens and their associated frequencies of each PHP program module (i.e. file) are extracted by PHP’s built-in function token_get_all. These tokens represent language keywords, punctuation, and other code constructs. Notice, firstly, comments and whitespace are ignored. Secondly, string and numeric literals are converted into fixed tokens. For example, T_STRING is used to represent a string literal instead of the string contents.

### 3.3.3 | Statistics

It is not hard to find that these two ways of measuring extracted program modules do not consider programming language characteristics. Therefore, these metrics are programming language independent and general.

The statistics of these data sets can be found in Table 1. These characteristics include the project name, the number of extracted modules, the number (percentage) of vulnerable modules, and the number of features. These web applications have been used in previous SVP studies [5,10,22,23], and the representativeness of chosen data sets can be guaranteed.

### 3.4 | Performing data preprocessing and constructing models

#### 3.4.1 | FRFS methods

There are three categories for FRFS methods: statistics-based methods, probability-based methods and instance-based methods [6,19,21]. In our study, we consider seven different FRFS methods (i.e. two rankers in statistic-based methods, three rankers in probability-based methods and two rankers in instance-based methods). The details of these different rankers, which include category, name, description and abbreviation, can be found in Table 2.
Since we could not know the optimal feature selection ratio in advance, we will investigate different feature selection ratio from 5% to 95% and set the step size to 5%. For a specific feature selection ratio, we will construct the model $M$ by using the selected features and evaluate the constructed model on the test set in terms of $R$ performance measure (illustrated in Section 3.6). The pseudocode of determining optimal feature selection ratio can be found in Algorithm 1.

**Algorithm 1 FRFS for SVP**

**Input:**
- $train$: training data
- $test$: test data
- $N$: the number of original features
- $classifier$: classifier
- $ranker$: ranker used for FRFS

**Output:**
- $(R, \text{ratio})$: a pair of the best $R$ and the corresponding feature selection ratio
- 1: $\text{best } R \leftarrow 0$, $\text{best } Ratio \leftarrow 0\%$
- 2: Sorting features in the descending order according to ranker based on the training data $train$
- 3: Applying the class imbalance method to $train$
- 4: for $\text{ratio}$ in $\{5\%, 10\%, \ldots, 95\%\}$ do
- 5: $\text{tmp } R \leftarrow 0$
- 6: Selecting top $[\text{ratio } \times N]$ features
- 7: Constructing the model $M$ by using $classifier$ based on $train$
- 8: Evaluating the model $M$ on the test data $test$ and store the performance of $R$ to $R_{\text{ratio}}$
- 9: if $R_{\text{ratio}} > \text{best } R$ then
- 10: $\text{best } R \leftarrow R_{\text{ratio}}$, $\text{best } Ratio \leftarrow \text{ratio}$
- 11: end if
- 12: end for
- 13: return $(\text{best } R, \text{best } Ratio)$

### 3.4.2 Class imbalanced method

Compared to defects, vulnerabilities are far fewer in the project and detecting vulnerabilities is similar to searching for a needle in a haystack [18]. In line with the previous study, we use a classical class imbalance method (i.e. SpreadSubsample) provided by Weka package and use the same setting suggested by Walden et al. [5].

#### 3.4.3 Classifier

We use RF as the classifier to construct SVP models, since RF classifier is widely used in previous SVP studies and shows competitiveness performance [5,23]. RF is an ensemble learning method for classification. It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes [30].

#### 3.5 Performing cross-validation

To evaluate the performance of trained models, we use 10×3-fold cross-validation (CV). The threefold CV process can be summarized as follows: The instances are randomly divided into three folds with approximately equal size. Each fold has the same ratio of vulnerable modules as the entire data set. A model is trained based on the instances in two folds (i.e. the training set), and then the model is tested based on the instances in the remaining fold (i.e. the test set). This process is repeated three times and each fold is used as the test set at least once. The threefold CV is repeated 10 times to overcome the effect of randomness. The reason for not using the 10-fold CV is that the number of security vulnerabilities in some applications is too small. For example, the Moodle application only contains 24 vulnerabilities. If considering the 10-fold CV used by [10,23], each fold can only have 2~3 vulnerabilities, which is not very helpful for building a high-quality SVP model.

### 3.6 Performance measures and statistical analysis methods

#### 3.6.1 Performance measures

According to the ground truths and the predicted class labels, we can compute the number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN), respectively based on the confusion matrix in Table 3. Then, we can compute the precision measure $P$ and the recall measure $R$ as follows:

$$P = \frac{TP}{TP + FP} \quad (1)$$

$$R = \frac{TP}{TP + FN} \quad (2)$$

The measure $P$ denotes the proportion of modules that are correctly classified as vulnerable modules among those predicted as vulnerable. The measure $R$ denotes the proportion of vulnerable modules, which are correctly predicted.

Considering the significant impact of vulnerable modules, the measure $R$ is more important than the measure $P$ in SVP. Because once some modules with vulnerabilities are not detected, they will cause huge losses after the project is deployed.

#### TABLE 1 The statistics of security vulnerability prediction data sets in our empirical studies

| Dataset     | # Files | (#%) Vulnerable | # Text | # Software | Metrics |
|-------------|---------|-----------------|--------|------------|---------|
| Moodle      | 2942    | 62 (30.69%)     | 18,306 | 12         |
| PHPMyAdmin  | 322     | 27 (8.39%)      | 5232   | 12         |
| Drupal      | 202     | 62 (30.69%)     | 3886   | 12         |

*Since we could not know the optimal feature selection ratio in advance, we will investigate different feature selection ratio from 5% to 95% and set the step size to 5%. For a specific feature selection ratio, we will construct the model $M$ by using the selected features and evaluate the constructed model on the test set in terms of $R$ performance measure (illustrated in Section 3.6). The pseudocode of determining optimal feature selection ratio can be found in Algorithm 1.*
It is not hard to find that the measure $R$ is considered from the perspective of benefit. However, if the model simply classifies all the modules as vulnerable modules, the value of $R$ will be 100% and this model has no value at all. Therefore, we consider the module inspection ratio (IR) measure from the perspective of cost. This measure returns the proportion of modules, which are classified as vulnerable modules. The higher value of the IR measure means that a developer should spend more effort to review these modules. It can be computed as follows:

$$IR = \frac{TP + FP}{TP + TN + FP + FN}$$  \hspace{1cm} (3)$$

Obviously, the smaller the IR value, the better the trained SVP model.

### 3.6.2 Statistical analysis methods

We use the Scott-Knott test [31] to rank all the SVP models using different FRFS methods in terms of a specific performance measure based on the results of all the empirical subjects. The Scott-Knott test was recommended by Ghotra et al. [32] when they compared different SDP methods. Since the Scott-Knott test does not suffer from the overlapping group’s issue in post hoc tests (such as Friedman-Nemenyi test), we can use the Scott-Knott test to analyse whether some SVP models outperform others, and it can generate a global ranking of all the SVP models. In particular, the Scott-Knott test performs the grouping process in a recursive way. Firstly, the Scott-Knott test uses a hierarchical cluster analysis method to partition all the SVP models into two ranks based on the mean performance in terms of a specific performance measure (such as the measure $R$ or the measure IR). Then, the Scott-Knott test is recursively executed again in each rank to divide the ranks further if the divided ranks are significantly different.

The test will terminate when the ranking can be no longer divided into statistically different rankings.

To determine whether an SVP model performs significantly better or worse than another SVP model, we need to use the Wilcoxon signed-rank test and Cliff’s $\delta$. In particular, the Wilcoxon signed-rank test [33] can be used to analyse whether the performance difference between two SVP models is statistically significant. We also use the Benjamini-Hochberg (BH) procedure [34] to adjust $p$ values if we make multiple comparisons. Cliff’s $\delta$ [35] is a non-parametric effect size measure, and it is used to examine whether the magnitude of the difference between two SVP models is substantial or not. In summary, a SVP model performs significantly better or worse than another SVP model, if BH corrected $p$ value is less than 0.05 and the effectiveness level is not negligible based on Cliff’s $\delta$ (i.e., $|\delta| \geq 0.147$). While the difference between two SVP models is not significant, if $p$ value is not less than 0.05 or $p$-value is less than 0.05 and the effectiveness level is negligible (i.e., $|\delta| < 0.147$).

### 3.7 Diversity analysis method on identifying vulnerable modules

To evaluate whether different FRFS methods result in distinct predictions on vulnerable modules, we test the following null hypothesis:
Ho: Both FRFS methods FSa and FSb can identify similar vulnerable modules in the same project.

We use McNemar's test [36] with the 95% confidence level to perform diversity analysis on vulnerable modules between different FRFS methods. McNemar's test is a non-parametric test. Therefore, it does not need any assumption on the distribution of a subject variable. Notice all the SVP models using different FRFS methods are constructed on the same training data, and then applied to the same test set. Therefore, McNemar's test is applicable to our study.

To perform McNemar’s test, we need to construct a contingency matrix based on the prediction results by two models using different FRFS methods (i.e., FSa and FSb), which is shown in Table 4. In this contingency matrix, $N_{cc}$ denotes the number of modules that both FRFS methods achieve correct predictions. $N_{cw}$ denotes the number of vulnerable modules that the FRFS method FSa achieves a correct prediction while the FRFS method FSb achieves a wrong prediction. $N_{wc}$ denotes the number of vulnerable modules that the FRFS method FSb achieves a correct prediction while the FRFS method FSa achieves a wrong prediction. Finally, $N_{ww}$ denotes the number of vulnerable modules that both FRFS methods have wrong predictions.

In our empirical studies, we use mcnemar function in stats-model provided by R package exact 2×2 to perform McNemar’s test. If $p$ value is smaller than 0.05, we will reject the null hypothesis H0 (i.e. The vulnerable modules identified by these two FSFR methods FSa and FSb are almost the same). We use an artificially constructed illustrative example in Table 5 to show the rationality of this diversity analysis method. In this table, there are 10 vulnerable modules (i.e., m0, m1, ..., m9). The prediction results of three different methods (i.e., M1, M2 and M3) can be found in the last three columns. Here 1 means that this module is predicted as the vulnerable module and 0 means that this module is predicted as the non-vulnerable module by the corresponding method. Based on McNemar’s test, the $p$ value of M1 versus M2 is 0.03 and this means these two methods can almost identify distinct vulnerable modules. While the $p$ value of M1 versus M3 is 0.56, and this means these two methods almost identify the same vulnerable modules.

4 | CASE STUDY RESULTS

4.1 | (RQ1) Can using FRFS methods improve the performance of SVP?

4.1.1 | Method

In this RQ, We use FS1 to FS7 to denote the FRFS methods using the ranker Ranker1, Ranker2, ..., Ranker7, respectively based on the corresponding optimal feature selection ratio by using Algorithm 1. Then, we want to compare FRFS methods based on different rankers with state-of-the-art baseline methods via the Scott-Knott test. In our study, we mainly consider the following three baseline methods:

4.1.2 | Results

By using the Scott-Knott test, the final comparison results on all the three empirical subjects in terms of the measure $R$ and the measure $IR$ can be found in Figure 3.
In terms of the measure $R$, all the FRFS methods have been grouped into the first group (i.e. the categories of FRFS methods do not influence the performance), and all the baseline methods have been grouped into the second group. Therefore, no matter what FRFS method is used, SVP models using FRFS methods can perform significantly better than the baseline methods NFSA, NFST and NFSM. Among seven FRFS methods, the method FS6 has the best performance, and the method FS7 has the worst performance. Based on the median value, The FRFS method FS6 can improve the performance by 13.66%, 12.50% and 16.67% on average, when compared to the baseline methods NFSA, NFST and NFSM, respectively.

In terms of the measure $IR$, regardless of which FRFS method is used, the code inspection cost of all the models is at the same level because these methods cannot be further partitioned. This means whether using FRFS or not, the developers would spend similar efforts to perform code inspection on identified modules.

**Finding 1**
Based on the above analysis, we can find that when given the similar code inspection cost, FRFS can improve the performance of SVP no matter what ranker is used. These results confirm the effectiveness of using FRFS for SVP model optimization.

Then we collect the optimal feature selection ratio for each FRFS and show the distribution of these selection ratios in Figure 4 for each data set. In these sub-figures, we also use the dotted line to indicate the recommended value in previous studies [6,8,9]. Notice, there are three recommend values (i.e. top-10, top-100, and top-$\log_2 N$). Since the gathered data sets contain a large number of features, we only draw the dotted line corresponding to top-100. For the data set Drupal, the median/mean values of the optimal ratios range from 20%/31.17% to 65%/68%, while the recommended value is 0.26% (i.e. top-10), 2.57% (i.e. top-100) and 0.31% (i.e. top-$\log_2 N$), respectively. For the data set PHPMyAdmin, the median/mean values of the optimal ratios range from 22.50%/34% to 65%/60.33%, while the recommended value is 0.19%, 1.91% and 0.24%, respectively. For the data set Moodle, the median/mean values of the optimal ratios range from 10%/22.17% to 27.5%/33.33%, while the recommended value is 0.05%, 0.55% and 0.08%. In these sub-figures, we can find that the optimal feature selection ratios in our empirical study are inconsistent with the recommendations by previous studies [6,8–10].

**Finding 2**
The optimal feature selection ratio is closely related to the SVP data set's characteristics and should be chosen carefully in future SVP studies.

Finally, we analyse the number of selected SM and TFs by FRFS. Since we use 10×3-fold CV, there are 30 optimal feature subsets for each SVP data set, and we show the mean value and standard deviation. If there are no features of a specific type (SM or TF) exist in the final selected feature, we call these two types of features are not complementary. Otherwise, we call these two types are complementary. The final results based on the FRFS method FS6 can be found in Table 6. Based on Table 6, we can find that using FRFS, most of the features from SM
are selected, and a small number of features from TF are selected. Similar results can also be found when considering other FRFS methods.

Finding 3
To construct high-quality SVP models, we should consider features from SM and TF together since two types of features have a certain complementarity.

4.2 | (RQ2) Can different FRFS methods identify the same vulnerable modules?

4.2.1 | Method

We perform diversity analysis on identifying vulnerable modules for different FRFS methods by using the method introduced in Section 3.7. Because we use 10×3-fold CV, there are 10 different prediction results (considering all the vulnerable modules) for each SVP data set, and we show the number of cases, which p value of the McNemar’s test is smaller than 0.05.

Then, we want to analyse the number of vulnerable modules which cannot be correctly identified by the methods in a specific category. Since we use 10×3-fold CV, there are ten different results (considering all the vulnerable modules) for each SVP data set. The process of identifying vulnerable modules for each time can be found in Figure 5. For the methods in FRFS category, we first identify the vulnerable modules, which cannot be correctly identified by the method FS1, and use noVul(FS1) to denote these modules. After analysing all the FRFS methods, we can identify the vulnerable modules, which cannot be correctly identified by all the FRFS methods, and use noVul(FS) to denote these modules. Notice noVul(FS) = noVul(FS1)∩⋯∩noVul(FS7). For the methods in no feature selection (NFS) category (i.e. the baseline methods in RQ1), we can first use noVul(NFSA) to denote the vulnerable modules, which cannot be correctly identified by the method NFSA. After analysing all the NFS methods, we can use noVul(NFS) to denote the vulnerable modules, which cannot be identified by all the NFS methods. noVul(NFS) = noVul(NFSA)∩noVul(NFST)∩noVul(NFSM). Finally, we can use noVul(ALL) to denote the vulnerable modules, which cannot be identified by all the methods considered in this study.

4.2.2 | Results

We use McNemar's test to perform diversity analysis on vulnerable modules to evaluate whether SVP models using different FRFS methods can result in distinct predictions. The final results can be found in Table 7. In this table, we find that the prediction diversity phenomenon on vulnerable modules for different FRFS methods does not exist in most cases. When analysing from the method perspective, we only find
that the prediction diversity phenomenon only exists between FS3 and other FRFS methods.

Finding 4

The prediction diversity phenomenon on vulnerable modules for different FRFS methods does not exist in most cases.

Moreover, we analyse the number of vulnerable modules which cannot be correctly identified by the methods in a specific category. Using the process in Figure 5, we show the mean value and standard deviation for these ten runs. The results can be found in Table 8. The number in parentheses indicates the number of vulnerable modules in each project. FS category includes all the FRFS methods and we identify the number of vulnerable modules, which none of the methods in this category can correctly identify. NFS category includes all the baseline methods (i.e. NFSA, NFST and NFSM) and we identify the number of vulnerable modules, which none of the methods in this category can correctly identify. Finally, ALL category includes all the FRFS methods and baseline methods. We also identify the number of vulnerable modules, which none of the methods in this category can correctly identify. Based on Table 8, we first can find the number of vulnerable modules that the methods in the FS category can not correctly identify is less than the number of vulnerable modules that the methods in the NFS category cannot correctly identify except for PHPMyAdmin data set. Then we find that the number of vulnerable modules that the methods in the ALL category cannot correctly identify is less than the methods in the FS category and the methods in the NFS category, which indicate that the methods in the FS category and the methods in the NFS category have a certain complementary. Finally, we also find there exists a performance bottleneck in state-of-the-art SVP methods, since some of vulnerable modules can not be identified by any of our considered SVP methods.

Taking Drupal data set as an example, for the first threefold CV process, we find nine vulnerable modules, which cannot be identified by the methods in the FRFS category. These modules are actions.inc, database.inc, session.inc, tablesort.inc, book-all‐books‐block.tpl.php, comment.tpl.php, dblog.module, forum-topic-list.tpl.php and user-profile-category.tpl.php. Then we find six vulnerable modules, which cannot be identified by the methods in the NFS category. These modules are session.inc, book-all-books-block.tpl.php, comment.tpl.php, forum-topic-list.tpl.php, tracker.module and user-profile-category.tpl.php. Finally, we find five modules cannot be identified by all the methods considered in our study. These modules are session.inc, book-all-books-block.tpl.php, comment.tpl.php, forum-topic-list.tpl.php and user-profile-category.tpl.php. This means these vulnerable modules can not be identified by using existing features. The details of the unidentified vulnerable modules can be found in our project homepage.2

Finally, we perform analysis in terms of different types of vulnerabilities. In this study, we mainly consider the following five types of vulnerabilities: authorization related issues, code injection, cross-site request forgotten, cross-site scripting (XSS), path disclosure. Given a specific type of vulnerability, we first compute the percentage of these vulnerable modules (i.e. All Correct), which can be detected by all the methods in the FS category. Then we compute the percentage of these vulnerable modules (i.e. All Wrong), which cannot be detected by all the methods in the FS category. The final results can be found in Figure 6. In this figure, we can find code injection vulnerability (81.3%) can be most easily identified. Moreover, we can also find XSS vulnerability (16.8%) can be most difficult identified, and the percentage of all wrong for other types of vulnerabilities is between 3.8% and 8.9%.

Finding 5

We should design more powerful SVP methods in the future. For example, we can use deep learning to learn semantic features from program modules automatically.

2 https://github.com/ATOMYAD/vulnerable-modules-not-identified
In this section, we mainly discuss the potential threats to the validity of our empirical studies.

5.1 Threats to internal validity

These threats are mainly concerned with the uncontrolled internal factors that might influence the experimental results. We have double checked our experiments and implementation of different SVP methods. Later, we use mature third-party...
libraries, such as Weka, to guarantee the correctness of classifiers and rankers. Finally, in our empirical studies, all the values of hyper-parameters for classifiers are set to the default values.

5.2 | Threats to external validity

These threats are about whether the observed experimental results can be generalized to other subjects. We only consider web applications projects written in PHP. These data sets are high-quality and have been widely used in previous SVP studies [5,10,22,23]. In the future, we want to consider more commercial and open-source projects in other domains, such as mobile applications.

5.3 | Threats to construct validity

These threats are about whether the performance measures used in the empirical studies reflect the real-world situation. We mainly consider R and IR performance measures [10,23]. In the future, we want to investigate effort-aware performance measures (i.e. ACCand Popt) [40].

5.4 | Threats to conclusion validity

These threats are mainly concerned with the inappropriate use of statistical techniques. To better rank all the feature selection methods in terms of a given performance measure, we use the Scott-Knott test, which has been widely used in previous SDP and SVP studies [13,32,40–43]. In the Scott-Knott test, to better determine whether an SVP model performs significantly better than another SVP model, we also use a statistical analysis method (i.e. Wilcoxon signed-rank test with Benjamini-Hochberg) and an effect size method (i.e. Cliff’s d). These methods have also been widely used in previous studies [22,44–46].

6 | RELATED WORK

Zimmermann et al. [18] first investigated the possibility of SVP by considering traditional metrics in SDP, such as complexity, code churn, dependency measures, and organizational structure of the company. They found that these traditional metrics have statistically significant correlations with the number of vulnerabilities by using Spearman’s rank correlation based on a Windows Vista project. However, the effects of these correlations were small. Then they used Logistic regression to evaluate the prediction performance of SVP. They found that the constructed models have good precision values but low recall values.

Meneely and Williams [37] analysed the relationship between developer activity-based metrics and vulnerabilities. They found that the correlations do exist based on three real-world open-source projects (i.e. the Linux kernel, the PHP programming language, and the Wireshark network protocol analyser). However, the correlations varied and were not very strong. They also evaluated the prediction performance of these metrics by using the Bayesian network classifier. However, precision and recall values were unsatisfactory.

Shin et al. [39] investigated the possibility of SVP by considering traditional metrics (i.e. code complexity, code churn and developer activity). Final results showed that the constructed models could predict 70.8% vulnerabilities by inspecting only 10.9% files for the Firefox web browser and predict 68.8% vulnerabilities by inspecting only 3.0% files for Red Hat Linux kernel. Later, Shin and Williams [38] further found that SDP models can be used to predict vulnerabilities. However, false positives of these models should be reduced while high recall values should be retained.

Walden et al. [5] compared the performance between SM-based methods and text mining-based methods [47,48]. In their empirical studies, they analysed three large-scale open source projects (i.e. Drupal, Moodle and PhpMyAdmin) and shared these data sets to facilitate the follow-up studies. They found that text mining-based methods could achieve better recall values. Based on their shared data sets, Zhang et al. [23] proposed VULPREDICTOR method to improve prediction performance. In particular, they first built six base classifiers based on SM or text mining. Then they constructed a meta classifier to ensemble the outputs of six base classifiers. Empirical results showed the effectiveness of their proposed method. Tang et al. [22] considered code inspection cost and used effort-aware performance measures to evaluate the performance of SVP methods. Empirical results showed that whether in effort-aware ranking-based measures or effort-aware classification based measures, these two different kinds of methods had similar performance. Recently, Stuckman et al. [10] investigated the effect of dimensionality reduction methods for SVP. They considered two kinds of dimensionality reduction methods: feature selection methods and feature synthesis methods (such as PCA). They found using dimensionality reduction could narrow the performance gap between these two kinds of methods.

Based on the above analysis, we can find that using the feature selection to improve the performance of SVP has not been thoroughly analysed. The most relevant study is conducted by Stuckman et al. [10]. The differences between our study and Stuckman et al. [10] can be summarized as follows: (1) We perform feature selection on the combination of SM and TFs, while Stuckman et al. [10] only performed dimensionality reduction on SM and TFs, respectively (2) Stuckman et al. [10] only considered entropy-based feature selection (i.e. information gain), on TFs. Our study considers other new six feature ranking methods. (3) For FRFS methods used by Stuckman et al. [10], they suggested that selecting top-10 or top-100 features can help to achieve better performance. However, we find the optimal feature selection ratio in our study has a certain difference from their suggested value. (4) Except for the performance analysis of different FRFS methods, we also perform diversity analysis on identified vulnerable modules for these FRFS methods.
CONCLUSION AND FUTURE WORK

We design empirical studies to analyse the impact of FRFS on the performance of SVP. Based on the empirical results, our study can bring the following benefits for both academia and industry. First, using FRFS methods can improve the performance of SVP. Therefore, FRFS methods cannot be ignored when optimizing SVP models. Second, when performing SVP, we should use both SM and TFs together, and then should tune the setting in FRFS (such as feature selection ratio, the ranker in the FRFS). Third, there is a performance bottle in state-of-the-art SVP methods. Therefore, more effective methods should be designed in the future, such as deep learning-based methods.

In the future, we plan to extend our research in several ways. First, we want to investigate whether our conclusion can be generalized to other open-source or commercial projects. Then, we should investigate more advanced imbalance learning methods since the class imbalance problem is more serious in SVP when compared to SDP [18]. Except for the within-project SVP scenario considered in our study, the cross-project SVP scenario [49,50] also needs investigation. Cross-project SVP denotes that SVP models are constructed from one project and then used to predict potentially vulnerable modules in another project. This scenario is especially suitable for new projects or projects with limited training data.

ACKNOWLEDGEMENTS

The authors would like to thank the editors and the anonymous reviewers for their insightful comments and suggestions, which can substantially improve the quality of this work. This work is supported in part by National Natural Science Foundation of China (Grant nos. 61702041 and 61202006), Guangxi Key Laboratory of Trusted Software (Grant no. kx202012) and Jiangsu Government Scholarship for Overseas Studies.

REFERENCES

1. Porter, B., McGraw, G.: Software security testing. IEEE Security & Privacy 2(5), 81–85 (2004)
2. Ghaffarian, S.M., Shahriari, H.R.: Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. ACM Comput. Surv. 50(4), 56 (2017)
3. Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S.: A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Software Eng. 38(6), 1276–1304 (2012)
4. Kamei, Y., Shihab, E.: Defect prediction: Accomplishments and future challenges. Proc. Int. Conf. Software Anal. Evol. Reengineering, 5, 33–45 (2016)
5. Walden, J., Stuckman, J., Scandarolli, R.: Predicting vulnerable components: Software metrics vs text mining. In: Proceedings of international symposium on software reliability engineering, pp. 23–33. (2014)
6. Ghosh, B., McInerney, S., Hassan, A.E.: A large-scale study of the impact of feature selection techniques on defect classification models. In: Proceedings of the international conference on mining software repositories, pp. 146–157 (2017)
7. Xu, R., Zhang, X., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. IEEE Trans. Evol. Comput. 20(4), 606–626 (2016)
8. Gao, K., Khoshgoftaar, T.M., Wang, H., Selviya, N.: Choosing software metrics for defect prediction: an investigation on feature selection techniques. Software Pract. Ex. 41(5), 579–606 (2011)
9. Liu, S., Chen, X., Liu, W., et al.: A feature selection framework for software defect prediction. In: Proceedings of the 2014 IEEE 38th Annual computer software and applications conference, pp. 426–435 (2014)
10. Stuckman, J., Walden, J., Scandarolli, R.: The effect of dimensionality reduction on software vulnerability prediction models. IEEE Trans. Reliab. 66(1), 17–37 (2017)
11. Ni, C., Chen, X., Xia, X., Gu, Q., Zhao, Y.: Multitask defect prediction. J. Software: Evol. Process. 31(12), e2203 (2019)
12. Chen, X., Zhang, D., Cui, Z.-Q., Gu, Q., Ju, X.-L.: Dp-share: Privacy-preserving software defect prediction model sharing through differential privacy. J. Comput. Sci. Technol. 34(5), 1020–1038 (2019)
13. Chen, X., Zhang, D., Zhao, Y., Cui, Y., Ni, C.: Software defect number prediction: Unsupervised vs supervised methods. Inf. Software Technol. 106, 161–181 (2019), Palmed Partial Author title Volume Page
14. Ni, C., Liu, W.-S., Chen, X., Gu, Q., Chen, D.-X., Huang, Q.-G.: A cluster based feature selection method for cross-project software defect prediction. J. Comput. Sci. Technol. 32(6), 1090–1107 (2017)
15. Liu, W., Liu, S., Gu, Q., Chen, J., Chen, X., Chen, D.: Empirical studies of a two-stage data preprocessing approach for software fault prediction. IEEE Trans. Reliab. 65(1), 38–53 (2015)
16. Yuan, Z., Chen, X., Cui, Z., Mu, Y.: Altra: Cross-project software defect prediction via active learning and tradaboost. IEEE Access. 8, 30037–30049 (2020)
17. Chen, D., Chen, X., Li, H., Xie, J., Mu, Y.: Deepdpq: Deep learning based cross-project defect prediction. IEEE Access. 7, 184832–184848 (2019)
18. Zimmermann, T., Nagappan, N., Williams, L.: Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In: Proceedings of international conference on software testing, verification and validation, pp. 421–428 (2010).
19. Ni, L., Chen, X., Wu, F., Shen, Y., Gu Q.: An empirical study on pareto based multiprojective feature selection for software defect prediction. J. Syst. Software. 152, 215–238 (2019)
20. Yu, Q., Qian, J., Jiang, S., Wu, Z., Zhang, G.: An empirical study on the effectiveness of feature selection for cross-project defect prediction. IEEE Access. 7, 35710–35718 (2019)
21. Xu, Z., Liu, J., Yang, Z., An, G., Jia, X.: The impact of feature selection on defect prediction performance: an empirical comparison. In: Proceedings of 2016 IEEE 27th international symposium on software reliability engineering, IEEE, pp. 309–320 (2016)
22. Tang, Y., Zhao, F., Yang, Y., Lu, H., Zhou, Y., Xu, B.: Predicting vulnerable components via text mining or software metrics? an effort-aware perspective. In: Proceedings of 2015 IEEE international conference on software quality, reliability and security, IEEE, pp. 27–36 (2015)
23. Zhang, Y., Lo, D., Xia, X., Xu, B., Sun, J., Li, S.: Combining software metrics and test features for vulnerable file prediction. In: Proceedings of international conference on engineering of complex computer systems, 40–49 (2015)
24. Liu, H., Setiono, R.: Chi2: Feature selection and discretization of numeric attributes. In: Proceedings of the international conference on tools with artificial intelligence, pp. 388–391 (1995)
25. Polat, K., Güneş, S.: A new feature selection method on classification of medical datasets: Kernel f-score feature selection, Expert Syst. Appl. 36(7), 10367–10373 (2009)
26. Quinlan, J.R., C4.5: programs for machine learning, Elsevier (2014)
27. Cover, T.M., Thomas, J.A., Elements of information theory. John Wiley & Sons (2012)
28. Li, J., Cheng, K., Wang, S., et al: Feature selection: A data perspective, ACM Comput. Surv. 50(6), 94 (2018)
29. Kononenko, I.: Estimating attributes: analysis and extensions of relief. In: Proceedings of the International Conference on European conference on machine learning, pp. 171–182 (1994)
30. Liaw, A., Wiener, M.: Classification and regression by random forest, R News, 2 (3), 18–22 (2002)
31. Jelihovschi, E.G., Faria, J.C., Allaman, I.B.: Scottknott: A package for performing the Scott-Knott clustering algorithm in R. TeMA. 15(1), 3–17 (2014)
32. Ghotra, B., McIntosh, S., Hassan, A.E.: Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proceedings of the 37th international conference on software engineering. IEEE Press, pp. 789–800 (2015)
33. Wilcoxson, F.: Individual comparisons by ranking methods. In: Breakthroughs in statistics, pp. 196–202. Springer (1992)
34. Ferreira, J., Zwinderman, A.: On the Benjamini–Hochberg method. Ann. Stat. 34(4), 1827–1849 (2006)
35. Benjamini, Y., Hochberg, Y.: Controlling the false discovery rate: A practical and powerful approach to multiple testing. J. Roy. Stat. Soc. B. 57(1), 289–300 (1995)
36. Dietterich, T.G.: Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput. 10(7), 1895–1923 (1998)
37. Meneely, A., Williams, L.: Strengthening the empirical analysis of the relationship between linus’ law and software security. In Proceedings of the international symposium on empirical software engineering and measurement, p. 9 (2010)
38. Shin, Y., Williams, L.: Can traditional fault prediction models be used for vulnerability prediction? Empir. Software Eng. 18(1), 25–59 (2013)
39. Shin, Y., Meneely, A., Williams, L., Osborne, J.A.: Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Trans. Software Eng. 37(6), 772–787 (2011)
40. Chen, X., Zhao, Y., Wang, Q., Yuan, Z.: Multi: Multi-objective effort-aware just-in-time software defect prediction. Inf. Software Technol. 93), 1–13 (2018)
41. Taninthamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K.: Automated parameter optimization of classification techniques for defect prediction models, in Proceedings of the 38th international conference on software engineering, pp. 321–332 (2016)
42. Chen, X., Mu, Y., Qu, Y., et al.: Do different cross-project defect prediction methods identify the same defective modules? J. Sofw. Evol. Proc. 32(5), e2234 (2020)
43. Chen, X., Zhao, Y., Cui, Z., Meng, G., Liu, Y., Wang, Z.: Large-scale empirical studies on effort-aware security vulnerability prediction methods. IEEE Trans. Reliab. 69(1), 70–87 (2020)
44. Yang, Y., Zhou, Y., Liu, J., et al.: Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models. In: Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering, pp. 157–168 (2016)
45. Fu, W., Menzies, T.: Revisiting unsupervised learning for defect prediction. In: Proceedings of the 2017 11th joint meeting on foundations of software engineering, pp. 72–83 (2017)
46. Rahman, F., Devanbu, P.: How, and why, process metrics are better. In: Proceedings of 2013 35th international conference on software engineering. IEEE, pp. 432–441 (2013)
47. Hovsepyan, A., Scandariato, R., Joosen, W., Walden, J.: Software vulnerability prediction using text analysis techniques. In: Proceedings of the international workshop on Security measurements and metrics, pp. 7–10 (2012)
48. Scandariato, R., Walden, J., Hovsepyan, A., Joosen, W.: Predicting vulnerable software components via text mining. IEEE Trans. Software Eng., 40(10), 993–1006 (2014)
49. Xu, Z., Yuan, P., Zhang, T., Tang, Y., Li, S., Xia, Z.: Hda: Cross-project defect prediction via heterogeneous domain adaptation with dictionary learning. IEEE Access, 6, 57597–57613 (2018)
50. Li, Y., Huang, Z., Wang, Y., Fang, B.: Evaluating data filter on cross-project defect prediction: Comparison and improvements. IEEE Access. 5, 25646–25656 (2017)

How to cite this article: Chen X, Yuan Z, Cui Z, Zhang D, Ju X. Empirical studies on the impact of filter-based ranking feature selection on security vulnerability prediction. IET Sofw. 2021;15:75–89. https://doi.org/10.1049/sfw2.12006