Articles

Publicaton source	Title	Year
TSE	MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction	2017
WCRE	An Incremental Update Framework for Efficient Retrieval from Software Libraries for Bug Localization	2013
TSE	Text Filtering and Ranking for Security Bug Report Prediction	2019

An Incremental Update Framework for Efficient Retrieval from Software Libraries for Bug Localization -WCRE.2013

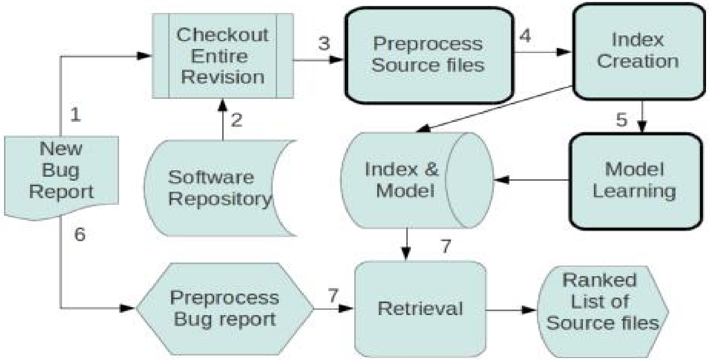


Fig. 1. A typical bug localization process shown for a single bug.

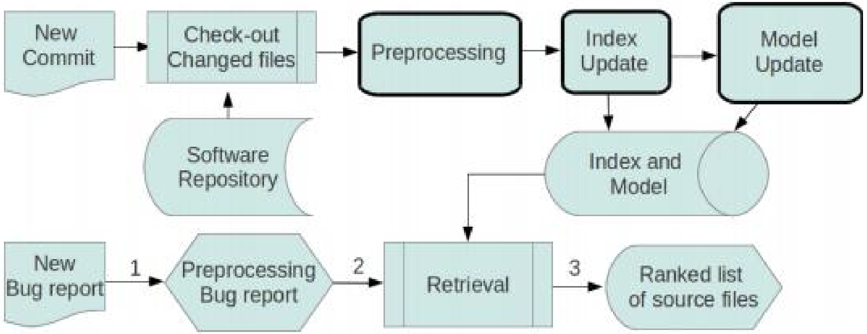


Fig. 3. Incremental update framework for bug localization.

An Incremental Update Framework for Efficient Retrieval from Software Libraries for Bug Localization

1 Text Preprocessing and Index Creation

- Addition: $A^{t+1} = [A^t Add]$.
- When the j^{th} source file is modified: $A^{t+1} = [A_1^t A_2^t ... A_j^{t+1} ... A_M^t]$
- When the j^{th} source file is deleted: $A^{t+1} = [A_1^t A_2^t ... \mathbf{0} ... A_M^t]$

$$idf(w) = log(\frac{M}{df(w)+1})$$

$$df^{t+1}(w) = df^t(w) + sign(A_m^{t+1}(w) - A_m^t(w))$$

$sign(x) = 1$ if $x > 0$, -1 if $x < 0$ and 0 if $x = 0$.

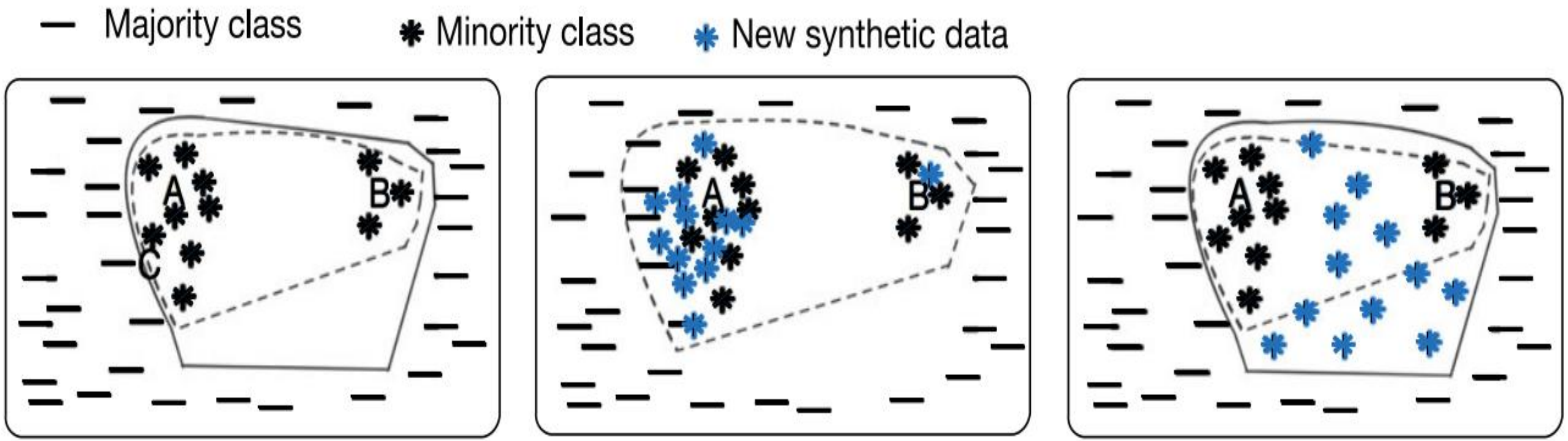
2 Learning parameters of the text model. VSM and SUM

$$\begin{aligned} p_{uni}(w|d_m) &= \mu\frac{A_m(w)}{dl(m)} + (1-\mu)p_c(w) \\ p_c(w) &= \frac{cf(w)}{\sum_w cf(w)} \qquad (2) \\ cf(w) &= \sum A_m(w) \quad \& \quad dl(m) = \sum A_m(w) \end{aligned}$$

$$\begin{aligned} cf^{t+1}(w) &= cf^t(w) + A_m^{t+1}(w) - A_m^t(w) \\ dl^{t+1}(m) &= dl^t(m) + A_m^{t+1}(w) - A_m^t(w) \end{aligned}$$

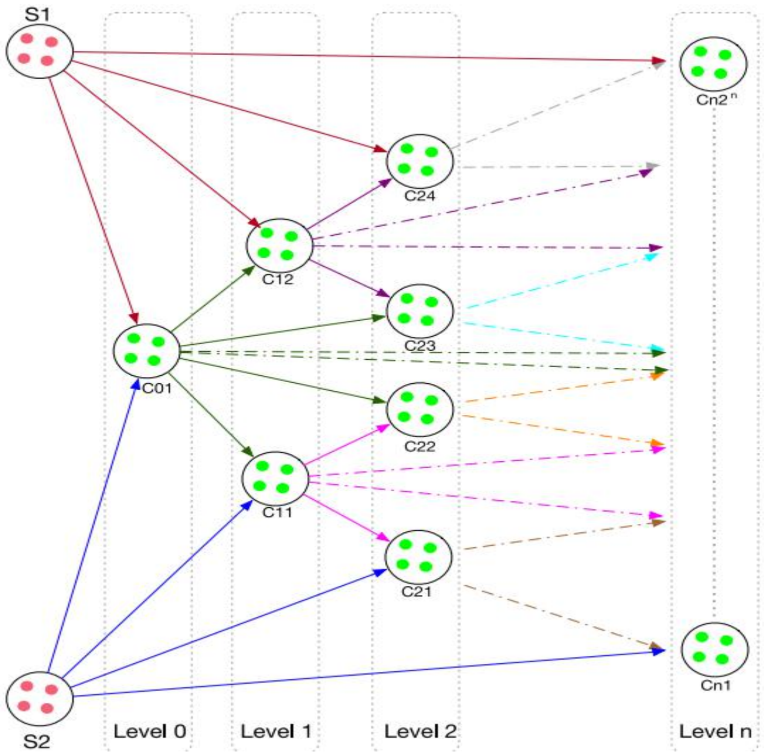
MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction -TSE.2017

Technology:data sampling approaches are synthetic based.
problem:over-generalization and boundary widened



(a) before resampling (b) after resampling (c) ideal resampled data

MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction



1

Diversity Measurement

Mahalanobis distance(D2)proposed by P. C. Mahalanobis[28].

2

Data Partitioning and Pairing

uses partitional cluster approach
The pairing is systematic order

3

Synthetic Sample Generation

computing the mean or average between two paired instances

Text Filtering and Ranking for Security Bug Report Prediction

汇报人：刘文杰

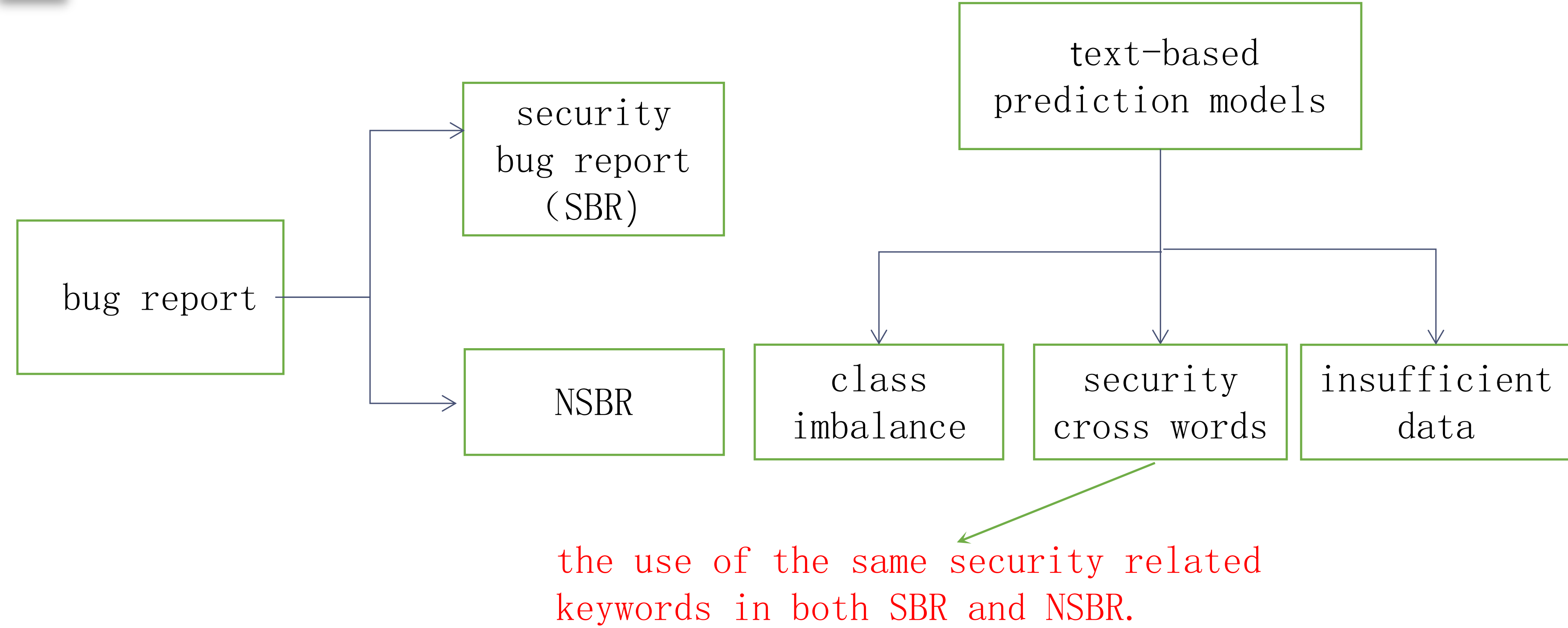
CONTENTS

01 Background and Problem

02 Structure

03 Experiments Setup

Background


bug report
security bug report (SBR)
NSBR
text-based prediction models
class imbalance
security cross words
insufficient data
the use of the same security related keywords in both SBR and NSBR.

Contributions

A

An approach to automatically identify keywords

B

An automatic filtering and ranking method

C

A tractable method to use both bug reports

D

A ranking generate a useful ranked list

FARSEC Structure

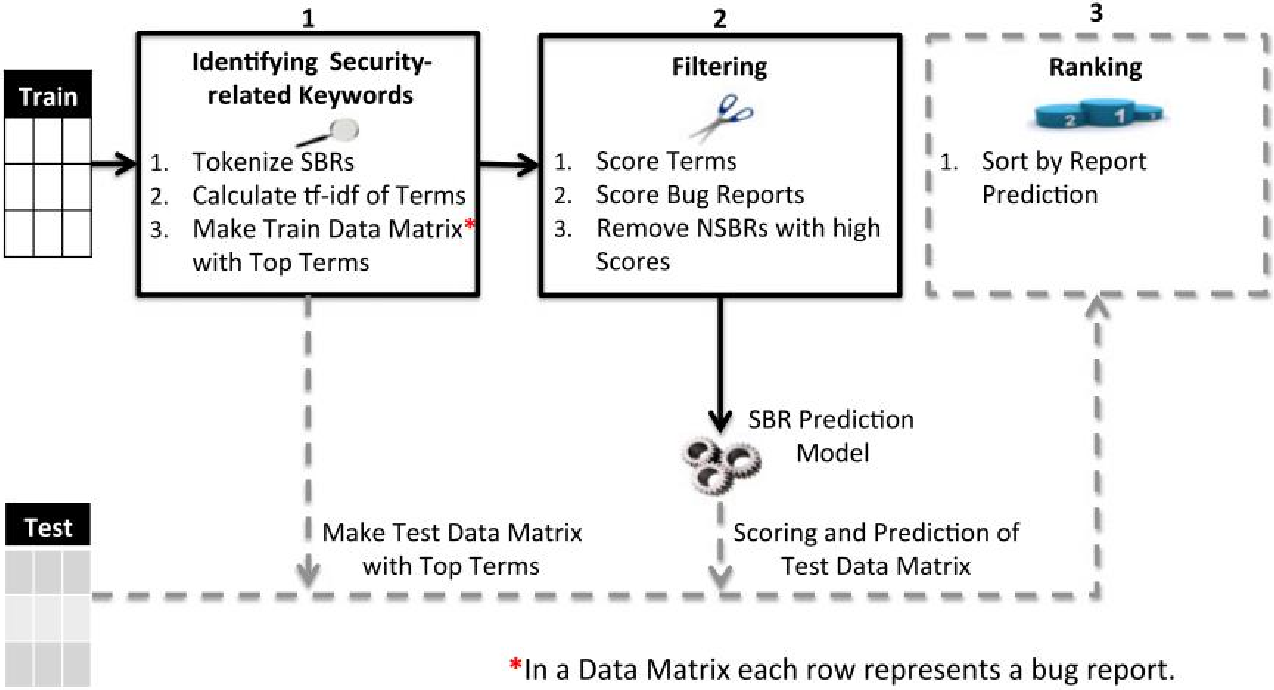
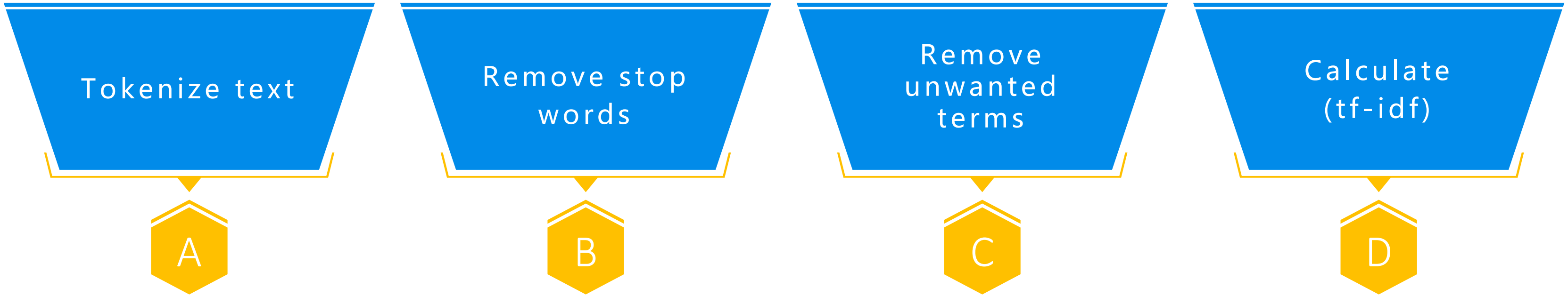


Fig. 2. Overview of the FARSEC approach.

Identifying Security Related Keywords



splitting text into sentences and words

Chromium:
&lt &quot .org
n't .html .dll
/script&gt

$$tf(t,d) = 0.5 + \frac{0.5 \times f(t,d)}{max\{f(w,d) : w \in d\}} \quad (1)$$

$$idf(t,D) = log \frac{N}{|\{d \in D : t \in d\}|} \quad (2)$$

$$tf\text{–}idf(t,d,D) = tf(t,d) \times idf(t,D). \quad (3)$$

Filtering Bug Reports

FARSEC filtering is based on the scoring of the terms in the feature set. Using these scores to calculate an overall score for bug reports.

was a poor ranking heuristic for low frequency evidence.

1) farsecsq, the frequency of words found in SBRs
2) farsectwo, whichthe frequency by two
3) farsec, which offers no support

Algorithm 1. Score Keywords

ScoreWords(B, support) {B is the bug reports data with a feature set, and support adds bias in favour of SBRs.}

Partition(B) $\mapsto \{S, NS, W\}$ {S is SBR data, NS is NSBR data and W is the feature set.}

for w in W **do**

{w represents each word in the feature set.}

$P(S_w) \leftarrow \mathrm{Min}\left(1, \frac{support(tf(S_w))}{|S|}\right)$

$\boxed{P(NS_w) \leftarrow \mathrm{Min}\left(1, \frac{tf(NS_w)}{|NS|}\right)}$

$\mathrm{Score}(w) \leftarrow \mathrm{Vector}\left(\mathrm{w}, \mathrm{Max}\left(0.01, \mathrm{Min}\left(\boxed{0.99}, \frac{P(S_w)}{P(S_w)+P(NS_w)}\right)\right)\right)$

in SBR not in NSBR

end for

return *Hashmap(Score(w))* {Returns dictionary of w mapped to Score(w).}

Filtering Bug Reports-Score Bug Report

NSBRs are selected using the threshold score of ≥ 0.75

higher scores are likely to be false positives.

Algorithm 2. Score Bug Report

ScoreReport(R, B, **support**) {R is a bug report and B is the bug reports data with a feature set, and support adds bias in favour of SBRs.}
$M \leftarrow$ **ScoreWords**(B, **support**)
$M^* \leftarrow \emptyset$ {Initialized list of scores for security related keywords in R.}
$M' \leftarrow \emptyset$ {Initialized list of complement scores for security related keywords in R.}
for w in R **do**
  $P(w) \leftarrow$ GetScore(w, M) {Returns score for each keyword (w) if present in dictionary and a score of zero if not present.}
  $M^* \leftarrow P(w)$
end for
for m in M^* **do**
  $M' \leftarrow 1 - m$
end for
return $\frac{\prod_{i=1}^{|M^*|} m_i}{\prod_{i=1}^{|M^*|} m_i + \prod_{i=1}^{|M'|} (1-m_i)}$

Ranking Bug Reports

01

goal: actual SBRs are closer to the top of the list.

02

idea:ensemble learning,which combines predictions from multiple models using another

03

For example： according to the prediction results of the farsecsq filter

Step 1: (Sort by prediction in descending order)：only if when the number of predicted SBRs is less than that of farsecsq

Step 2: (Sort by prediction of farsecsq)

other filters or no filters predict SBR

EXPERIMENT SETUP

数据集：uses JIRA6 as its bug tracking system

TABLE 1
Characteristics of the Projects and Bug Reports

Project	Domain	Start Date	End Date	BRs	SBRs	SBRs (%)
Chromium	Web browser called Chrome.	Aug 30 2008	Jun 11 2010	41,940	192	0.5
Wicket	Component-based web application framework for the Java programming.	Oct 20 2006	Nov 9 2014	1,000	10	1.0
Ambari	Hadoop management web UI backed by its RESTful APIs.	Sep 26 2011	Aug 8 2014	1,000	29	3.0
Camel	A rule-based routing and mediation engine.	Jul 8 2007	Sep 18 2013	1,000	32	3.0
Derby	A relational database management system.	Sep 28 2004	Sep 17 2014	1,000	88	9.0

Five machine learning algorithms:
Random Forest, Naive Bayes, Logistic Regression, Multilayer Perceptron and K-Nearest Neighbor.

Performance Measures

		Actual	
		SBRs	NSBRs
Predict	SBRs	TP	FN
	NSBRs	FP	TN
	pd	$\frac{TP}{TP+FN}$	
	pf	$\frac{FP}{FP+TN}$	
	prec	$\frac{TP}{TP+FP}$	
	f-measure	$\frac{2*pd*prec}{pd+prec}$	
	g-measure	$\frac{2\times pd\times \boxed{100-pf}}{pd+(100-pf)}$	
Rank	AP_n	$\sum_{k=1}^{n}\frac{P(k)}{n}$	
	MAP_n	$\sum_{i=1}^{N}\frac{AP_{n_i}}{N}$	

(not predicting NSBRs as SBRs)

probability of detection (pd),probability of false alarm (pf), precision, f-measures and g-measures

EXPERIMENT DESIGN AND RESULTS

Within Project Prediction(WPP) and Transfer ProjectPrediction (TPP)
filters： unfiltered, FARSEC filtered, and CLNI filtered：removes noisy NSBRs

- RQ1: Can security cross words lead to mislabelled security bug reports by prediction models?
- RQ2: How do we build *effective* prediction models for security bug reports when data scarcity is an issue?

RQ1: Can Security Cross Words Lead to Mislabelled Security Bug Reports by Prediction Models

Source	Security Cross Words (SCWs)	
	Filter	# SCWs
Chromium	train	100
	farsecsq	95
	farsectwo	100
	farsec	100
	clni	100
	clnifarsecsq	95
	clnifarsectwo	100
	clnifarsec	100
Wicket	train	74
	farsecsq	12
	farsectwo	13
	farsec	40
	clni	74
	clnifarsecsq	12
	clnifarsectwo	13
	clnifarsec	40
Ambari	train	95
	farsecsq	25
	farsectwo	57
	farsec	88
	clni	94
	clnifarsecsq	25
	clnifarsectwo	57
	clnifarsec	88
Camel	train	88
	farsecsq	27
	farsectwo	47
	farsec	82
	clni	88
	clnifarsecsq	27
	clnifarsectwo	47
	clnifarsec	82
Derby	train	95
	farsecsq	1
	farsectwo	72
	farsec	90
	clni	94
	clnifarsecsq	1
	clnifarsectwo	70
	clnifarsec	89

TABLE 5
WPP Results with FARSEC and CLNI Filtering (Those with the Highest G-Measures Are Highlighted)

Target	Filter	Learner	TN	TP	FN	FP	pd	pf	prec	f-measure	g-measure
Chromium	train	logistic_regression	20,815	18	97	40	15.7	0.2	31.0	20.8	27.1
	farsecsq	random_forest	20,801	17	98	54	14.8	0.3	23.9	18.3	25.7
	farsectwo	logistic_regression	20,815	18	97	40	15.7	0.2	31.0	20.8	27.1
	farsec	logistic_regression	20,815	18	97	40	15.7	0.2	31.0	20.8	27.1
	clni	logistic_regression	20,808	18	97	47	15.7	0.2	27.7	20.0	27.1
	clnifarsecsq	**multilayer_perceptron**	**20,066**	**57**	**58**	**789**	**49.6**	**3.8**	**6.7**	**11.9**	65.4
	clnifarsectwo	logistic_regression	20,808	18	97	47	15.7	0.2	27.7	20.0	27.1
	clnifarsec	logistic_regression	20,808	18	97	47	15.7	0.2	27.7	20.0	27.1
Wicket	train	naive_bayes	459	1	5	35	16.7	7.1	2.8	4.8	28.3
	farsecsq	logistic_regression	305	4	2	189	66.7	38.3	2.1	4.0	64.1
	farsectwo	**logistic_regression**	**313**	**4**	**2**	**181**	**66.7**	**36.6**	**2.2**	**4.2**	65.0
	farsec	logistic_regression	454	2	4	40	33.3	8.1	4.8	8.3	48.9
	clni	naive_bayes	467	0	6	27	0.0	5.5	0.0	0.0	0.0
	clnifarsecsq	logistic_regression	368	2	4	126	33.3	25.5	1.6	3.0	46.1
	clnifarsectwo	logistic_regression	357	2	4	137	33.3	27.7	1.4	2.8	45.6
	clnifarsec	logistic_regression	442	3	3	52	50.0	10.5	5.5	9.8	64.2
Ambari	train	multilayer_perceptron	485	1	6	8	14.3	1.6	11.1	12.5	24.9
	farsecsq	random_forest	422	3	4	71	42.9	14.4	4.1	7.4	57.1
	farsectwo	**random_forest**	**478**	**4**	**3**	**15**	**57.1**	**3.0**	**21.1**	**30.8**	71.9
	farsec	multilayer_perceptron	469	1	6	24	14.3	4.9	4.0	6.3	24.8
	clni	multilayer_perceptron	480	1	6	13	14.3	2.6	7.1	9.5	24.9
	clnifarsecsq	random_forest	455	4	3	38	57.1	7.7	9.5	16.3	70.6
	clnifarsectwo	random_forest	471	2	5	22	28.6	4.5	8.3	12.9	44.0
	clnifarsec	random_forest	493	1	6	0	14.3	0.0	100.0	25.0	25.0
Camel	train	logistic_regression	464	2	16	17	11.1	3.5	10.5	10.8	19.9
	farsecsq	random_forest	426	3	15	55	16.7	11.4	5.2	7.9	28.1
	farsectwo	**logistic_regression**	**280**	**9**	**9**	**201**	**50.0**	**41.8**	**4.3**	**7.9**	53.8
	farsec	logistic_regression	448	3	15	33	16.7	6.9	8.3	11.1	28.3
	clni	naive_bayes	422	3	15	59	16.7	12.3	4.8	7.5	28.0
	clnifarsecsq	multilayer_perceptron	415	3	15	67	16.7	13.9	4.3	6.8	27.9
	clnifarsectwo	multilayer_perceptron	445	2	16	37	11.1	7.7	5.1	7.0	19.8
	clnifarsec	logistic_regression	458	3	15	24	16.7	5.0	11.1	13.3	28.4
Derby	train	naive_bayes	427	16	26	31	38.1	6.8	34.0	36.0	54.1
	farsecsq	ib_k	321	23	19	137	54.8	29.9	14.4	22.8	61.5
	farsectwo	**random_forest**	**401**	**20**	**22**	**57**	**47.6**	**12.4**	**26.0**	**33.6**	61.7
	farsec	naive_bayes	429	16	26	29	38.1	6.3	35.6	36.8	54.2
	clni	random_forest	456	10	32	2	23.8	0.4	83.3	37.0	38.4
	clnifarsecsq	ib_k	321	23	19	137	54.8	29.9	14.4	22.8	61.5
	clnifarsectwo	random_forest	416	15	27	42	35.7	9.2	26.3	30.3	51.3
	clnifarsec	naive_bayes	427	16	26	31	38.1	6.8	34.0	36.0	54.1

RQ2: : How Do We Build Effective Prediction Models for Security Bug Reports When Data Scarcity Is an Issue

TABLE 5
WPP Results with FARSEC and CLNI Filtering (Those with the Highest G-Measures Are Highlighted)

Target	Filter	Learner	TN	TP	FN	FP	pd	pf	prec	f-measure	g-measure
Chromium	train	logistic_regression	20,815	18	97	40	15.7	0.2	31.0	20.8	27.1
	farsecsq	random_forest	20,801	17	98	54	14.8	0.3	23.9	18.3	25.7
	farsectwo	logistic_regression	20,815	18	97	40	15.7	0.2	31.0	20.8	27.1
	farsec	logistic_regression	20,815	18	97	40	15.7	0.2	31.0	20.8	27.1
	clni	logistic_regression	20,808	18	97	47	15.7	0.2	27.7	20.0	27.1
	clnifarsecsq	**multilayer_perceptron**	**20,066**	**57**	**58**	**789**	**49.6**	**3.8**	**6.7**	**11.9**	65.4
	clnifarsectwo	logistic_regression	20,808	18	97	47	15.7	0.2	27.7	20.0	27.1
	clnifarsec	logistic_regression	20,808	18	97	47	15.7	0.2	27.7	20.0	27.1
Wicket	train	naive_bayes	459	1	5	35	16.7	7.1	2.8	4.8	28.3
	farsecsq	logistic_regression	305	4	2	189	66.7	38.3	2.1	4.0	64.1
	farsectwo	**logistic_regression**	**313**	**4**	**2**	**181**	**66.7**	**36.6**	**2.2**	**4.2**	65.0
	farsec	logistic_regression	454	2	4	40	33.3	8.1	4.8	8.3	48.9
	clni	naive_bayes	467	0	6	27	0.0	5.5	0.0	0.0	0.0
	clnifarsecsq	logistic_regression	368	2	4	126	33.3	25.5	1.6	3.0	46.1
	clnifarsectwo	logistic_regression	357	2	4	137	33.3	27.7	1.4	2.8	45.6
	clnifarsec	logistic_regression	442	3	3	52	50.0	10.5	5.5	9.8	64.2
Ambari	train	multilayer_perceptron	485	1	6	8	14.3	1.6	11.1	12.5	24.9
	farsecsq	random_forest	422	3	4	71	42.9	14.4	4.1	7.4	57.1
	farsectwo	**random_forest**	**478**	**4**	**3**	**15**	**57.1**	**3.0**	**21.1**	**30.8**	71.9
	farsec	multilayer_perceptron	469	1	6	24	14.3	4.9	4.0	6.3	24.8
	clni	multilayer_perceptron	480	1	6	13	14.3	2.6	7.1	9.5	24.9
	clnifarsecsq	random_forest	455	4	3	38	57.1	7.7	9.5	16.3	70.6
	clnifarsectwo	random_forest	471	2	5	22	28.6	4.5	8.3	12.9	44.0
	clnifarsec	random_forest	493	1	6	0	14.3	0.0	100.0	25.0	25.0

TABLE 6
TPP Results with FARSEC and CLNI Filtering (Those with the Highest G-Measures Are Highlighted)

Target	Source	Filter	Learner	TN	TP	FN	FP	pd	pf	prec	f-measure	g-measure
Chromium	Derby	train	random_forest	20,835	2	113	20	1.7	0.1	9.1	2.9	3.4
	Ambari	farsecsq	random_forest	19,279	34	81	1,576	29.6	7.6	2.1	3.9	44.8
	Ambari	farsectwo	random_forest	20,454	53	62	401	46.1	1.9	11.7	18.6	62.7
	Derby	farsec	multilayer_perceptron	20,502	12	103	353	10.4	1.7	3.3	5.0	18.9
	Camel	clni	logistic_regression	20,262	25	90	593	21.7	2.8	4.0	6.8	35.5
	Ambari	**clnifarsecsq**	**random_forest**	**19,817**	**56**	**59**	**1,038**	**48.7**	**5.0**	**5.1**	**9.3**	64.4
	Derby	clnifarsectwo	random_forest	20,332	26	89	523	22.6	2.5	4.7	7.8	36.7
	Camel	clnifarsec	multilayer_perceptron	20,590	8	107	265	7.0	1.3	2.9	4.1	13.0
Wicket	**Camel**	**train**	**naive_bayes**	**437**	**3**	**3**	**57**	**50.0**	**11.5**	**5.0**	**9.1**	63.9
	Chromium	farsecsq	multilayer_perceptron	475	1	5	19	16.7	3.8	5.0	7.7	28.4
	Camel	farsectwo	random_forest	490	1	5	4	16.7	0.8	20.0	18.2	28.5
	Camel	farsec	naive_bayes	431	3	3	63	50.0	12.8	4.5	8.3	63.6
	Ambari	clni	multilayer_perceptron	476	1	5	18	16.7	3.6	5.3	8.0	28.4
	Chromium	clnifarsecsq	random_forest	493	1	5	1	16.7	0.2	50.0	25.0	28.6
	Camel	clnifarsectwo	random_forest	489	1	5	5	16.7	1.0	16.7	16.7	28.5
	Camel	clnifarsec	naive_bayes	433	3	3	61	50.0	12.3	4.7	8.6	63.7
Ambari	Derby	train	multilayer_perceptron	484	2	5	9	28.6	1.8	18.2	22.2	44.3
	Chromium	**farsecsq**	**multilayer_perceptron**	**474**	**3**	**4**	**19**	**42.9**	**3.9**	**13.6**	**20.7**	59.3
	Chromium	farsectwo	naive_bayes	472	3	4	21	42.9	4.3	12.5	19.4	59.2
	Camel	farsec	multilayer_perceptron	492	1	6	1	14.3	0.2	50.0	22.2	25.0
	Derby	clni	multilayer_perceptron	477	2	5	16	28.6	3.2	11.1	16.0	44.1
	Chromium	clnifarsecsq	random_forest	492	1	6	1	14.3	0.2	50.0	22.2	25.0
	Chromium	clnifarsectwo	naive_bayes	474	2	5	19	28.6	3.9	9.5	14.3	44.1
	Camel	clnifarsec	multilayer_perceptron	492	1	6	1	14.3	0.2	50.0	22.2	25.0

谢谢观看

熊 猫 达 人 原 创 精 品