Software Quality Journal (2021) 29:509–553
https://doi.org/10.1007/s11219-020-09546-7

An improved text classification modelling approach to identify security messages in heterogeneous projects

Tosin Daniel Oyetoyan[1,2] · Patrick Morrison[3]

Accepted: 26 December 2020 / Published online: 27 May 2021

Abstract
Security remains under-addressed in many organisations, illustrated by the number of large-scale software security breaches. Preventing breaches can begin during software development if attention is paid to security during the software's design and implementation. One approach to security assurance during software development is to examine communications between developers as a means of studying the security concerns of the project. Prior research has investigated models for classifying project communication messages (e.g., issues or commits) as security related or not. A known problem is that these models are project-specific, limiting their use by other projects or organisations. We investigate whether we can build a generic classification model that can generalise across projects. We define a set of security keywords by extracting them from relevant security sources, dividing them into four categories: asset, attack/threat, control/mitigation, and implicit. Using different combinations of these categories and including them in the training dataset, we built a classification model and evaluated it on industrial, open-source, and research-based datasets containing over 45 different products. Our model based on harvested security keywords as a feature set shows average recall from 55 to 86%, minimum recall from 43 to 71% and maximum recall from 60 to 100%. An average f-score between 3.4 and 88%, an average g-measure of at least 66% across all the dataset, and an average AUC of ROC from 69 to 89%. In addition, models that use externally sourced features outperformed models that use project-specific features on average by a margin of 26–44% in recall, 22–50% in g-measure, 0.4–28% in f-score, and 15–19% in AUC of ROC. Further, our results outperform a state-of-the-art prediction model for security bug reports in all cases. We find using sound statistical and effect size tests that (1) using harvested security keywords as features to train a text classification model improve classification models and generalise to other projects significantly. (2) Including features in the training dataset before model construction improve classification models significantly. (3) Different security categories represent predictors for different projects. Finally, we introduce new and promising approaches to construct models that can generalise across different independent projects.

Keywords Security · Classification model · Text classification · Software repository · Machine learning

Tosin Daniel Oyetoyan
tosin.daniel.oyetoyan@hvl.no

Extended author information available on the last page of the article

1 Introduction

Security breaches have become regular occurrences, with devastating consequences and costs to organisations and society (Ponemon-Institute, IBM-Security, 2017). Unfortunately, security as a non-functional requirement is under-addressed in many software development projects. Cois and Kazman (2015) analysed 400,000 projects on GitHub and found only 1.4% to explicitly identify security-related issues. It is therefore relevant to understand the security picture of a project by assessing product security and the project's management capabilities (Ellison et al., 2010).

Researchers (Cois & Kazman 2015; Cleland-Huang et al., 2006; Hindle et al., 2013; Ray et al., 2016) have investigated security concerns in software management repositories (e.g. Issue Trackers and Version Control Systems), seeking relevant quantitative measures that could be derived from security analysis of software management repositories. Such measures could assist project managers and development teams in taking informed decisions regarding the security posture of a project by providing answers to, e.g., How many security-related changes have been made in the system? How many security-related bugs are left unresolved? What is the average window-of-exposure (in days) for security-related issues in a project? However, these studies are project specific, and we do not know how their results generalise beyond the environments studied.

Researchers have constructed text classification models by building a document term matrix (DTM) from a project's document corpus and selecting promising features from the DTM(Sparck Jones, 1972; Salton et al., 1983; Salton & McGill, n.d.; Salton and Buckley 1988; Wu et al., 2008). This approach limits the model to the environment where it is constructed. (Gegick et al., 2010) advised against applying a text classification model for identifying security messages trained on one system to a different system. Based on their findings, cross-project classification suffered significant performance reduction. In (Anvik et al., 2006), the precision of their model decreased from 64 to 6% with low recalls in general when applied to another project's dataset thus showing the task to develop a generic text classification model that could be practically useful across different systems to be non-trivial.

Many benefits could be derived from a model that works across heterogeneous projects. First, it could eliminate the need to create a unique model for each project in an organisation. Organisations sometimes maintain several hundred projects, making development of classification models messy and difficult. Second, in many projects and small businesses (Unterkalmsteiner et al., n.d.), time-to-market is critical, and as a result, security activities are not prioritised. An off-the-shelf model that can be quickly leveraged to identify security-related issues in such projects would reduce the time and effort required to build a model for each project. Third, thid could provide the research community with a benchmark for security-related issues across heterogeneous projects and organisations.

The goal of this work is to investigate the hypothesis that a generic text classification model can be developed for classifying security-related messages in software development project communications. Generalising to unseen datasets is the goal of a classification model. Typical text classification models in the security domain are usually constructed based on an oracle dataset (Nigam et al., 2000) containing experts' (human/system) classifications of a sample of the population that a classification algorithm should learn. The requirement for an oracle presents a limitation to generalising to unseen data.

We address the problem of generalisation by augmentation of the training set with security keywords, using these terms as features to train a text classification model. Our research question is:

RQ: Does the performance of our classification model outperform models built using project-specific terms when generalising to new projects? In other words, can we build a model on one project and transfer it to another project using transfer project prediction (TPP) approach (e.g., Peters et al., 2017)?

Our contributions in this paper are:

- A new and promising approach to weighting and training a generalised text classification model for identifying security messages across projects.
- A list of security terms that can be used independently to identify security-related issues in any repository.
- A new approach of using security terms divided into different categories to build classification models.
- A security classifier framework and a add-on for the JIRA[1] platform, based on our best-performing trained model.

The remainder of the paper is organised as follows: In Sect. 2, we discuss studies related to our work. In Sect. 3, we describe our dataset selection and analysis approach to answer our research questions. In Sect. 4, we present the results of our study. We provide discussion in Sect. 5. We discuss various threats to the validity of our study in Sect. 6. Lastly, we conclude in Sect. 7.

2 Related work

Researchers have applied text classification approaches to topics such as text filtering, document categorisation, automatic text indexing, and to various text sources such as the World Wide Web, Internet news feeds, electronic mail, corporate databases, medical patient, records, and digital libraries (Nigam et al., 2000; Sebastiani, 2002). These approaches leverage supervised (Debole & Sebastiani, 2004) or unsupervised learning techniques (Le & Mikolov, 2014). A supervised technique uses an already labelled dataset to train a classification algorithm. In an unsupervised approach, a dataset is labelled using certain heuristics such as distance measures to cluster related texts (Yan et al., 2017).

Our problem is one of text classification for predicting security messages. We discuss related works in text classification and specifically its application to predicting security messages in software development repositories such as issue tracking systems and version control systems. Researchers have applied a variety of approaches to identify and characterise security-related messages in software development repositories. A problem in this domain is the need for a labelled and large enough training dataset sample from which a classifier can learn accurately (Nigam et al., 2000).

One approach for selecting a training dataset is to select a sample of text from a corpus by using a set of security keywords (Cleland-Huang et al., 2006; Pletea et al., 2014; Ray et al., 2014; Pletea et al., 2014) mined security discussions in commits and pull requests

[1] https://github.com/tosdanoye/fsecext and https://bitbucket.org/ootos/jirasecplugin/downloads/

in GitHub by using a selected set of security keywords and then used the dataset to train a classifier to understand emotions between security-related and non-security-related discussions. Ray et al. (2014) used a restricted set of keywords covering different quality attributes (e.g. security) to select training dataset from commit messages in GitHub as a sample representation to determine the code quality of different programming languages. We have applied text classification to develop a generic model for identifying security-related messages in different software development projects.

An alternative to generic security keyword lists is to mine them from existing repositories where security issues have been labelled. Such training sample can be obtained from labelled messages by developers, security champions in an organisation, or security researchers (Gegick et al., 2010; Ohira et al., 2015; Riaz et al., 2014). This approach can be fraught with subjectivity and inaccuracies (Gegick et al., 2010); (Scandariato et al., 2014). Existing training samples can also be mined from security advisory reports (Zaman et al., 2011) (e.g. Mozilla Firefox Security Advisor, Apache Security Advisories, or CVEs), or mined from reports generated by a static analysis tool (Scandariato et al., 2014). The MITRE CWE framework has also been used to manually categorise and label security-related messages (Tyo, 2016).

Gegick et al. (2010) used text classification models constructed on manually labelled bug reports to identify security bug reports that are manually mislabelled as non-security bug reports in a large Cisco project. Their approach was based on document term matrix construction to identify classification features. Their model successfully identified 78% security bug reports previously misclassified as non-security bug reports by the Cisco engineers with 98% probability. However, the model performance decreased significantly when applied on other systems.

Wijayasekara et al. (2014) applied text mining to the Linux bug database to detect 'hidden impact' bugs (HIB). Their approach is based on term-frequency determination from the Linux bug reports and mapping to the publicly disclosed vulnerability data from MITRE CVE within the same period. Our study differs from Gegick et al. (2010) and Wijayasekara et al. (2014) by exploiting security domain knowledge to extract promising features from relevant security sources.

Cois and Kazman (2015) developed text classifiers that could identify security-related messages in issue trackers and applied their models on the large-scale open source Chromium software project. They have used features constructed as *n*-grams from the text artefacts of Chromium and recorded an average of over 85% in recall, precision and f-score. Our approach differs from Cois and Kazman (2015) because we have manually defined our set of features whereas they have used terms obtained automatically from the project's text artefacts.

Zaman et al. (2011) studied the characteristics of security vs. performance bugs in terms of fix rates, change rates, and who fixes the bugs. They have used security bugs from the Mozilla Foundation Security Advisory to represent a security dataset. Our study aims to develop a generic model for identifying security-related messages in bug reports and related repositories.

Riaz et al. (2014) manually classified requirement documents in the healthcare domain to identify security-relevant requirements. They applied machine-learning algorithm to develop automated tools built on their previously classified oracle dataset to assist requirement engineers during analysis. The results show 79% recall and 82% precision.

Cleland-Huang et al. (2006) developed a text classifier by using term-frequency to identify non-functional requirements from requirement specification documents that were developed by MS students. The result of the model when applied on an industrial dataset with terms from initial documents suffered performance reduction. Their results significantly improved when the terms were augmented with terms from the industrial dataset.

Hindle et al. (2013) performed topic labelling of various non-functional requirements using semi-unsupervised and supervised techniques on three large-scale relational databases. They defined a set of generic keywords to build models that could generalise on the different projects. Their results show an average f-score between 39 and 48% and a range between 4 and 90%. Unlike the studies above that addressed a broad range of non-functional requirements, we have focused primarily on security.

Scandariato et al. (2014) applied a static code analysis tool to collect security-related records of 20 Android applications. By applying text mining approach to software source code, they formulated their features as a set of source components (actual source files) with their associated frequencies and then built a prediction model to predict which component is vulnerable or not. This study is based on mining the actual source code which is different from our work that focuses on the natural language contained in the software development repositories.

Peters et al. (2017) observed that the presence of security-related keywords in both security and non-security bug reports (SBR, NSBR) can lead to mislabelling of SBRs and reduce prediction performance on within project prediction (WPP) and transfer project prediction (TPP). They proposed a framework, FARSEC, to filter and rank bug reports to reduce NSBRs with security-related keywords before building a prediction model. They applied their framework to data and security classifications drawn from Ohira et al. (2015) and from the Chromium project. Their results showed a 38% reduction in mislabelled SBRs. The goal of our work is similar to Peters et al. (2017). However, on the one hand, our work is different from Peters et al. (2017) in the approach we have used to construct a generalised model. While Peters et al. (2017). uses filtering techniques on the dataset, we have used a harvested set of security keywords for our models. On the other hand, we investigate how to improve model's performance by increasing the weight of security terms in the training dataset thereby showing some similarities to their work.

3 Methodology

In this section, we first present and describe our datasets. Next, we present project-specific approach for extracting features to train a text classification model for a project. Then, we present our proposed generalised modelling approach. Finally, we describe our experiment setup, the metrics and our approach for answering the research questions.

3.1 Dataset selection

For testing our models, we have used published datasets from nine different sources (Table 1), containing messages that have been manually tagged as security related. We now describe the bug report (BR) dataset we have used in this study.

1) Industrial systems (Comm): We collected data from two projects recorded in the JIRA production environment of one of our industrial partners. The organisation has a security champion and has created a security field in JIRA where they manually label created issues (bug, story, task, or improvement) as security-related or not. This classification

Table 1 Datasets used in our study

Project	Domain	Start date	End date	No. of BRs	No. of SBR	SBR (%)
Ambari	Hadoop management web UI backed by its RESTful APIs	Sep 26 2011	Aug 8 2014	1000	29	2.9
Wicket	Component-based web application framework for Java programming	Oct 20 2006	Nov 9 2014	1000	10	1.0
Camel	A rule-based routing and mediation engine	Jul 8 2007	Sep 18 2013	1000	32	3.2
Derby	A relational database management system	Sep 28 2004	Sep 17 2014	1000	88	8.8
Chromium	Web browser	Aug 30 2008	Jun 11 2010	41,940	192	0.5
Apache	Different products	–	–	2566	1283	50
Mozilla	Different web browsers	–	–	6552	3276	50
Comm	Ticketing software	–	–	946	20	2.1
ODCV	Web browsers	–	–	1166	583	50

has been applied on one of the projects. We downloaded 946 records of which 20 are labelled as security related.

2) The high-impact bug dataset by Ohira et al. (2015) (HIB): This publicly available dataset has been manually classified by a team of researchers to allow its use for research studies. The four projects covered by this dataset are: Apache Camel, Apache Derby, Apache Ambari and Apache Wicket. The dataset contains a total of 4000 records with 160 records manually labelled as security related.
3) Chromium dataset: The chromium dataset has been prepared and used in Peters et al. (2017) The dataset is available online[2], and we have downloaded the entire dataset and reuse as-is in our environment.
4) Apache projects dataset (Apache): We downloaded data from Apache JIRA open-source projects where developers have manually labelled their recorded issues as security related. We search the apache Jira URL at 'https://issues.apache.org/jira' and use the search criteria 'labels = security OR labels = 'security' OR labels = security-issue'. The query yields a total of 1283 records for different projects in Apache. The records form our security bug reports (SBR). We separately downloaded 10,000 records for 10 different products using the search criteria 'labels is EMPTY' from which we randomly extracted 1283 records as our non-security-related bugs (NSBRs).
5) Mozilla dataset (Mozilla): We downloaded security bugs from the Mozilla bug tracking system at 'https://bugzilla.mozilla.org' by using the search criteria 'group: security' yielding a total of 3276 security bug reports (SBR). We separately downloaded 10,000 records for 31 different products using the search criteria 'status: all' from which we randomly extracted 3276 records as our non-security-related bugs (NSBR)
6) Orthogonal defect classification-vulnerabilities (ODCV)(Morrison et al., 2018a) is a publicly available dataset (https://sites.google.com/a/ncsu.edu/odc-v/) of 583 security vulnerabilities and 583 non-security defects collected from three open-source projects (Chrome, Firefox, phpMyAdmin), classified according to a security-enhanced version of Orthogonal Defect Classification (Chillarege et al., 1992).

3.2 Design and analysis approach

3.2.1 Feature selection approaches

In this section, we describe three feature selection approaches that we have used in our experiments:

3.2.1.1 tf and its variants—project-specific modeling approach Term frequency (tf) is a standard metric for determining features in document classification problems (Manning et al., 2008). Term frequency is computed by tokenizing each document in a document corpus and collecting the number of occurrence (frequency) of the unique terms in the entire corpus. Subsequently, a document term matrix (DTM) can then be constructed from the document corpus using these terms. As an example, if we have three documents namely:

D1 = 'owasp top software errors'

[2] https://bitbucket.org/fayola21-lero/farsec47/src/master/resources/data/scrubbed/

D2 = ‘buffer overflow is dangerous’

D3 = ‘software errors include buffer overflow errors’. A DTM is represented by tokenizing the documents and generating a term frequency for each text token as:

$$DTM = \begin{bmatrix} & owasp & top & software & errors & buffer & overflow & is & dangerous & include \\ D_1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ D_2 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ D_3 & 0 & 0 & 1 & 2 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

where the topmost row represents the unique tokens in the entire document corpus, and the leftmost column represents each document. The values in the matrix represent the frequency of terms (number of times the term appears). An algorithm can then use term weighting or normalisation techniques (Manning et al., 2008; Sparck, 1972; Forman, 2003; Sebastiani, 2002) to determine the most interesting terms that could represent each document from the document corpus.

In this study, we have considered the normalised variant of term frequency-inverse document frequency (tf-idf) (Manning et al., 2008) as used in Peters et al. (2017).

The normalised *tf-idf* is computed as:

Normalised term frequency: $ntf_{t,d} = 0.5 + \frac{0.5 \times tf_{t,d}}{max_t(tf_{t,d})}$ (1)

where $tf_{t,d}$ represents the number of times, term t appears in document d, and max_t represents the frequency of the term with the maximum occurrence in document d. The smoothing term, 0.5, is used to avoid a large swing in $ntf_{t,d}$ from a small change in $tf_{t,d}$.

Inverse document frequency: $idf_t = log\frac{N}{df_t}$ (2)

where N = number of documents in the entire document corpus, and df_t represents the number of documents where term t appears. The intuition behind *idf* is that terms that are frequent in all documents may not discriminate very well and will thus be penalised with low *idf*. Conversely, terms that occur in a few documents may be more interesting for the documents where they appear and will thus be weighted with higher *idf*.

Term frequency-inverse document frequency: $tf - idf_t = ntf_{t,d} \times idf_t$ (3).

The *tf-log* is computed as

Term frequency (Logarithm): $tf - log_t = 1 + \log(tf_{t,d})$ (4).

A project-specific approach can use any of *tf*, *ntf*, *tf-idf* or *tf-log* metrics based on a threshold (e.g. top *n* terms) to determine the most promising features. For example, Peters et al. (2017) use the top 100 terms with the highest *tf-idf* values as their feature set.

3.2.1.2 Our generalised modelling approach In this approach, we first manually define security keyword list by extracting security keywords from popular and relevant security sources as depicted in Fig. 1. The feature set used in this work is based primarily on keywords

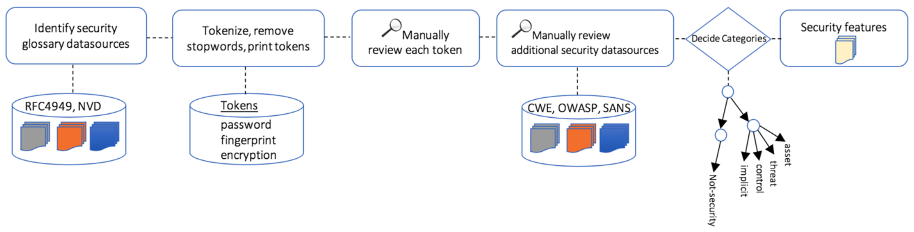
Identify security glossary datasources
Tokenize, remove stopwords, print tokens
Manually review each token
Manually review additional security datasources
Decide Categories
Security features
RFC4949, NVD
Tokens
password
fingerprint
encryption
CWE, OWASP, SANS
Not-security
implicit
control
threat
asset

Fig. 1 Extracting security feature list from security data sources

extracted from the RFC4949[3] (Internet Security Glossary) and NVD[4] datasets. We also used MITRE CWE,[5] OWASP,[6] SANS,[7] and programming language exceptions (e.g. PHP[8]). We parsed and tokenized the full text removing English stop words from the list and manually reviewing the collected terms. We reviewed the other listed sources (CWE, OWASP and SANS) manually for security keywords by reviewing their high-level documents. For example, with CWE, we mainly reviewed the 'Research Concepts' view where the graph shows high-level description of the CWE errors. Similarly, for OWASP, we reviewed annotated categories such as attack and control, and for SANS, we reviewed the top 25 software errors and extracted relevant terms. Next, we classified the keywords into four categories namely:

- **Assets:** set of terms that represents data or information that should be protected by an entity (e.g. PII, credit card, ecash, token, keys, and session)
- **Control:** list of terms that represents implemented security controls or mitigations (e.g. authentication, checksum, captchas and encryption)
- **Attack/threats:** list of terms that represents security attacks or threats to a system (e.g. xss, backdoor, injection, vulnerability, untrusted data)
- **Implicit:** list of terms that does not fall under these categories and many times are implicit security-related terms (e.g. incorrect, wrong, invalid, restrict, null pointer exception)

Our approach relies on this set of keywords that we have defined to form a feature set. A key point in machine learning is that better features tend to outperform a cleverer algorithm (Sparck, 1972; Salton & Buckley, 1988; Wu et al., 2008; Forman, 2003). Domain knowledge can improve data knowledge and representation and thus improve the model. In addition, the feature space can be reduced with less computationally expensive modelling operations (Wijayasekara et al., 2014) than using frequent terms computed from a document term matrix (Feinerer, 2013).

3.2.1.3 Classification models and performance metrics Based on the approaches described in Sects. 3.2.1.1 and 3.2.1.2, we derive four different classification models based on two feature selection methods (see Fig. 2):

1. Model *fsec-ext+* uses the harvested security keywords as feature set and includes security keywords from the list of attack/threat, control, asset and implicit terms as records in the training dataset. This approach aims to increase the weight of the security terms by including them in our training set and setting them as SBRs (see Table 2).
2. Model *fsec-tfidf* uses the *tf-idf* metric (Eq. 3) and preset to top 100 as used in Peters et al. (2017) This metric is project specific as terms are determined from the security bug reports of the project. Following the argument that the top 100 terms span nearly all the feature families of the vulnerability reports (Bozorgi et al., 2010), we adopt these terms for our experiments with the aim to also be able to compare our results.

[3] https://tools.ietf.org/html/rfc4949

[4] https://nvd.nist.gov/vuln/data-feeds#XML_FEED

[5] https://cwe.mitre.org/data/definitions/1000.html

[6] https://www.owasp.org/index.php/Category:Attack, https://www.owasp.org/index.php/Category:Control

[7] https://www.sans.org/top25-software-errors/

[8] https://www.php.net/manual/en/spl.exceptions.php

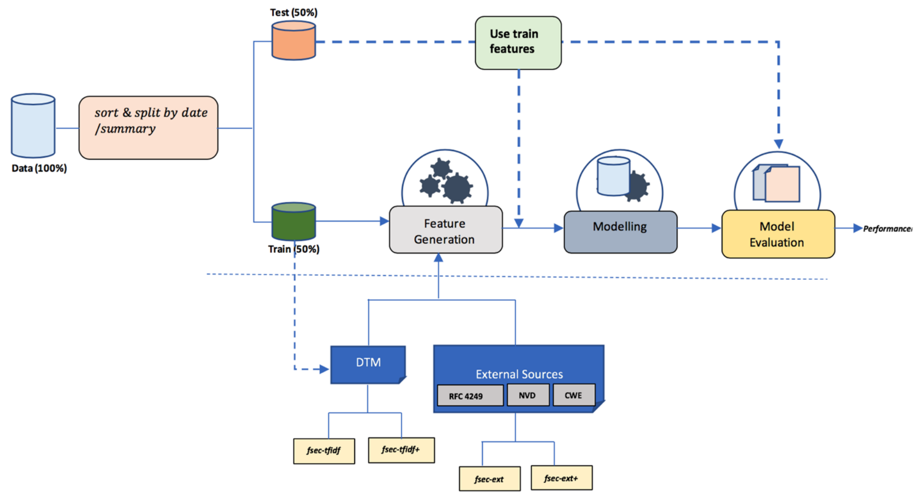

Fig. 2 Experiment setup and modelling approach

3. Model *fsec-ext* uses the harvested security keywords as feature set as in *fsec-ext*+ model but exclude the features from the training dataset.
4. Model *fsec-tfidf*+ uses the *tf-idf* metric (Eq. 3) as in *fsec-tfidf* but includes the features in the training dataset.

3.2.1.4 Performance metrics We report the recall, precision, probability of false (pf) alarm, f-score, g-measure and area under the curve of receiver operating curve (Powers, 2011) (aucroc) to show the performance of the text classification models (Sebastiani, 2002; Peters et al., 2017). We included pf and g-measure in order to compare our work with Peters et al. (2017). The g-measure provides the harmonic mean between recall (true positive rate) and true negative rate (100—pf) while the f-score measures the harmonic mean between recall and precision. The roc measures the relationship between true

Table 2 Example of text and security keywords added to form our training dataset

Text/feature	Label
backdoor	1
password	1
CAPTCHAS	1
cipher	1
security	1
CSS style Parser does not handle style names with leading dash Consider the 0 following fragment:	0
ldap injection	1
Authentication with ZeppelinHubRealm throws 403 error In some cases, depending where authentication server is deployed and what services are used, it is possible that httpcomponents client request headers are not approved by the security rules…	1
Font size to paragraph config font size for paragraph text	0

positive rate and false positive rate (aka pf) of a classifier at different thresholds, and the auc computes a single metric that provides the area of a classifier under the roc. This allows a fair comparison of the performance of one model to another. The metrics are computed from true positive (TP), true negative (TN), false positive (FP), and false negative (FN) where:

TP = number of security records correctly identified as security records
TN = number of non-security records correctly identified as non-security records
FP = number of non-security records incorrectly identified as security records
FN = number of security records incorrectly identified as non-security records
as follows

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$pf = \frac{FP}{FP + TN}$$

$$F - score = 2 \times \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$g - measure = \frac{2 \times \text{Recall} \times (100 - pf)}{\text{Recall} + (100 - pf)}$$

3.2.1.5 Constructing training and test matrices from feature set We use the hand-crafted keywords (custom dictionary list) for *fsec-ext* model as the feature vector. In the case of *fsec-tfidf*, the 100 terms forming the dictionary list is generated from SBR of the training dataset as used in Peters et al. (2017). In both cases (*fsec-ext* and *fsec-tfidf*), the derived features are used to construct the training and test matrices as shown in Fig. 2. We use the *tf-log* metric (Eq. 4) when a word matches the feature vector otherwise, we record a ‘0’. The example below demonstrates how the document-term matrix is constructed using a predetermined dictionary list.

Assume a document contains 2 texts with classification categories as follows:

$$\begin{bmatrix} \text{Text} & \text{Classification} \\ \text{Diffucult to place caret at the end of a line in Show All Tags view} & 0 \\ \text{Allow Personal Security Password } > 30 \textit{ characters} & 1 \end{bmatrix}$$

Given a set of security keywords, F determined by *fsec-ext*, or *fsec-tfidf*, such that F = {password, xmli, security, overflow}, we arrive at vectors:
V(text1)$^{\mathrm{F}}$ = [0,0,0,0]
V(text2)$^{\mathrm{F}}$ = [1,0,1,0]
and a matrix: $M\left(text_1, text_2\right) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

3.2.1.6 Text cleaning and transformation approaches We have used the four datasets from Ohira and the cleaned and scrubbed version of Chromium dataset from Peters et al. (2017). In addition to the four datasets described in Sect. 3.1, we remove stop words from all the nine datasets and remove unwanted terms from Chromium and the HIB dataset using the list from Peters et al. (2017). We then collect only text data from each dataset using the regular expression [^a-z A-Z]. After these steps, we tokenise the text and transform the documents (SBR or NSBR) into a document term matrix as described in 3.2.1.5.

3.2.1.7 Generating training and test dataset We split the Ohira dataset and Chromium dataset by using date and exactly as reported in (Peters et al., 2017) into 50% past and 50% present. The training dataset represents the past while the test dataset represents the present. In the case of ODCV, Apache, Mozilla, and Comm dataset, we sort by the 'Summary' field and split into 50% training and 50% test dataset. Table 3 lists the properties of our training and test dataset in terms of the number of security bug reports (SBR) and non-security bug reports (NSBR). The test dataset is unseen data that is used to evaluate the true performance of the constructed model.

3.2.1.8 Class imbalance and sampling techniques Wicket, Derby, Camel, Ambari, Chromium and Comm datasets are highly skewed with SBRs accounting for 1%, 8.8%, 3.2%, 2.9%, 0.5%, and 2.1% of the total bug reports respectively. Imbalance data is known to reduce the performance of classifier on the class where the data is sparse. The BR is highly skewed with very few reports labelled as SBR. We deal with this imbalance in our training dataset by either over sampling or under sampling (He & Garcia, 2009) the NSBR depending on the ratio of SBR to NSBR in the project. We treat sampling ratio of NSBR to SBR in our training dataset as a treatment in our experiment to determine which sampling ratio can produce adjusted training dataset that can best be learned by the classifier. In other words, we do not change the size of the SBR in our training dataset; rather, we adjust the size of NSBR by a factor of SBR as listed in Algorithm 1. In line 6, instances from NSBR are randomly picked by using a random number that returns an integer bounded by the size of NSBR.

Table 3 Properties of the training and test data

Project			Training		Test	
	No. of BRs	No. of SBR	No. of training	No. of SBR in training	No. of test	No. of SBR in test
Ambari	1000	29	500	22	500	7
Wicket	1000	10	500	4	500	6
Camel	1000	32	500	14	500	18
Derby	1000	88	500	46	500	42
Chromium	41,940	192	20,970	77	20,970	115
Apache	2566	1283	1283	664	1283	619
Mozilla	6552	3276	3276	1664	3276	1612
Comm	946	20	473	6	473	14
ODCV	1157	571	579	128	578	443

Algorithm 1: OverOrUnderSample NSBR in Training Dataset

```
balanceNSBR (SBR, NSBR, factor) [factor is of primitive type double and > 0]
add (datastore, SBR)
if NSBR.size > SBR.size or NSBR.size < SBR.size then
    adjustedSize = factor * SBR.size
    for 1 to adjustedSize do
        add(datastore, NSBR[RandomInteger(NSBR.size)])
    end for
else // SBR == NSBR
    add (datastore, NSBR)
return datastore
```

3.2.1.9 Training parameters and treatments Some terms must be weighted higher than the others. Term weighting is a known technique in machine learning (Salton & Buckley, 1988; Wu et al., 2008). The objective is to allow the algorithm to learn a function that prioritises the term or feature with a higher weight. Our approach is to give a higher weight to the security features (attack/threat, asset, control, and implicit terms). To achieve this, we classify every term in our feature set to be security related. These terms are then included as SBRs in our training dataset as shown in Table 2.

For the *fsec-ext* feature selection method, we experiment with using different combinations of security category such as attack/threat (T), asset (A), control (C) and implicit (I) terms to train our classification model. We devise seven combinations from these categories (see Table 4)—T, C, TC, TA, CA, TCA, TCAI. The intuition is that different projects may report on different security category which in turn may influence the quality of the learner. In addition, this approach can provide insight into the dominant security category and issues for a project. Furthermore, we experiment with including and excluding these security categories as SBRs in our training dataset. Lastly, we experiment with class imbalance ratio by adjusting (over- or undersampling) NSBR size relative to the size of SBR. For instance, a value of 0 means the original size of SBR to NSBR in the training dataset is kept, and a value of 1.0 means the ratio SBR:NSBR is 1:1. Table 4 lists the parameters and the corresponding values we have adopted in our experiments.

3.2.1.10 Experiment setup and modelling approach Figure 2 shows our experiment setup and modelling approach. To answer our research question, we investigate two prediction targets, Within Project Prediction (WPP) where a project's prediction model is used to predict on the project's test dataset and Transfer Project Prediction (TPP) where a project's prediction model is used to predict on the test dataset of other projects. In total and using the treatments listed in Table 4, we constructed 3600 prediction models for each project's dataset, 400 models for WPP and 3200 models for TPP as listed in Table 5. All data, source

Table 4 Treatments and parameters for training dataset

Treatment	Values	Feature selection method
Security Category	Threat (T), control (C), asset (A), implicit (I)	fsec-ext
Include Security Category in Training dataset	true, false	fsec-ext, fsec-tfidf
Class Ratio (SBR: NSBR)	0.0, 0.5, 1.0, 1.5, 2.0	fsec-ext, fsec-tfidf

Table 5 Experiment details

Prediction approach	Prediction models	Total
WPP	1 dataset	
	5 machine learning algorithms (RF, SVM, NB, LR, KNN)	
	2 feature selection approaches (fsec-ext, fsec-tfidf)	
	7 security categories (T, C, TC, TA, CA, TCA, TCAI) for fsec-ext	
	2 Boolean include security category in train (True, False)	
	5 class ratios (0.0, 0.5, 1.0, 1.5, 2.0)	400
TPP	1 dataset	
	8 targets	
	5 machine learning algorithms (RF, SVM, NB, LR, KNN)	
	2 feature selection approaches (fsec-ext, fsec-tfidf)	
	7 security categories (T, C, TC, TA, CA, TCA, TCAI) for fsec-ext	
	2 Boolean include security category in train (True, False)	
	5 class ratios (0.0, 0.5, 1.0, 1.5, 2.0)	3200
Total		3600

code and analysis scripts for our experiments are available online.[9] We have used five common text classification algorithms that have been used extensively in the vulnerability and defect prediction community (Joachims, 1998); (Peters et al., 2017) namely support vector machine (SVM), random forest (RF), Naïve Bayes (NB), logistic regression (LR) and K-nearest neighbour (KNN). We have used the Weka (Witten & Frank, 2005) machine learning library version 3.8.4 with the default parameter settings for the five algorithms.

3.2.1.11 Complexity of our approach A DTM typically yields a large number of terms that are used as features for training a prediction model. Both the size of the features and the training dataset influence the complexity of learning algorithms. For instance, the best-case time complexity of random forest (Louppe, 2014) for building forests of M randomised tree on N observation points and using K randomly drawn features (variables) from a total of p variables (where $\mathrm{K} \leq p$) is given as $\mathrm{O}(MKN\log^2 N)$ and the worst case as $\mathrm{O}(MKN^2\log N)$.

Assume the number of features in *fsec-ext* is $\boldsymbol{P}$, and $\boldsymbol{N}$ is the total bug records (SBR + NSBR) in the training dataset (i.e. observation points), *fsec-ext+* includes the features $\boldsymbol{P}$ in the training dataset to increase the observation points to $\boldsymbol{N+P}$. This implies a complexity that is dependent on $\boldsymbol{N}$ or $\boldsymbol{P}$ depending on which one is higher. $\boldsymbol{P>N}$ exists for cases where the number of features is more than the training bug reports. Examples are the Ambari, Wicket, Derby, Camel, Comm, ODCV and Apache dataset. While $\boldsymbol{N>P}$ exists for cases where the number of training bug reports is higher than the number of features. Examples are Chromium and Mozilla dataset. Using random forest, which has the worst running time in our experiments, the worst-case time complexity when $\boldsymbol{N>P}$ will be $\mathrm{O}(MKN^2\log(\boldsymbol{N+P}))$ and when $\boldsymbol{P>N}$, it will be $\mathrm{O}(MKP^2\log(\boldsymbol{N+P}))$.

3.2.1.12 Comparing with Peter's et al. (2017) We compare our results to Peters et al. (2017) where applicable. Peters et al. (2017) use filtering techniques named FARSEC to remove

[9] https://github.com/tosdanoye/fsecext

NSBRs that contain security crosswords from the training dataset. A security crossword is a feature (or dictionary term) that exists in both the SBR and NSBR. FARSEC filtering works by first identifying 100 terms with the highest *tf-idf* as defined in Eq. 3 to form a dictionary. Next, for each term in this dictionary, FARSEC computes a score which is the probability of the term appearing in SBRs and NSBRs. FARSEC proceeds to score each bug report by retrieving the scores for all the dictionary terms found in the bug report and then producing an overall score or ranking for this bug report. Finally, FARSEC filters out NSBR with overall ranking ≥ 0.75 from the training dataset. Rather than focusing on filtering NSBR, we take a different approach by focusing on feature selection methods, term weighting and sampling of imbalance dataset. To have a fair comparison with our work, we have used the top 100 terms identified using the same *tf-idf* metric defined in Eq. 3 to generate our feature set for the model *fsec-tfidf*.

Table 6 lists the security keywords generated by *fsec-tfidf* for each project compared with the list in Peters et al. (2017). We observe some slight differences in both lists. We speculate that the differences may be due to differences in the programming language used and especially how tokens are derived from the documents. For example, the token 'interfacerequest' keyword from Peters et al. (2017) in Wicket is not a single word but a substring from e.g. BookmarkableListenerInterfaceRequestTarget. Such differences can account for the disparity between our list and Peters et al. (2017).

3.2.1.13 Approach to research questions We now define our research questions and hypotheses based on our overall question. Broadly, we investigate whether using features based on external data sources can produce prediction models that improve transfer project predictions on other projects. We then investigate whether including security features in the training dataset can improve the performance of a model. Lastly, we investigate if these modelling approaches have significant effect on within-project predictions.

We formulate our research questions and hypotheses as follows:

RQ1: Do externally sourced features generalise better than project-specific features for TPP?

We test the hypothesis: **H1:** *fsec-ext*/ *fsec-ext*+ can generalise on other projects significantly better than *fsec-tfidf*/*fsec-tfidf*+ for TPP.

RQ2: Does including security features as SBRs in training dataset improve model's performance over models that exclude them for both WPP and TPP?

We test the hypothesis: **H2:** *fsec-ext*+ significantly outperform *fsec-ext* and *fsec-tfidf*+ significantly outperform *fsec-tfidf*.

RQ3: Do models that utilise external sources for features outperform model that uses project-specific features in within-project prediction (WPP)?

We test the hypothesis: **H3:** *fsec-ext*/*fsec-ext*+ significantly outperforms *fsec-tfidf*/*fsec-tfidf*+ for WPP.

We investigate the performance measures of each model type we have defined on its own dataset (WPP) as well as other projects' datasets (TPP) and compare the results. Both goals are complementary and important to the goal of our study, as a model that performs well on its own dataset but poorly on another dataset would not be useful for generalisation purpose. Conversely, a model that performs well on other datasets but poorly on its own dataset may be useless. We use summary statistics showing the minimum, maximum and average of the performance metrics to explain our results. A well-generalised and robust model will show high minimum and mean g-measure, recall, f-score and low standard deviations. To capture this generalisation effect for a target, we collect the prediction

Table 6 Security-related keywords derived from using the normalised *tf-idf* (Manning et al., 2008)

Project	Peters et al. (2017) (top 100)	Ours (top 100)	%Match
Chromium	file security chrome page http download user starred person notified changes may see url site bug open google browser like windows window https web code one memory firefox function tests problem seems tab also version use would using view used make users chromium crash click password think vulnerability sure browsers link attached attacker data get fix const content something safari new error javascript lcamtuf malicious please could risk release try found allow expected time example corruption test back access crashes urls int without know versions way uses cause fail want system still files arbitrary html details ssl need loaded might	file chrome security page download user http url see starred notified person changes may issue site open bug google script like browser windows web code problem one users tab memory firefox use html function seems version also would view using used tests safari data chromium crash content example window click get vulnerability test please make browsers password think link attached fix attacker sure javascript new type expected error malicious back something could found risk fail try files allow amp corruption time const release access int lcamtuf https urls versions skylinedchromium want know way system cause uses without line run crashes	93
Wicket	statelesshomepage attached calls dataprovider regards component limited files count jan couple database integer entries reason page get first java error unknown source signinform jetty requestlistenerinterface credentialsexpected params http exception interfacerequest homepage manually become inform stateful listener happends statelesschecker insucceeded creates causes sucessfull opensactual final abstractlistener quickstart mvn login analysis open unpack temporary enter sign valuemap happy statelessproblem signinpanel requestcycle occured fix delete deleting fails upload org webapplication hole space created handle example security server find important method multipart easy hope makes incomplete cancelled think workaround uploading parserequest want really anyone disk large eating bug throw posting developers call threads servletwrapper	reason jan statelesshomepage get signinform component calls first couple integerregards quickstart count entries attached files page database limited dataprovider error source temporary exception signinpanel happy enter happends analysis params become credentialsexpected login causes insucceeded abstractlistenerinterfacerequesttarget open requestlistenerinterface listener reproduce opensactual manually final http creates valuemap statelesschecker sucessfull unpack jetty stateful mvn statelessproblem homepage upload fix deleting delete fails occured org requestcycle httpservlet javautil developers hope large method want handle anyone cancelled workaround example think makes eating easyfor fileuploadexceptioni bug throw wicketutiluploadfileuploadbase parserequest space incomplete find uploading created important disk multipart security servers soonthanks filesi posting really hole user webapplication concurrently	87

Table 6 (continued)

Project	Peters et al. (2017) (top 100)	Ours (top 100)	%Match
Ambari	security wizard service secure fails permissions allow start validation user cluster cannot set make page configs datanode add use default property name request instead enable web ssl password used services error disable fix permission options executed setup http nagios registration hosts also change url configuration try enabling check disabled host install time return provide call script issue file failures principal touched incorrect artifacts assignments slaves side username directory path customised effects mode unwanted false causes broken primary testmode mapred names working httpd state missing navigation ganglia prepare locked master ambari smoke wrong hbase node test zookeeper back need either true	security wizard service secure permissions fails allow validation user cluster start cannot set property make use add configs default name page datanode request web enable username ssl instead used disable services fix permission executed install status setup config url change also nagios options registration configuration true try hosts disabled time check call provide enabling return script host file prepare zookeeper touched incorrect test directory false navigation httpd customised side names namegt mode locked artifacts amp working master ganglia state node mapred broken controller jdk testmode missing hbase etcconfhadoop assignments slaves primary ambari causes tracker wrong effects error principal smoke unwanted	92
Camel	message http endpoint header org would uri component files also issue stop expose see static jetty endpoints specified port processing login route file error one messages headers support server lines using throws provide currently instance ftp sent means use issues ignore license dsl interfaces opened host fails redelivery auth interceptsendtoendpoint used connection check memory heap nabblehttp bundles consumer pass path new case based classes however default errorhandler pushing expectedbody resultendpoint exchange following exception else apache null inoptionalout handler token custom ignores servers something like protected sftpendpoint remove jms concatenated delegate underlying ssh data copy ftpcomponent method second csv results mockendpoint false	org message header endpoint http would uri component also processing files issue stop see endpoints static port jetty headers expose login specified route file error messages one support server instance provide currently lines means using throws case ftp use sent issues camelmessage check remove ignore fails interceptsendtoendpoint underlying delegate auth host license dsl heap bundles routegt memory import interfaces connection used jms opened nabblehttp redelivery path pass new consumer based classes however pushing content exchange following exception else apache method protected sftpendpoint ignores custom httpuri handler ssh something concatenated token data like servers removeheader uriftplocalhostinboxusernameusramp least componentjmsjmsmessage resultgetouttruecopyfromout information comjavaheapspaceissuewithreadinglargecsvfiletthtmla	88

Table 6 (continued)

Project	Peters et al. (2017) (top 100)	Ours (top 100)	%Match
Derby	org security server test derby permission java using access tests error support denied junit file exception database code user read manager locale run fails need version fail running securitymanager network following files call required statement source table connect class thread block securityexception would used like failed problem privileged client see jdbc set method filepermission trying granted authentication needs directory connection new think encryption sun jar policy start stack unknown rows revoke found information alpha thrown without name one create end update http could make trigger though native contains looks two key mode results sql use int classpath message incorrectly check	org server security java derby test permission error using exception testing access tests support denied database code junit found file read user run jdbc fails locale need manager fail version following network connect statement running files securitymanager table would block source required class securityexception problem used see set trying method thread think call encryption filepermission needs new connection privileged main revoke granted create start failed like client policy jar authentication one information current rows thrown directory alpha name end make without update could stack though int trigger mode looks key jira execute values sql insert use contains message two incorrectly	93

performance results from each source per treatment—algorithm, selection method, ratio and security category. For eight (8) sources, this will yield a total of 8 performance results per target per treatment. We then compute the minimum, maximum, mean and standard deviations for g-measure, f-score and recall. These statistics allow us to truly appreciate how well a given model can generalise across all projects.

We follow the guideline of Demsar (2006) (He & Garcia, 2009) and use the Wilcoxon signed rank test (a non-parametric test) at 95% confidence level to compare (1) whether the performance of each model (*fsec-ext or fsec-tfidf*) significantly improve when security features are included (*fsec-ext*+ *or fsec-tfidf*+) over when security features are excluded and (2) whether the performance of model *fsec-ext* is better and statistically significant or not to model *fsec-tfidf*. In addition, we perform effect size check on our results. As noted in Kampenes et al. (2007), effect size quantifies the size of the difference between two groups and allows us to judge whether the conclusions drawn from our hypotheses testing are meaningful or not. It might be possible that the effect size is negligible even when the statistical test is significant and vice versa. We apply the Hedges, g standardised effect size measure calculated as:

$$\text{Hedges}, g = \frac{\bar{X}_1 - \bar{X}_2}{S_p}$$

where $\bar{X}_1$ and $\bar{X}_2$ represent the sample means for classification measures (recall, f-score and g-measure) for the model pairs per project (e.g. *fsec-ext* vs. *fsec-tfidf*) and S_p represents the pooled standard deviation computed from the standard deviations of s_1 and s_2 of the two groups. Effect size results can be interpreted in different ways as explained in Kampenes et al. (2007). For *fsec-ext*+ *vs fsec-ext* and *fsec-tfidf*+ *vs fsec-tfidf*, we use paired data. That is, each data pair contains metric values of model before and after treatment. The treatment in this case is including security features as SBR in the training dataset, all other variables for both groups are kept constant—i.e. algorithm, ratio and security category. For *fsec-ext*+ *vs fsec-tfidf*+ and *fsec-ext vs fsec-tfidf*, we use unpaired data by collecting top 20 results for each model type and then test which group's mean is significant to the other. We compare our results with those reported in Software Engineering empirical studies categorised under Table 9. The size category for 284 estimated values for Hedges, g is given as Small: 0.00–0.376, Medium: 0.378–1.000 and Large: 1.002–3.40. We have used standard statistical packages in *R* (2008) for these purposes.

4 Results

4.1 Harvested security keywords

We have mined a total of 54 keywords for the asset category, 334 keywords for the control category, 329 keywords for the attack category and 203 keywords for the implicit category. We present a sample of the security words we have mined in Table 7. We caution that this list is not, and cannot be, exhaustive, as additional keywords can be extracted from other sources.

We present the results of the models (*fsec-ext* and *fsec-tfidf*). We have reported the best results and the summary statistics for TPP. The best result is determined by sorting

Table 7 Sample of security keywords extracted from different sources and grouped into 4 categories

	Assets	Control	Attack/threat	Implicit
Total	54	334	329	203
	account, bank identification number, BIN, biometric, card, cookie, credential, credit card, date of birth, dob, e-cash, electronic commerce, electronic signature, electronic wallet, fingerprint, Geld-Karte, home address, identity, ip address, keys, log in, login, passphrase, password, payment card, personal health information, personal identifiable information, pii, token	3DES, access control, access level, access profile, access right, access role, ACL, admin, administrator, Advanced Encryption Standard, AES, anomaly detection, anonymous credential, anonymous login, anti-jam, attack tree, audit, audit log, audited, AUTH, back up, basicauth, bcrypt, biometric, black list, Blocking, Blowfish, Bounds Checking, Bytecode obfuscation, Canonicalization, CAPTCHAS	active wiretapping, adversary, attack, attack potential, attacker, adversary, anonymous login, arbitrary code, attempt, blocked, backdoor, backtrack attack, Binary planting, birthday attack, blind attack, Blind SQL Injection, Blind XPath Injection, brute force, buffer overflow, bypass, Cache Poisoning, Code Injection, command injection, compromise	configuration, degradation, delete, deny, disk space, enable, error, exception, execute, expiration, expire, expiring, expiry, flaw, guest login, inactive, incorrect, invalid, invalidate, log, loop, memory, misuse, null, null pointer, null-pointer, nullpointer, nullpointerexception, non-final public field, obstruction, parameter

Table 8 Summary statistics of best generalised models for TPP

					g-measure				f-score				Pd				aucroc			
Target	Method	Cat	Ratio	Learner	min	max	avg	std	min	max	avg	std	min	max	avg	std	min	max	avg	std
Chromium	fsec-ext+	TC	0.0	RF	72.1	78.8	74.1	2.3	2.6	5.3	3.4	0.8	65.2	80.9	72.8	5.8	71.1	83.2	78.9	3.7
	fsec-ext	C	2.0	LR	55.0	71.7	61.5	6.0	3.6	24.1	12.2	7.1	38.3	64.3	46.5	8.9	63.5	73.8	69.6	3.2
	fsec-tfidf+	–	1.5	RF	31.5	70.4	55.7	13.5	0.7	2.7	1.8	0.7	38.3	77.4	59.1	12.2	35.5	75.1	61.6	12.5
	fsec-tfidf	–	0.5	NB	11.4	64.4	35.9	22.8	0.8	9.0	2.9	2.9	6.1	53.0	27.7	20.6	39.1	74.4	58.3	13.1
Wicket	fsec-ext+	TC	0.5	LR	69.1	82.3	75.7	5.4	4.9	9.7	7.0	1.9	66.7	83.3	79.2	7.7	67.8	87.2	78.1	6.9
	fsec-ext	TA	1.0	LR	28.1	76.3	56.3	19.3	3.3	12.7	6.9	2.8	16.7	66.7	45.8	21.4	51.4	79.1	63.1	9.6
	fsec-tfidf+	–	–	LR	42.7	74.9	59.6	10.9	2.3	9.8	4.3	2.4	50.0	83.3	70.8	14.8	45.3	81.3	66.3	11.7
	fsec-tfidf	–	–	NB	27.5	73.5	48.9	15.9	1.8	8.0	4.1	1.7	16.7	66.7	39.6	19.8	53.0	71.2	61.1	7.6
Ambari	fsec-ext+	C	0.5	KNN	57.6	78.0	72.3	6.5	7.9	12.2	9.3	1.5	42.9	71.4	66.1	10.6	55.0	80.8	73.2	8.2
	fsec-ext	CA	1.0	NB	58.3	81.4	74.4	8.1	9.1	26.3	16.6	6.6	42.9	71.4	64.3	10.8	67.2	86.0	78.5	5.6
	fsec-tfidf+	–	1.5	LR	34.0	69.9	54.6	13.7	3.1	5.6	4.2	1.0	42.9	100.0	82.1	19.8	51.0	84.3	72.2	10.8
	fsec-tfidf	–	0.5	NB	24.0	77.4	60.2	17.6	1.5	11.2	5.6	3.0	14.3	85.7	57.1	22.9	38.5	79.0	61.5	14.0
Camel	fsec-ext+	CA	0.5	LR	57.2	71.3	65.9	4.3	11.8	16.8	14.6	2.1	44.4	66.7	59.7	7.7	68.7	76.1	71.4	2.7
	fsec-ext	TCAI	0.5	NB	33.6	68.3	54.2	13.7	7.7	24.1	12.9	5.0	22.2	88.9	53.5	20.1	52.5	76.1	64.4	8.0
	fsec-tfidf+		1.5	LR	40.7	70.4	54.1	8.8	7.2	19.8	9.9	4.2	44.4	83.3	68.8	12.9	47.8	80.6	61.0	10.2
	fsec-tfidf		0.5	NB	46.4	68.0	54.9	6.8	6.1	13.5	8.9	2.5	38.9	66.7	52.1	10.3	43.3	67.9	54.5	7.3
Derby	fsec-ext+	C	2.0	LR	65.5	75.5	70.4	2.9	26.2	50.0	32.3	7.5	57.1	71.4	66.1	5.2	71.1	79.0	74.9	2.5
	fsec-ext	TCA	2.0	LR	50.4	73.0	60.6	6.6	22.8	38.7	28.8	5.2	35.7	78.6	50.6	13.6	49.6	80.5	66.7	9.4
	fsec-tfidf+		2.0	LR	41.1	67.1	51.1	9.1	11.2	31.6	17.9	6.8	31.0	69.0	53.3	13.4	43.8	68.8	55.5	8.8
	fsec-tfidf		0.5	NB	48.1	61.9	55.3	5.3	13.8	22.0	17.7	3.1	42.9	81.0	60.4	11.4	51.1	66.0	58.4	5.1
ODCV	fsec-ext+	TC	0.5	KNN	64.2	67.3	65.7	0.9	62.1	65.8	63.8	1.1	51.2	60.3	55.2	2.7	67.8	70.4	69.2	1.0
	fsec-ext	TCA	2.0	LR	4.6	58.8	34.5	21.1	4.6	57.7	33.4	20.4	2.4	42.7	23.4	16.3	54.8	69.3	59.9	5.5
	fsec-tfidf+		0.0	KNN	31.3	55.8	46.9	7.8	29.5	58.1	44.9	9.3	19.0	59.7	38.2	13.8	51.5	59.3	55.7	2.7
	fsec-tfidf		1.0	LR	0.0	50.0	15.8	21.3	0.0	46.9	14.9	19.8	0.0	35.9	10.7	15.0	46.4	59.3	53.6	4.4
Apache	fsec-ext+	TC	0.5	LR	78.5	93.1	88.5	5.0	77.5	92.8	88.0	5.3	67.4	95.0	84.6	9.5	83.2	94.3	89.4	4.4

Table 8 (continued)

Target	Method	Cat	Ratio	Learner	g-measure				f-score				Pd				aucroc			
					min	max	avg	std	min	max	avg	std	min	max	avg	std	min	max	avg	std
	fsec-ext	CA	1.0	LR	60.2	91.7	72.7	9.7	58.9	91.3	72.0	9.9	44.1	90.1	59.6	14.2	56.0	93.9	75.0	11.0
	fsec-tfidf+		2.0	LR	38.0	70.8	55.9	9.8	47.7	75.7	59.9	8.8	45.2	90.3	64.0	17.4	49.3	87.1	65.7	12.1
	fsec-tfidf		0.5	NB	51.8	72.6	64.1	7.8	47.9	70.6	62.5	8.5	38.3	73.8	59.3	12.2	57.6	79.7	70.1	7.0
Mozilla	fsec-ext+	TCAI	0.5	KNN	75.2	78.6	77.3	1.2	74.3	78.0	76.3	1.3	62.5	71.2	66.4	2.7	76.9	83.3	79.2	2.0
	fsec-ext	CA	1.0	LR	9.3	69.5	47.7	17.7	9.2	69.4	47.6	17.7	4.9	53.4	32.8	14.0	60.1	73.0	65.7	4.3
	fsec-tfidf+		0.0	KNN	39.7	63.6	54.4	7.8	35.2	61.9	52.4	9.4	27.0	65.2	47.5	13.9	54.2	67.6	60.3	4.7
	fsec-tfidf		0.5	NB	2.3	53.6	25.4	20.3	2.3	53.5	24.7	20.0	1.2	36.7	16.2	13.7	46.3	73.1	62.5	8.0
Comm	fsec-ext+	TC	2.0	SVM	73.7	88.2	81.8	5.8	12.8	45.5	26.1	11.5	71.4	100.0	85.7	10.1	78.0	88.3	83.2	4.3
	fsec-ext	TCA	2.0	LR	34.5	77.1	59.8	15.3	7.7	21.1	13.8	4.6	21.4	85.7	52.7	22.9	44.6	83.1	65.3	13.6
	fsec-tfidf+		2.0	RF	30.9	75.4	57.7	14.4	5.9	17.1	9.4	3.7	35.7	100.0	67.0	19.4	43.3	81.8	65.3	14.5
	fsec-tfidf		0.5	NB	41.6	80.6	58.7	12.1	6.3	16.4	9.6	3.2	28.6	92.9	54.5	21.2	60.5	86.3	67.9	8.3

NB Naïve Bayes, *LR* logistic regression, *RF* random forest, *KNN* K-nearest neighbour, *SVM* support vector machine

Table 9 TPP results with the best g-measure

Target	Source	Method	Cat	Ratio	Learner	TN	TP	FN	FP	Pd	Pf	Prec	F-score	G-measure	AUCROC
Chromium	ODCV	fsec-ext+	TC	0.0	RF	17,852	84	31	2994	73,0	14,4	2,7	5,3	78,8	83,2
	ODCV	fsec-ext	TCA	2.0	RF	17,942	80	35	2904	69,6	13,9	2,7	5,2	76,9	80,9
	ODCV	fsec-tfidf+	–	0.0	RF	15,119	79	36	5727	68,7	27,5	1,4	2,7	70,6	75,1
	ODCV	fsec-tfidf	–	2.0	RF	18,386	70	45	2460	60,9	11,8	2,8	5,3	72,0	75,3
Wicket	Camel	fsec-ext+	TCAI	2.0	LR	412	6	0	82	100,0	16,6	6,8	12,8	90,9	94,0
	Apache	fsec-ext	TC	2.0	LR	422	5	1	72	83,3	14,6	6,5	12,0	84,4	84,5
	Comm	fsec-tfidf+	–	0.0	LR	411	5	1	83	83,3	16,8	5,7	10,6	83,3	87,9
	Apache	fsec-tfidf	–	0.5	RF	335	5	1	159	83,3	32,2	3,0	5,9	74,8	73,0
Ambari	Mozilla	fsec-ext+	CA	1.5	NB	471	5	2	22	71,4	4,5	18,5	29,4	81,7	80,3
	Comm	fsec-ext	TCA	0.0	NB	476	5	2	17	71,4	3,4	22,7	34,5	82,1	86,1
	Derby	fsec-tfidf+	–	0.0	SVM	415	6	1	78	85,7	15,8	7,1	13,2	84,9	84,9
	Chromium	fsec-tfidf	–	1.5	RF	467	5	2	26	71,4	5,3	16,1	26,3	81,4	71,6
Camel	Derby	fsec-ext+	TCAI	1.0	LR	359	13	5	123	72,2	25,5	9,6	16,9	73,3	71,2
	Apache	fsec-ext	CA	0.0	SVM	443	11	7	39	61,1	8,1	22,0	32,4	73,4	76,5
	Apache	fsec-tfidf+	–	0.5	LR	306	15	3	176	83,3	36,5	7,9	14,4	72,1	81,5
	Apache	fsec-tfidf	–	1.5	LR	363	13	5	119	72,2	24,7	9,8	17,3	73,7	80,1
Derby	Apache	fsec-ext+	TC	1.5	SVM	374	32	10	84	76,2	18,3	27,6	40,5	78,8	78,9
	Apache	fsec-ext	TCA	1.5	SVM	389	30	12	69	71,4	15,1	30,3	42,6	77,6	78,2
	Apache	fsec-tfidf+	–	1.5	RF	329	30	12	129	71,4	28,2	18,9	29,9	71,6	76,2
	Apache	fsec-tfidf	–	1.0	RF	342	27	15	116	64,3	25,3	18,9	29,2	69,1	71,4
ODCV	Mozilla	fsec-ext+	T	0.0	RF	103	278	165	32	62,8	23,7	89,7	73,8	68,9	67,5
	Mozilla	fsec-ext	TA	1.0	RF	103	259	184	32	58,5	23,7	89,0	70,6	66,2	65,3
	Chromium	fsec-tfidf+	–	0.0	RF	82	238	205	53	53,7	39,3	81,8	64,9	57,0	60,2
	Mozilla	fsec-tfidf	–	0.5	KNN	193	143	152	91	48,5	32,0	61,1	54,1	56,6	59,9
Apache	Mozilla	fsec-ext+	TC	2.0	SVM	611	585	34	53	94,5	8,0	91,7	93,1	93,2	93,3
	Mozilla	fsec-ext	TCA	2.0	LR	618	573	46	46	92,6	6,9	92,6	92,6	92,8	93,9
	Mozilla	fsec-tfidf+	–	2.0	SVM	516	479	140	148	77,4	22,3	76,4	76,9	77,5	77,5

Table 9 (continued)

Target	Source	Method	Cat	Ratio	Learner	TN	TP	FN	FP	Pd	Pf	Prec	F-score	G-measure	AUCROC
	Mozilla	fsec-tfidf	–	1.5	SVM	555	472	147	109	76,3	16,4	81,2	78,7	79,7	79,9
Mozilla	Apache	fsec-ext+	TCAI	0.0	LR	1610	1153	459	54	71,5	3,2	95,5	81,8	82,2	85,4
	Apache	fsec-ext	TCA	2.0	LR	1662	918	694	2	56,9	0,1	99,8	72,5	72,5	76,6
	Apache	fsec-tfidf+	–	0.5	RF	1135	999	613	529	62,0	31,8	65,4	63,6	64,9	71,4
	Chromium	fsec-tfidf	–	2.0	LR	1321	783	829	343	48,6	20,6	69,5	57,2	60,3	63,3
Comm	Chromium	fsec-ext+	TC	2.0	LR	426	12	2	33	85,7	7,2	26,7	40,7	89,1	85,9
	Chromium	fsec-ext	TCAI	2.0	NB	430	11	3	29	78,6	6,3	27,5	40,7	85,5	90,1
	Apache	fsec-tfidf+	–	1.0	SVM	382	11	3	77	78,6	16,8	12,5	21,6	80,8	80,9
	Apache	fsec-tfidf	–	0.5	NB	327	13	1	132	92,9	28,8	9,0	16,4	80,6	86,3

Table 10 H1-Wilcoxon and effect size tests comparing models *fsec-ext and fsec-tfidf* (TPP)

		fsec-ext+ vs. fsec-tfidf+		*fsec-ext vs. fsec-tfidf*	
	Metrics	*p* value	Hedges, *g*	*p* value	Hedges, *g*
Chromium	recall (Pd)	< 0.0001	1.88	< 0.0001	1.96
	f-score	< 0.0001	1.06	< 0.0001	1.61
	g-measure	< 0.0001	8.04	< 0.0001	3.59
Wicket	recall (Pd)	< 0.0001	1.53	0.07	0.32
	f-score	< 0.0001	1.97	< 0.0001	2.34
	g-measure	< 0.0001	8.23	< 0.0001	2.01
Ambari	recall (Pd)	0.91	-0.60	0.001	1.03
	f-score	< 0.0001	3.48	< 0.0001	3.05
	g-measure	< 0.0001	2.13	< 0.0001	2.65
Camel	recall (Pd)	0.17	0.28	0.01	0.34
	f-score	< 0.0001	3.92	< 0.0001	2.52
	g-measure	< 0.0001	6.70	< 0.0001	2.76
Derby	recall (Pd)	< 0.0001	1.60	0.16	0.1
	f-score	< 0.0001	5.12	< 0.0001	2.73
	g-measure	< 0.0001	8.15	< 0.0001	3.10
ODCV	recall (Pd)	0.0002	1.40	< 0.0001	1.12
	f-score	0.0004	1.00	< 0.0001	1.33
	g-measure	< 0.0001	4.47	< 0.0001	2.10
Apache	recall (Pd)	< 0.0001	2.61	< 0.0001	2.86
	f-score	< 0.0001	6.63	< 0.0001	7.03
	g-measure	< 0.0001	10.1	< 0.0001	8.60
Mozilla	recall (Pd)	< 0.0001	3.73	< 0.0001	4.70
	f-score	< 0.0001	17.9	< 0.0001	6.81
	g-measure	< 0.0001	19.9	< 0.0001	6.21
Comm	recall (Pd)	< 0.0001	1.65	0.03	0.58
	f-score	< 0.0001	2.64	0.0003	1.24
	g-measure	< 0.0001	4.73	< 0.0001	2.18

first by the g-measure, followed by the f-score and then by recall. Table 8 presents the summary statistics (i.e. mean, minimum, maximum and standard deviations) of the most generalised model for Transfer Project Prediction (TPP). Table 9 lists the results with the best g-measure for TPP. Table 10 lists the statistical tests for comparing the two models for TPP. Table 11 presents the statistical tests for comparing the effect of including and excluding features during training on both WPP and TPP. Table 12 lists the results with the best g-measure for WPP. Figure 3 shows the mean of top 20 results of each model for WPP. Table 13 lists the statistical tests that compare the two models for WPP. Finally, Tables 14 and 15 list the results with the best g-measure from all the models for WPP and TPP respectively in comparison with the best results from Peters et al. (2017).

Table 11 H2—Wilcoxon and effect size tests comparing models *fsec-ext and fsec-tfidf* when feature is included or excluded

	Metrics	*fsec-ext+ vs fsec-ext*				*fsec-tfidf+ vs. fsec-tfidf*			
		WPP		TPP		WPP		TPP	
		p value	Hedges, *g*	*p* value	Hedges, *g*	*p* value	Hedges, *g*	*p* value	Hedges, *g*
Chromium	recall (Pd)	< 0.0001	0.35	< 0.0001	0.67	< 0.0001	1.51	< 0.0001	1.17
	f-score	0.95	– 0.17	0.67	– 0.21	0.77	0.06	< 0.0001	0.16
	g-measure	0.23	– 0.02	< 0.0001	0.46	0.0001	0.90	< 0.0001	0.73
Wicket	recall (Pd)	< 0.0001	0.48	< 0.0001	0.86	< 0.0001	2.18	< 0.0001	1.01
	f-score	0.99	– 0.25	< 0.0001	0.04	< 0.0001	1.50	0.20	– 0.02
	g-measure	< 0.0001	0.70	< 0.0001	0.64	< 0.0001	2.61	< 0.0001	0.51
Ambari	Recall (Pd)	0.99	– 0.58	< 0.0001	0.14	0.04	0.40	< 0.0001	0.96
	f-score	1.0	– 0.77	1.0	– 0.37	0.41	– 0.15	0.99	– 0.25
	g-measure	0.99	– 0.57	0.88	– 0.03	0.07	0.40	0.11	0.15
Camel	recall (Pd)	< 0.0001	0.33	< 0.0001	0.60	0.33	0.24	< 0.0001	1.02
	f-score	0.002	0.18	0.58	– 0.03	0.74	– 0.11	< 0.0001	0.36
	g-measure	< 0.0001	0.78	< 0.0001	0.43	0.51	0.17	< 0.0001	0.50
Derby	recall (Pd)	0.62	– 0.01	< 0.0001	0.46	0.0003	0.89	< 0.0001	0.78
	f-score	0.99	– 0.72	0.81	0.01	0.99	– 0.73	0.04	0.24
	g-measure	0.99	– 0.41	< 0.0001	0.30	0.08	0.34	0.0005	0.40
ODCV	recall (Pd)	0.41	0.13	< 0.0001	0.84	< 0.0001	1.11	< 0.0001	1.21
	f-score	0.66	0.02	< 0.0001	1.05	< 0.0001	0.85	< 0.0001	1.46
	g-measure	0.99	– 0.18	< 0.0001	0.78	0.003	0.22	< 0.0001	0.66
Apache	recall (Pd)	0.03	0.08	< 0.0001	0.56	0.003	0.56	< 0.0001	1.14
	f-score	0.51	– 0.02	< 0.0001	0.46	0.91	– 0.33	< 0.0001	0.71
	g-measure	0.72	– 0.12	< 0.0001	0.27	0.99	– 0.64	0.06	0.18
Mozilla	recall (Pd)	0.001	0.17	< 0.0001	0.80	0.005	0.57	< 0.0001	1.46
	f-score	0.08	0.05	< 0.0001	0.88	0.01	0.45	< 0.0001	1.60
	g-measure	0.40	– 0.07	< 0.0001	0.50	0.01	0.47	< 0.0001	0.60
Comm	recall (Pd)	< 0.0001	1.23	< 0.0001	0.84	0.0003	0.93	< 0.0001	1.12
	f-score	< 0.0001	0.81	< 0.0001	0.18	0.006	0.52	0.25	0.10

Table 11 (continued)

Metrics	*fsec-ext+ vs fsec-ext*				*fsec-tfidf+ vs. fsec-tfidf*			
	WPP		TPP		WPP		TPP	
	p value	Hedges, *g*	*p* value	Hedges, *g*	*p* value	Hedges, *g*	*p* value	Hedges, *g*
g-measure	< 0.0001	1.05	< 0.0001	0.53	< 0.0001	1.10	< 0.0001	0.36

4.2 RQ1: Do externally sourced features generalise better than project-specific features for TPP?

To answer RQ1, we use the summary statistics listed in Table 8 complimented by the results in Table 9 that lists the results for TPP with the best g-measure. The summary statistics show the treatments—ratio, security category, algorithm—and the feature selection method that yield the best results from using different sources (8) to predict a target. For example, for Chromium, random forest (RF) combined with 0.0 ratio and using security categories threats and control produce the best and most stable results when applied on each source project to predict Chromium. Predictions using TPP shows model *fsec-ext+* with an average recall from 55 to 86%, minimum recall from 43 to 71% and maximum recall from 60 to 100% across all projects. Model *fsec-ext* has an average recall from 23 to 64%, minimum recall from 2.4 to 44% and maximum recall from 42 to 90%. Model *fsec-tfidf+* has an average recall from 38 to 82%, minimum recall from 19 to 50% and maximum recall from 60 to 100%. Model *fsec-tfidf* has an average recall from 11 to 60%, minimum recall from 0 to 43% and maximum recall from 36 to 60%.

In comparison with model *fsec-tfidf+* , model *fsec-ext+* increased recall on average by 4–17% and by 26–44% when compared with *fsec-tfidf*. The average f-score for model *fsec-ext+* is between 3.4 and 88% which is higher by 1.6–28% in all projects when compared with model *fsec-tfidf+*. One reason for the low precision and subsequent low f-score is because of scarcity of SBRs in the Commercial, Ambari, Derby, Camel and Wicket datasets. Although the f-score is lower, the average g-measure for *fsec-ext+* is at least 66% across all projects and higher than model *fsec-tfidf+* by 19–28% across the projects. This shows that the true negative rate is relatively high and that the cost of misclassification can be manageable. For instance, the TPP result of Camel → Wicket in Table 9 gives a TP = 6, FN = 0, TN = 412, and FP = 82. This confusion matrix produces a recall of 100%, f-score of 13% and a g-measure of 91%. Based on this model's prediction, a total of 88 (TP = 6 and FP = 82) security bug reports will have to be examined by a security/quality assurance team that relies on such model which is equivalent to reviewing 17.6% (88 out of 500) of the total bug reports for security-related bugs.

A striking observation is that *fsec-ext+* produces the most stable model that generalises better across all projects with the exception of Ambari where *fsec-ext* is slightly better in g-measure and f-score. The standard deviations of the measures for *fsec-ext+* are also lower in most cases compared with the other models. This suggests that including security features can produce a more robust, stable and generalisable model. As regards AUC of ROC, model *fsec-ext+* has an average between 69 and 89%, *fsec-ext* has an average between 60 and 78%, *fsec-tfidf+* has an average from 55 to 72%, and *fsec-tfidf* from 54 to 70%. For all the projects, except Ambari, the AUC of ROC of *fsec-ext+* is the highest and

Table 12 Best g-measure per project for WPP

Target	Method	Cat	Ratio	Learner	TN	TP	FN	FP	Pd	Pf	Prec	F-score	G-measure	AUCROC
Chromium	fsec-ext+	TCA	0.5	KNN	18,325	80	35	2521	69.6	12.1	3.1	5.9	77.7	80.6
	fsec-ext	TCA	2.0	LR	19,106	76	39	1740	66.1	8.3	4.2	7.9	76.8	78.8
	fsec-tfidf+	–	0.0	NB	16,238	80	35	4608	69.6	22.1	1.7	3.3	73.5	81.4
	fsec-tfidf	–	1.5	LR	13,360	90	25	7486	78.3	35.9	1.2	2.3	70.5	76.3
Wicket	fsec-ext+	TCAI	2.0	LR	427	5	1	67	83.3	13.6	6.9	12.8	84.9	82.9
	fsec-ext	TCAI	1.0	SVM	433	4	2	61	66.7	12.3	6.2	11.3	75.7	77.2
	fsec-tfidf+	–	1.0	SVM	308	5	1	186	83.3	37.7	2.6	5.1	71.3	72.8
	fsec-tfidf	–	0.5	NB	310	2	4	184	33.3	37.2	1.1	2.1	43.5	37.9
Ambari	fsec-ext+	TCAI	0.0	KNN	366	5	2	127	71.4	25.8	3.8	7.2	72.8	72.4
	fsec-ext	TCAI	1.5	RF	466	5	2	27	71.4	5.5	15.6	25.6	81.4	74.2
	fsec-tfidf+	–	1.0	NB	345	4	3	148	57.1	30.0	2.6	5.0	62.9	60.5
	fsec-tfidf	–	2.0	NB	289	4	3	204	57.1	41.4	1.9	3.7	57.9	56.5
Camel	fsec-ext+	CA	0.5	LR	407	12	6	75	66.7	15.6	13.8	22.9	74.5	80.0
	fsec-ext	TCA	2.0	NB	431	9	9	51	50.0	10.6	15.0	23.1	64.1	67.3
	fsec-tfidf+	–	0.5	LR	246	12	6	236	66.7	49.0	4.8	9.0	57.8	63.4
	fsec-tfidf	–	1.5	NB	268	9	9	214	50.0	44.4	4.0	7.5	52.7	51.2
Derby	fsec-ext+	C	0.0	LR	426	26	16	32	61.9	7.0	44.8	52.0	74.3	76.7
	fsec-ext	TCA	0.5	RF	361	29	13	97	69.0	21.2	23.0	34.5	73.6	78.2
	fsec-tfidf+	–	0.0	NB	293	31	11	165	73.8	36.0	15.8	26.1	68.5	75.6
	fsec-tfidf	–	2.0	LR	405	23	19	53	54.8	11.6	30.3	39.0	67.6	70.8
ODCV	fsec-ext+	TC	0.0	KNN	212	178	117	72	60.3	25.4	71.2	65.3	66.7	72.2
	fsec-ext	TC	1.0	RF	115	258	185	20	58.2	14.8	92.8	71.6	69.2	72.3
	fsec-tfidf+	–	0.0	KNN	201	200	95	83	67.8	29.2	70.7	69.2	69.3	71.9
	fsec-tfidf	–	1.0	LR	224	180	115	60	61.0	21.1	75.0	67.3	68.8	72.5
Apache	fsec-ext+	TC	0.5	LR	633	578	41	31	93.4	4.7	94.9	94.1	94.3	95.1
	fsec-ext	TC	1.5	RF	638	563	56	26	91.0	3.9	95.6	93.2	93.4	96.2
	fsec-tfidf+	–	1.5	LR	621	511	108	43	82.6	6.5	92.2	87.1	87.7	90.5

Table 12 (continued)

Target	Method	Cat	Ratio	Learner	TN	TP	FN	FP	Pd	Pf	Prec	F-score	G-measure	AUCROC
	fsec-tfidf	–	1.0	LR	616	517	102	48	83.5	7.2	91.5	87.3	87.9	91.2
Mozilla	fsec-ext+	TCAI	0.0	LR	1616	1192	420	48	73.9	2.9	96.1	83.6	84.0	86.6
	fsec-ext	TCAI	2.0	LR	1637	1136	476	27	70.5	1.6	97.7	81.9	82.1	85.2
	fsec-tfidf+	–	0.5	LR	1511	1197	415	153	74.3	9.2	88.7	80.8	81.7	86.7
	fsec-tfidf	–	0.5	LR	1520	1186	426	144	73.6	8.7	89.2	80.6	81.5	86.8
Comm	fsec-ext+	TC	0.5	LR	400	13	1	59	92.9	12.9	18.1	30.2	89.9	90.3
	fsec-ext	TCAI	1.0	NB	410	8	6	49	57.1	10.7	14.0	22.5	69.7	78.2
	fsec-tfidf+	–	1.5	LR	343	10	4	116	71.4	25.3	7.9	14.3	73.0	76.9
	fsec-tfidf	–	2.0	NB	308	9	5	151	64.3	32.9	5.6	10.3	65.7	73.9

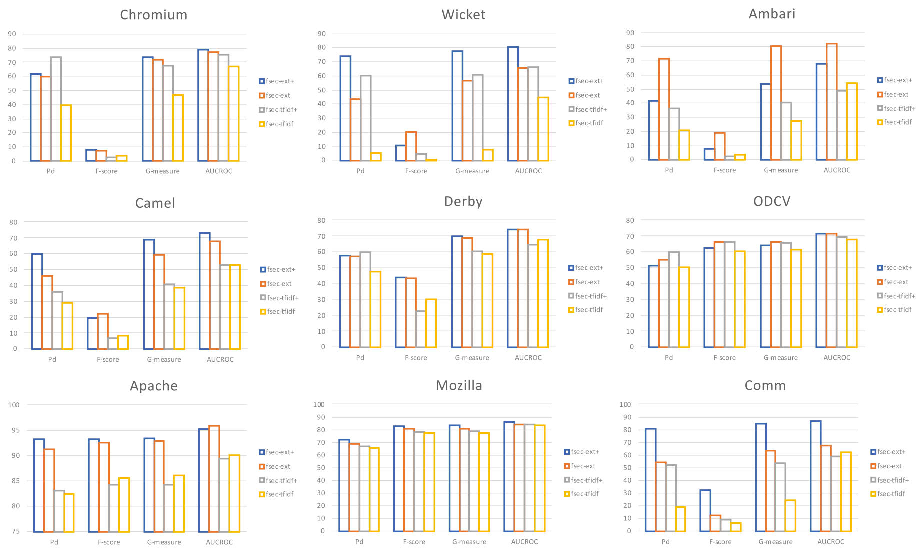

Fig. 3 Mean of recall, f-score and g-measure of top 20 results for WPP

most stable, ranging between 67 and 94%. In comparison with the other models that have minimum AUC-ROC below 45%, *fsec-ext+* models show the best classification results at different thresholds.

In terms of security category used for training, the threat (T) and control (C) categories dominate the best result. The exceptions are Ambari and Camel that include the asset (A) category and Mozilla with the asset (A) and implicit (I) categories. Camel, for example, contains security bugs that feature terms in the asset category such as ipaddress, token and password. This is plausible as Camel is a rule-based routing and mediation engine where such keywords frequently feature.

Results in Table 9 compliment the results in Table 8 with *fsec-ext+* producing the best results for g-measure in seven out of nine projects. However, model *fsec-tfidf+* fair slightly better for Ambari and Camel. The ratio of SBR to NSBR varies as well as the machine learning algorithms. We cannot infer any specific pattern as different ratio and algorithm work for different projects and feature methods.

The result in Table 8 shows that model *fsec-ext+* thrives well for projects Chromium, Ambari, Derby, Wicket, Camel and Comm where security bug reports are scarce as well as projects with balanced SBR to NSBR—Apache, ODCV and Mozilla. In general, we can infer that model *fsec-ext+* generalises better than models *fsec-ext* and *fsec-tfidf* on all unseen and independent dataset. As observed in the result (Table 8), the effect of including the security keywords as records in the training dataset produced models that can generalise on unseen security instances in a different project's dataset.

Hypothesis testing—H1: fsec-ext generalises significantly better than fsec-tfidf

We test the null hypothesis H_0: *fsec-ext+* $\leq$ *fsec-tfidf+* for transfer project predictions (TPP). We use Table 10 to present our results. It lists the p values for the Wilcoxon test performed on the hypothesis and the Hedges, g values for the effect size. Results show that the mean values for recall, f-score and g-measure for model *fsec-ext+* are higher and statistically significant than those for model *fsec-tfidf+* with the exception of

Table 13 H3—Wilcoxon and effect size tests comparing models *fsec-ext and fsec-tfidf* (WPP)

		fsec-ext+ *vs. fsec-tfidf*+		*fsec-ext vs. fsec-tfidf*	
	Metrics	*p* value	Hedges, *g*	*p* value	Hedges, *g*
Chromium	recall (Pd)	0.99	– 1.28	0.0002	1.07
	f-score	< 0.0001	3.08	0.0002	1.04
	g-measure	< 0.0001	1.41	< 0.0001	1.62
Wicket	recall (Pd)	0.002	1.05	< 0.0001	3.03
	f-score	< 0.0001	2.10	< 0.0001	2.70
	g-measure	< 0.0001	2.67	< 0.0001	4.03
Ambari	recall (Pd)	0.16	0.27	< 0.0001	3.86
	f-score	< 0.0001	2.93	< 0.0001	4.17
	g-measure	0.002	0.99	< 0.0001	4.68
Camel	recall (Pd)	< 0.0001	1.60	< 0.0001	1.59
	f-score	< 0.0001	4.63	< 0.0001	2.54
	g-measure	< 0.0001	3.41	< 0.0001	2.02
Derby	recall (Pd)	0.71	-0.26	0.0006	1.08
	f-score	< 0.0001	3.01	< 0.0001	1.50
	g-measure	< 0.0001	1.92	< 0.0001	1.76
ODCV	recall (Pd)	0.99	-1.58	0.02	0.68
	f-score	0.99	-1.27	< 0.0001	1.21
	g-measure	0.99	-0.79	< 0.0001	1.33
Apache	recall (Pd)	< 0.0001	3.16	< 0.0001	3.15
	f-score	< 0.0001	5.72	< 0.0001	6.43
	g-measure	< 0.0001	3.82	< 0.0001	5.90
Mozilla	recall (Pd)	< 0.0001	1.51	0.01	0.76
	f-score	< 0.0001	2.57	< 0.0001	1.57
	g-measure	< 0.0001	2.40	< 0.0001	1.41
Comm	recall (Pd)	< 0.0001	2.18	< 0.0001	2.00
	f-score	< 0.0001	3.01	< 0.0001	1.43
	g-measure	< 0.0001	5.02	< 0.0001	2.12

recalls for Ambari and Camel. The effect sizes are medium to large between 0.84 and 4.54 across all the metrics. The effect size test validates the result of our hypotheses showing that the differences in the mean are indeed not trivial and suggests that model *fsec-ext*+ are better predictors of security bug reports in different software projects compared with models *fsec-tfidf* and *fsec-tfidf*+. Similarly, the significant tests between *fsec-ext* vs. *fsec-tfidf* show *fsec-ext* to have average recalls, f-scores and g-measures that are higher and statistically significant than model *fsec-tfidf* and mostly with large effect sizes (1.22–4.66). Three cases—f-score for Chromium, recall for Ambari and recall for Camel—have medium effect sizes.

In summary, using the statistical test results, we can strongly suggest that models that use the harvested security keywords and include those keywords in the training sample can, on the average, predict better than models that use project-specific terms.

Table 14 WPP results with the best g-measure in comparison to the best results from Peters et al. (2017)

Target	Paper (Model)	Cat	Ratio	Learner	TN	TP	FN	FP	pd	pf	prec	f-score	g-measure
Chromium	Peters et al. (2017) (clnifarsecsq)	–		MLP	20,066	57	58	789	49.6	3.8	6.7	11.9	65.4
	fsec-ext+	TCA	0.5	KNN	18,325	80	35	2521	69.6	12.1	3.1	5.9	77.7[a]
	fsec-ext	TCA	2.0	LR	19,106	76	39	1740	66.1	8.3	4.2	7.9	76.8
	fsec-tfidf+	–	0.0	NB	16,238	80	35	4608	69.6	22.1	1.7	3.3	73.5
	Peters et al. (2017) (train)	–	0.0	LR	20,815	18	97	40	15.7	0.2	31.0	20.8	27.1
	fsec-tfidf	–	0.0	NB	20,387	11	104	459	9.6	2.2	2.3	3.8	17.4
Wicket	Peters et al. (2017) (farsectwo)	–		LR	313	4	2	181	66.7	36.6	2.2	4.2	65.0
	fsec-ext+	TCAI	2.0	LR	427	5	1	67	83.3	13.6	6.9	12.8	84.9
	fsec-ext	TCAI	1.0	SVM	433	4	2	61	66.7	12.3	6.2	11.3	75.7
	fsec-tfidf+	–	1.0	SVM	308	5	1	186	83.3	37.7	2.6	5.1	71.3
	Peters et al. (2017) (train)	–	0.0	NB	459	1	5	35	16.7	7.1	2.8	4.8	28.3
	fsec-tfidf	–	0.0	NB	459	1	5	35	16.7	7.1	2.8	4.8	28.3
Ambari	Peters et al. (2017) (farsectwo)	–		RF	478	4	3	15	57.1	3.0	21.1	30.8	71.9
	fsec-ext+	TCAI	0.0	KNN	366	5	2	127	71.4	25.8	3.8	7.2	72.8
	fsec-ext	TCAI	1.5	RF	466	5	2	27	71.4	5.5	15.6	25.6	81.4[b]
	fsec-tfidf+	–	1.0	NB	345	4	3	148	57.1	30.0	2.6	5.0	62.9
	Peters et al. (2017) (train)	–	0.0	MLP	485	1	6	8	14.3	1.6	11.1	12.5	24.9
	fsec-tfidf		0.0	NB	407	2	5	86	28.6	17.4	2.3	4.2	42.5
Camel	Peters et al. (2017) (farsectwo)	–		LR	280	9	9	201	50.0	41.8	4.3	7.9	53.8
	fsec-ext+	CA	0.5	LR	407	12	6	75	66.7	15.6	13.8	22.9	74.5
	fsec-ext	TCA	2.0	NB	431	9	9	51	50.0	10.6	15.0	23.1	64.1
	fsec-tfidf+	–	0.5	LR	246	12	6	236	66.7	49.0	4.8	9.0	57.8
	Peters et al. (2017) (train)	–	0.0	LR	464	2	16	17	11.1	3.5	10.5	10.8	19.9
	fsec-tfidf	–	0.0	NB	396	5	13	86	27.8	17.8	5.5	9.2	41.5
Derby	Peters et al. (2017) (farsectwo)	–		RF	401	20	22	57	47.6	12.4	26.0	33.6	61.7
	fsec-ext+	C	0.0	LR	426	26	16	32	61.9	7.0	44.8	52.0	74.3
	fsec-ext	TCA	0.5	RF	361	29	13	97	69.0	21.2	23.0	34.5	73.6

Table 14 (continued)

Target	Paper (Model)	Cat	Ratio	Learner	TN	TP	FN	FP	pd	pf	prec	f-score	g-measure
	fsec-tfidf+	–	0.0	NB	293	31	11	165	73.8	36.0	15.8	26.1	68.5
	Peters et al. (2017) (train)	–	0.0	NB	427	16	26	31	38.1	6.8	34.0	36.0	54.1
	fsec-tfidf	–	0.0	NB	431	20	22	27	47.6	5.9	42.6	44.9	63.2

[a]Adjusted threshold: TP = 57, FP = 1617, prec = 3.4

[b]Adjusted threshold: TP = 4, FP = 14, prec = 22.2

Table 15 TPP results with the best g-measure in comparison to the best results from Peters et al. (2017)

Target	Source	Paper (Model)	Cat	Ratio	Learner	TN	TP	FN	FP	pd	pf	prec	f-score	g-measure
Chromium	Ambari	Peters et al. (2017) (clnifarsecsq)	–	–	MLP	19,817	56	59	1,038	48.7	5.0	5.1	9.3	63.9
	Camel	fsec-ext+	TCAI	2.0	LR	16,737	84	31	4109	73.0	19.7	2.0	3.9	76.5[a]
	Ambari	fsec-ext	TC	1.0	LR	19,144	57	58	1702	49.6	8.2	3,2	6.1	64.4
	Camel	fsec-tfidf+	–	0.0	LR	14,578	68	47	6268	59.1	30.1	1.1	2.1	64.1
	Derby	Peters et al. (2017) (train)	–	0.0	RF	20,835	2	113	20	1.7	0.1	9.1	2.9	3.4
	Wicket	fsec-tfidf	–	0.0	NB	20,188	5	110	658	4.3	3.2	0.8	1.3	8.3
Wicket	Camel	Peters et al. (2017) (train)	–		NB	437	3	3	57	50.0	11.5	5.0	9.1	63.9
	Camel	fsec-ext+	TCAI	2.0	LR	412	6	0	82	100.0	16.6	6.8	12.8	90.9
	Chromium	fsec-ext	TA	2.0	LR	446	4	2	48	66.7	9.7	7.7	13.8	76.7
	Camel	fsec-tfidf+	–	1.5	LR	334	4	2	160	66.7	32.4	2.4	4.7	67.1
	Camel	Peters et al. (2017) (train)	–	0.0	NB	437	3	3	57	50.0	11.5	5.0	9.1	63.9
	Ambari	fsec-tfidf	–	0.0	NB	374	3	3	120	50.0	24.3	2.4	4.7	60.2
Ambari	Chromium	Peters et al. (2017) (farsecsq)	–		MLP	474	3	4	19	42.9	3.9	13.6	20.7	59.3
	Chromium	fsec-ext+	CA	1.5	SVM	456	5	2	37	71.4	7.5	11.9	20.4	80.6[b]
	Wicket	fsec-ext	C	1.5	RF	473	5	2	20	71.4	4.1	20.0	31.3	81.9
	Derby	fsec-tfidf+	–	0.0	SVM	415	6	1	78	85.7	15.8	7.1	13.2	84.9[c]
	Derby	Peters et al. (2017) (train)	–	0.0	MLP	484	2	5	9	28.6	1.8	18.2	22.2	44.3
	Chromium	fsec-tfidf	–	0.0	NB	465	4	3	28	57.1	5.7	12.5	20.5	71.2
Camel	Derby	Peters et al. (2017) (farsectwo)	–		NB	371	8	10	110	44.4	22.9	6.8	11.8	56.4
	Derby	fsec-ext+	TCAI	1.0	LR	359	13	5	123	72.2	25.5	9.6	16.9	73.3
	Derby	fsec-ext	TCAI	0.5	NB	427	10	8	55	55.6	11.4	15.4	24.1	68.3
	Derby	fsec-tfidf+	–	1.5	LR	241	14	4	241	77.8	50.0	5.5	10.3	60.9
	Derby	Peters et al. (2017) (train)	–	0.0	NB	457	3	15	24	16.7	5.0	11.1	13.3	28.4
	Wicket	fsec-tfidf		0.0	NB	445	4	14	37	22.2	7.7	9.8	13.6	35.8
Derby	Chromium	Peters et al. (2017) (clnifarsecsq)	–		NB	372	19	23	86	45.2	18.8	18.1	25.9	58.1
	Wicket	fsec-ext+	C	2.0	LR	419	27	15	39	64.3	8.5	40.9	50.0	75.5
	Chromium	fsec-ext	CA	0.5	RF	380	25	17	78	59.5	17.0	24.3	34.5	69.3

Table 15 (continued)

Target	Source	Paper (Model)	Cat	Ratio	Learner	TN	TP	FN	FP	pd	pf	prec	f-score	g-measure
	Wicket	fsec-tfidf+	–	2.0	LR	351	21	21	107	50.0	23.4	16.4	24.7	60.5
	Ambari	Peters et al. (2017) (train)	–	0.0	NB	393	13	29	65	31.0	14.2	16.7	12.7	45.5
	Wicket	fsec-tfidf	–	0.0	NB	355	19	23	103	45.2	22.5	15.6	23.2	57.1

MLP multilayer perceptron

[a]Adj. threshold: TP = 56, FP = 2422, prec = 2.2

[b]Adj. threshold: TP = 3, FP = 3, prec = 50

[c]Adj. threshold: TP = 3, FP = 6, prec = 33.3

4.3 RQ2: Does including security features as SBRs in training dataset improve model's performance over models that exclude them for both WPP and TPP?

To answer this research question, we perform a hypothesis testing on results reported by *fsec-ext*+ versus *fsec-ext* and *fsec-tfidf*+ versus *fsec-tfidf* for both WPP and TPP.

Hypothesis testing—H2: Including security terms in the training set during modelling can significantly outperform models that do not include them.

We test the null hypotheses $\mathbf{H_0}$: ***fsec-ext*+** $\leq$ ***fsec-ext*** and $\mathbf{H_0}$: ***fsec-tfidf*+** $\leq$ ***fsec-tfidf***. We use Table 11 to answer RQ2. It lists the p values for the Wilcoxon test performed on the hypotheses and the Hedges, g values for the effect size.

4.3.1 TPP

We observe a clear pattern of significant results for *fsec-ext*+ and *fsec-tfidf*+ when using TPP. All recall results in TPP are significant with medium to large effect sizes. The exception is Ambari where there is a small effect size showing that the mean recall of *fsec-ext*+ is only slightly higher than *fsec-ext*. G-measures show effect sizes between medium and large for six projects (Chromium, Wicket, Camel, Derby, ODCV, Mozilla) indicating a fair model with a relative balance between recalls and true negative rates. We observe worse test and effect size results for f-score for model *fsec-ext*+ for projects with few SBRs (Chromium, Wicket, Ambari, Camel, and Derby) whereas the results are better for *fsec-tfidf*+ in some cases. Examples are Chromium, Camel and Derby. One reason for this difference is the project-specific nature of the features used by *fsec-tfidf*+. *fsec-ext*+ uses externally sourced features resulting in higher false positives and consequently low precisions and low f-scores. Further study can investigate how to combine both project-specific and externally sourced features to both maximise recalls and f-scores.

Clearly, the test results demonstrate that including features in the training dataset produce superior results to models that exclude them when transferring model from one project to another.

4.3.2 WPP

Here, the results are mixed. For *fsec-ext*+, we observe that including security features in training clearly worsen the results in some of the projects (Ambari, Derby, ODCV). For instance, the result in Ambari indicates that the metrics for *fsec-ext* fair better as the mean values are significantly greater than *fsec-ext*+ as indicated by the negative Hedges, g values. Model *fsec-ext*+ also improves results in a few projects (Comm and Camel). The rest of the projects (Chromium, Wicket, Apache and Mozilla) contain mixed results. We also reckon from the results that *fsec-ext*+ increases recalls in six out of the nine projects albeit with mixed effect sizes ranging from negligible to medium.

Model *fsec-tfidf*+ significantly improve *fsec-tfidf* across all metrics in four projects (Wicket, ODCV, Mozilla, and Comm) while it worsens f-scores in Ambari, Camel, Derby and Apache. Model *fsec-tfidf*+ also significantly improve recall and g-measure results in Chromium with medium and large effect sizes. However, the f-score result is not significant, and the effect size is negligible although they are slightly higher than *fsec-tfidf*. Our observation from the WPP result is that model built with project-specific features (*fsec-tfidf*) fair better when treated (*fsec-tfidf*+) than model built with externally sourced features (*fsec-ext*) when treated (*fsec-ext*+). One reason may be that WPP model captures project-specific terms that are not possible with externally sourced features and therefore can predict better on own test dataset.

Overall, using the statistical test results, we can strongly suggest that models that include security keywords in the training sample can, on the average, generalise better than models that exclude them during model construction. When predicting within project, including security keywords can improve results especially by using project-specific features.

4.4 RQ3: Do models that utilise external sources for features outperform the model that uses project-specific features in within-project prediction (WPP)?

As listed in Table 12, *fsec-ext*+ has the best g-measure in seven out of the nine projects. In ODCV, *fsec-tfidf*+ records a slightly higher g-measure over *fsec-ext*+ and *fsec-ext*. The mean of top 20 values for the metrics (see Fig. 3) further reveals *fsec-tfidf*+ to perform better compared with *fsec-ext*+ for ODCV. Similarly, *fsec-tfidf*+ produces higher mean recalls for Chromium and Derby. Otherwise, *fsec-ext*+ records higher mean values for the metrics for the remaining projects. Different combinations of security category work for the different *fsec-ext* models demonstrating the usefulness of separating the categories. In addition, the ratio SBR to NSBR is different for the models suggesting that finding a useful model will require searching for which SBR to NSBR ratio is best for a given dataset and feature selection type. AUC of ROC also shows the same pattern similar to other metrics where the mean for AUC is higher for *fsec-ext*+ or *fsec-ext*. We can establish that models that use externally sourced features are more robust with the highest AUC at different thresholds.

In general, the results show that models that use the harvested security-related keywords as a feature set perform relatively and reasonably well on their own oracle dataset in terms of recall, f-score and g-measure. The f-scores suffer reduced performance in general for most of the projects with fewer cases of security bug reports in their dataset. Although, some f-scores can be judged to be relatively high. An example is Derby with 42 SBRs where 26 SBRs are correctly classified, and 32 NSBRs are incorrectly classified as SBR giving a 52% f-score. This translates to reviewing 58 bug reports if we are concerned with type I error. Similarly, the g-measure also shows a relatively useful model where majority of security bug reports are correctly classified (true positive rate) and majority of non-security bug reports are also correctly classified (true negative rate). As an example, the Comm performance shows 93% recall and a true negative rate of 87% (59 misclassified non-security bug reports out of 459).

Hypothesis testing—H3: using externally sourced features significantly outperform project-specific features for WPP

We test the null hypothesis $\mathbf{H_0}$: ***fsec-ext***+ $\leq$ ***fsec-tfidf***+ and $\mathbf{H_0}$: ***fsec-ext*** $\leq$ ***fsec-tfidf*** for WPP. Without treatment, models built with externally sourced features (*fsec-ext*) significantly outperformed those built with project-specific features (*fsec-tfidf*) in all projects. Across all metrics, the difference in mean values also has large effect sizes showing that the differences in the means values are not trivial. Only the recalls for ODCV and Mozilla have medium effect sizes. We can conclude that in the absence of treatments or noise filtering before training, models built on externally sourced features will be a better choice by default.

With treatment, *fsec-ext*+ shows significantly higher mean across the three metrics (g-measure, f-score, recall) than *fsec-tfidf*+ in five projects—Wicket, Camel, Apache, Mozilla and Comm—and with large effect sizes. However, *fsec-tfidf*+ outperformed *fsec-ext*+ only in ODCV and the recalls for Chromium and Derby as reflected by the negative effect sizes (see also Fig. 3). Otherwise, *fsec-ext*+ outperformed *fsec-tfidf*+ in the remaining metrics and with large effect sizes.

The results suggest that using externally sourced features can reliably predict SBRs when used in WPP and with better results in most cases than project-specific features. The mixed results in Chromium and Derby however, hint on the potential for improved results over both models when the features are combined. This remains a possible future work.

4.5 Comparing our results with Peters et al. (2017)

To compare our results fairly with Peters et al. (2017), we remove the other four projects (ODCV, Apache, Mozilla, and Comm) before we run our analysis for this comparison. We report the result with the best g-measure of using the feature selection method used in Peters et al. (2017) with treatment—fsec-tfidf+ (i.e. including features in the training set and adjusting the ratio of NSBR relative to SBR) and without treatment—*fsec-tfidf*. We note here that *fsec-tfidf* (without treatment) corresponds to the 'Peters et al. (2017) (train)'. We report the best overall results from Peters et al. (2017) when their filtering techniques are used as well as the best result for the 'train' benchmark. We report the best results of our generalised feature selection method (*fsec-ext*). We then compare the results of our best models on the five projects—Chromium, Wicket, Ambari, Camel and Derby. We use similar metrics which are the g-measure, f-score, probability of false alarm (pf) and recall (pd) to compare which result is better to the other.

In the case of WPP (Table 14), *fsec-ext*+ improves g-measure by 1–21% over the best results in all of the five projects (100%) reported in Peters et al. (2017). Similarly, for recall (Pd), our model outperformed (Peters et al., 2017) in all the projects by 14–20%. For f-score, our model score better in three out of the five projects. Similarly, for TPP (Table 15), *fsec-ext*+ improves g-measure by 13–27% over the best results in all of the five projects (100%) reported in Peters et al. (2017). Interestingly, for Ambari, our feature inclusion method (*fsec-tfidf*+) and adjusted ratio also improve the feature selection method in Peters et al. (*fsec-tfidf*) by 13.7%. For recall, fsec-ext+ improves by 19–50% over the best results in Peters et al. (2017) For f-score, our model scores better in three out of the five projects.

We note here, also, that applying our treatment to the same feature selection method used in Peters et al. improved both the g-measure and recall in 90% cases over the best results reported in Peters et al. (2017). However, in cases where business users care more about precision, we use the TP of Peters et al. (2017) as threshold to compute the FP equivalent for our best model. For WPP, *fsec-ext*+ reports 3.4% precision (see footnote in Table 14) for Chromium as against 6.7% reported by Peters et al. (2017) Similarly, for TPP, *fsec-ext*+ reports a precision of 2.2% (see footnote in Table 15) as against 5.1% reported by Peters et al. (2017) In this context, (Peters et al., 2017) score better than our model. For Ambari, our model is better than (Peters et al., 2017) when we apply the same threshold.

Overall, the results from our experiment demonstrate that using the model *fsec-ext*+ can be robust, generalisable and superior to existing state-of-the-art prediction models for classification of security bug reports in software development projects. It remains to see whether combining *fsec-ext*+ with FARSEC filtering approaches can improve *fsec-ext*+. We reserve this investigation as a future study.

5 Discussion

5.1 Project-specific vocabularies and explicit security keywords

We assess a sample of the records that are predicted to be false negatives by *fsec-ext*. We find that specific security-related project terms are used in many of these messages. Some of the examples we found are: 'SecureServerTest', 'NSSecurityMechanismTest', 'testOSReadOnly', 'DefaultShutdownStrategy', 'camel-xmlsecurity'. In Mozilla, among issues classified as false negatives, we found many issues identified by the team as security related but lacking the traditional security terms. Examples including '*Link Visitedness can be detected by redraw timing*', '*Cross-domain drag and drop across IFrames*' or '*Page can stomp on PRIMARY clipboard by calling select() at the right time*' are classified as medium security issues. These cases present limitations for our approach because the project-specific terms are missing and would need to be represented during model construction (Morrison et al., 2018b). One approach we hope to further investigate is to detect project-specific vocabularies by using *fsec-tfidf* feature selection approach and combine with the externally harvested keyword approach to improve the model's performance.

5.2 Security categories

Our results point to the usefulness of separating the security categories for building prediction models. We observe that different combinations of the categories work for different projects. Prior to our work and to the best of our knowledge, no study has separated features the way we have done—i.e. into threat, control, asset and implicit, and combining them to build a classifier. This can also provide insight into dominant security issues in a project. For instance, the best model for Camel is based on control and asset features. This new technique provides a new approach to separate features before applying them to build models. An interesting extension will be to look into how to extend our approach to automatically separate project-specific features into these categories before combining them with our externally sourced features. We aim to investigate word embedding (Dai et al., 2017) in the future for this purpose.

5.3 When data scarcity is an issue

As noted by Peters et al. (2017), there exist a core set of security-related keywords which can predict security bug reports on any project. However, we establish that these core security keywords can vary from project to project in terms of their predicting power. We show that by separating these keywords into different security categories, we can further refine models per projects. Our result shows that by using our harvested security-related terms and by including them in the training dataset, it is possible to build prediction models on small dataset that can be successfully transferred to other projects irrespective of their sizes. This is the case with all the projects—Camel, Derby, Wicket, Ambari and Comm—where security bug reports are scarce. Our approach can also prove useful for projects with no prior labels for security bug reports in their bug report repositories. As discussed in Cois and Kazman (2015), very few projects (1.4%) explicitly identify security-related issues when they analysed 400,000 projects on GitHub. Our set of security keywords can be used for auto labelling, and a WPP model can be constructed for future classification of newly reported bug reports.

5.4 Implications for practice and research

5.4.1 Practice

We have deployed the project[10] as open source on both github and bitbucket to provide an off-the-shelf add-on on JIRA platform. As an off-the-shelf add-on, it offers the possibility for immediate installation and usage. The idea of transferring the research results via a platform that is widely and popularly used is to increase the visibility and impact on the security consciousness across all projects. We believe that the add-on can be used to access the security portfolio of a system prior to a buy or build decision. In addition, the model/add-on can be used to collect useful metrics such as records of security-related issues that are still open, records of security-related issues that have been changed and the window of exposure of identified security-related records. The approach we have employed provides the possibility of adaptation to include 'implicit' security-related terms and generate an organisational-based classification model that can be used across projects within an organisation.

5.4.2 Research

Researchers have long conducted studies to develop text classification models. Our modelling approach and eventual add-on provide more possibilities for further research studies in this direction. First by making it possible to quickly collect security related messages from different projects. Second, by making it possible to apply the same approach to prediction in related areas where domain knowledge exists, and feature set can be extracted upfront.

6 Threats to validity

Manual term extraction: we manually harvested security-related terms from many popular sources. However, this is not exhaustive as there are many other sources where additional security-related keywords could be extracted.

Manual labelling of security-related records: manually labelled security records including those reported in publicly disclosed venues such as the MITRE CVE have been argued to be subjective and sometimes inaccurate (Gegick et al., 2010; Scandariato et al., 2014; Massacci & Nguyen, 2010; Christey & Martin, n.d.; Massacci & Nguyen, 2010; Dai et al., 2017) showed that using different vulnerability databases can yield different results. Further studies will be useful on the dataset we have used to replicate our study.

Modelling: We have used default settings for the parameters of the algorithms during model construction. It is possible that by tuning the parameters, different results might be obtained (Xia et al., 1805). As mitigation, we have performed our experiments under the same conditions, and we could expect further tuning of the algorithm to produce results in the same direction we have reported in this work. Additionally, we have only explored a subset of imbalance ratio. It is possible that higher SBR to NSBR ratio worsens or improves the results.

[10] https://github.com/tosdanoye/fsecext and https://bitbucket.org/ootos/jirasecplugin

Categorisation of terms: our categorisation of security-related terms is subjective and prone to inaccuracies. It is possible to find some of the terms we have included in the implicit category to be directly related to attack/threat and control categories.

Generalisation: Although we have evaluated our approach on 8 major open source datasets and one commercial dataset, we can still not claim generalisation of the result to all types of datasets. Further empirical studies are necessary to validate our result on new dataset in domains we have not covered in this study.

7 Conclusion

We have investigated an approach for training a text classification model to identify security messages in heterogeneous software project repositories such as issue tracking and version control systems. Due to the problem of unseen security instances in different projects, we proposed harvesting security keywords from relevant security sources as features and including them as security-related records in the training set during modelling versus typical text classification approaches that determine promising features from the project's dataset. We provide meaning to our feature set by grouping the extracted terms into different categories of asset, attack/threat, control/mitigation and implicit. We then use these groups as treatments in our experiments by combining them to derive classification models.

Our evaluation of these approaches shows that using a broad security keyword as feature set during model construction can out-perform the approach that uses project-specific features. Further, including these security keywords as SBRs in the training dataset produced models that generalised better on unseen and heterogeneous datasets. In addition, breaking down security features into different categories reveal that different projects perform better with different combinations of security categories. Finally, we compare our results with a state-of-the-art result for both WPP and TPP. Results demonstrate that our modelling approaches produce superior results in all of the projects evaluated.

In the future, we aim to investigate (1) using an unsupervised learning approach in combination with sentiment analysis of discussions and *fsec-tfidf* features to augment generic security keywords with project-specific keywords and (2) using our approach for risk-estimation during software development and maintenance. Our idea is that the categories we have formulated can be useful to derive a risk model for a project. For instance, it might be possible to compute risk = likelihood × Impact, where the likelihood can be approximated from the vulnerabilities and existing mitigations, and the impact can be derived from assets and the threats/attacks to these assets. In addition, it might be possible to assign weights to these categories during classification. Keywords in asset and attack categories could be assigned higher weights as compared with control and implicit keywords. We believe this could lay a good foundation for building classification tools with useful feedback and interesting use cases. Lastly, we want to investigate whether building a model on all the remaining projects' training dataset to predict a target will produce a better result. In addition, we want to investigate whether a bellwether effect exists in our TPP as this effect has been observed in many transfer learning tasks of software engineering.

Funding Open Access funding provided by Western Norway University Of Applied Sciences. This work is supported by the Research Council of Norway through the project SoS-Agile: Science of Security in Agile Software Development (247678/O70).

References

Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug? In: Proceedings of the 28th international conference on Software engineering, ACM, pp. 361–370.

Bozorgi, M., Saul, L. K., Savage, S. & Voelker, G. M. (2010). Beyond heuristics: learning to classify vulnerabilities and predict exploits. In Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 105–114.

Chillarege, R., Bhandari, I. S., Chaar, J. K., Halliday, M. J., Moebus, D. S., Ray, B. K., & Wong, M. Y. (1992). Orthogonal defect classification-a concept for in-process measurements. *IEEE Transactions on Software Engineering, 18*(11), 943–956.

Christey, S., & Martin, B. (n.d.). Buying into the bias: why vulnerability statistics suck, BlackHat, Las Vegas, USA, Tech. Rep 1.

Cleland-Huang, J., Settimi, R., Zou, X., & Solc, P. (2006). The detection and classification of non-functional requirements with application to early aspects. In Requirements Engineering, 14th IEEE International Conference, Minneapolis, Minnesota, pp. 39–48. https://doi.org/10.1109/RE.2006.65

Cois, C. A., & Kazman, R. (2015). Natural language processing to quantify security effort in the software development lifecycle. In SEKE, pp. 716– 721.

Dai, X., et al. (2017). From social media to public health surveillance: word embedding based clustering method for twitter classification. In SoutheastCon pp.1–7.

Debole, F., & Sebastiani, F. (2004). Supervised term weighting for automated text categorization. In Text mining and its applications, Springer, pp. 81–97.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research, 7,* 1–30.

Ellison, R. J., Goodenough, J. B., Weinstock, C. B., & Woody, C. (2010). *Evaluating and mitigating software supply chain security risks*. Tech. rep.: CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Feinerer, I. (2013). Introduction to the tm Package Text Mining in R. Accessible enligne: http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf

Forman, G. (2003). An extensive empirical study of feature selection metrics for text classification, *Journal of machine learning research, 3,* 1289–1305.

Gegick, M., Rotella, P., & Xie, T. (2010). Identifying security bug reports via text mining: An industrial case study. In Mining software repositories (MSR), 2010 7th IEEE working conference on, IEEE, pp. 11–20.

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering, 21*(9), 1263–1284.

Hindle, A., Ernst, N. A., Godfrey, M. W., & Mylopoulos, J. (2013). Auto- mated topic naming. *Empirical Softw. Engg., 18*(6), 1125–1155. https://doi.org/10.1007/s10664-012-9209-9.doi:10.1007/s10664-012-9209-9

Joachims, T. (1998). Text categorization with support vector machines: learning with many relevant features, Machine learning: ECML-98, 137–142.

Kampenes, V. B., Dybå, T., Hannay, J. E., & Sjøberg, D. I. K. (2007). A systematic review of effect size in software engineering experiments. *Information and Software Technology, 49*(11–12), 1073–1086.

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In Proceedings of the 31st International Conference on Machine Learning (ICML-14), pp. 1188–1196.

Louppe, G. (2014). Understanding random forests: from theory to practice. arXiv preprint arXiv: 1407.7502.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Scoring, term weighting and the vector space model. *Introduction to information retrieval, 100*(2–4), 100–123.

Massacci, F., & Nguyen, V. H. (2010). Which is the right source for vulnerability studies?: an empirical analysis on Mozilla Firefox. In Proceedings of the 6th International Workshop on Security Measurements and Metrics, ACM, p. 4.

Morrison, P., Oyetoyan, T. D., & Williams, L. (2018b). Identifying security issues in software development: are keywords enough? In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, pp. 426–427.

Morrison, P. J., Pandita, R., Xiao, X., Chillarege, R., & Williams, L. (2018a). Are vulnerabilities discovered and resolved like other defects? *Empirical Software Engineering, 23*(3), 1383–1421.

Nigam, K., McCallum, A. K., Thrun, S., & Mitchell, T. (2000). Text classification from labeled and unlabeled documents using em. *Machine learning, 39*(2), 103–134.

Ohira, M., Kashiwa, Y., Yamatani, Y., Yoshiyuki, H., Maeda, Y., Lim- settho, N., Fujino, K., Hata, H., Ihara, A., & Matsumoto, K. (2015). A dataset of high impact bugs: manually-classified issue reports. In Mining Soft- ware Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on, IEEE, pp. 518–521.

Peters, F., Tun, T., Yu, Y., & Nuseibeh, B. (2017). Text filtering and ranking for security bug report prediction. IEEE Transactions on Software Engineering.

Pletea, D., Vasilescu, B., & Serebrenik, A. (2014). Security and emotion: sentiment analysis of security discussions on github. In Proceedings of the 11th working conference on mining software repositories, ACM, pp. 348–351.

Ponemon-Institute, IBM-Security. (2017). Cost of data breach study: Global overview benchmark research sponsored by ibm security independently conducted by ponemon institute llc, Ponemon Institute Research Report.

Powers, D. (2011). Evaluation: from precision, recall and F-Factor to ROC, informedness, markedness & correlation. *J. Mach. Learn. Technol, 2*(1), 37–63.

R Development Core Team. (2008). R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, ISBN 3–900051–07–0. http://www.R-project.org

Ray, B., Hellendoorn, V., Godhane, S., Tu, Z., Bacchelli, A., & Devanbu, P. (2006). On the ”naturalness” of buggy code. In Proceedings of the 38th In- ternational Conference on Software Engineering, ICSE ’16, ACM, New York, NY, USA, pp. 428–439. https://doi.org/10.1145/2884781.2884848

Ray, B., Posnett, D., Filkov, V., & Devanbu, P. (2014). A large scale study of pro- gramming languages and code quality in github. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Soft- ware Engineering, ACM, pp. 155–165.

Riaz, M., King, J., Slankas, J., & Williams, L. (2014). Hidden in plain sight: auto- matically identifying security requirements from natural language arti- facts. In Proc. 22nd RE, IEEE, pp. 183–192.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management, 24*(5), 513–523.

Salton, G., Fox, E. A., & Wu, H. (1983). Extended Boolean information retrieval. *Communications of the ACM, 26*(11), 1022–1036.

Salton, G., & McGill, M. J. (n.d.). Introduction to modern information retrieval.

Scandariato, R., Walden, J., Hovsepyan, A., & Joosen, W. (2014). Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering, 40*(10), 993–1006.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR), 34*(1), 1–47.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation, 28*(1), 11–21.

Tyo, J. P. (2016). Empirical analysis and automated classification of security bug reports.

Unterkalmsteiner, M., Abrahamsson, P., Wang, X., Nguyen-Duc, A., Shah, S., Bajwa, S. S., Baltes, G. H., Conboy, K., Cullina, E., Dennehy D., et al. (n.d.). Software startups–a research agenda, e-Informatica Software En- gineering Journal 10 (1).

Wijayasekara, D., Manic, M., & McQueen, M. (2014). Vulnerability identification and classification via text mining bug databases. In Industrial Electronics Society, IECON 2014–40th Annual Conference of the IEEE, IEEE, pp. 3612–3618.

Witten, I. H., & Frank, E. (2005). *Data mining: practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.

Wu, H. C., Luk, R. W. P., Wong, K. F., & Kwok, K. L. (2008). Interpreting TF-IDF term weights as making relevance decisions. *ACM Trans Inf Systems (TOIS), 26*(3), 1–37.

Xia, T., Krishna, R., Chen, J., Mathew, G., Shen, X., & Menzies, T. (2018). Hyperparameter optimization for effort estimation. arXiv preprint arXiv: 1805.00336.

Yan, M., Zhang, X., Liu, C., Xu, L., Yang, M., & Yang, D. (2017). Automated change- prone class prediction on unlabeled dataset using unsupervised method. *Information and Software Technology, 92,* 1–16.

Zaman, S., Adams, B., & Hassan, A. E. (2011). Security versus performance bugs: a case study on firefox. In Proceedings of the 8th working conference on mining software repositories, ACM, pp. 93–102.

Tosin Daniel Oyetoyan Tosin Daniel Oyetoyan is an Associate Professor at the Department of Computing, Mathematics, and Physics, Western Norway University of Applied Sciences. He received his PhD in Computer and Information Science from the Norwegian University of Science and Technology (NTNU) in 2015. His research interests are in software security, agile software development, software quality and maintenance, software design, code analysis, software testing, software refactoring, software metrics, empirical software engineering, and application of machine learning in software engineering. He is a member of IEEE and IEEE Computer Society.

Patrick J. Morrison Patrick J. Morrison is a software engineer at IBM, and an adjunct professor at North Carolina State University (NCSU). He received his PhD candidate in Computer Science at NCSU under the supervision of Dr. Laurie Williams. He received the BS degree in computer science from the University of Florida and the MS degree in computer science from Florida Atlantic University His research interests are in empirical software engineering, security, and agile software development practices and processes. He has interned at Microsoft Research and IBM. He worked as a developer and consultant before returning to academia. He is a member of the ACM and IEEE.

Authors and Affiliations

Tosin Daniel Oyetoyan[1,2] · **Patrick Morrison**[3]

Patrick Morrison
pjmorris@ncsu.edu

[1] Department of Software Engineering, Safety and Security, SINTEF Digital, Trondheim, Norway

[2] Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences, Bergen, Norway

[3] Department of Computer Science, North Carolina State University, Raleigh, NC, USA