PUMiner: Mining Security Posts from Developer Question and Answer Websites with PU Learning

Triet Huynh Minh Le[1], David Hin[1,2] Roland Croft[1], and M. Ali Babar[1,2]
[1]School of Computer Science, The University of Adelaide, Adelaide, Australia
[2]Cyber Security Cooperative Research Centre
triet.h.le@adelaide.edu.au, david.hin@student.adelaide.edu.au,
roland.croft@student.adelaide.edu.au, ali.babar@adelaide.edu.au

ABSTRACT

Security is an increasing concern in software development. Developer Question and Answer (Q&A) websites provide a large amount of security discussion. Existing studies have used human-defined rules to mine security discussions, but these works still miss many posts, which may lead to an incomplete analysis of the security practices reported on Q&A websites. Traditional supervised Machine Learning methods can automate the mining process; however, the required negative (non-security) class is too expensive to obtain. We propose a novel learning framework, PUMiner, to automatically mine security posts from Q&A websites. PUMiner builds a context-aware embedding model to extract features of the posts, and then develops a two-stage PU model to identify security content using the labelled Positive and Unlabelled posts. We evaluate PUMiner on more than 17.2 million posts on Stack Overflow and 52,611 posts on Security StackExchange. We show that PUMiner is effective with the validation performance of at least 0.85 across all model configurations. Moreover, Matthews Correlation Coefficient (MCC) of PUMiner is 0.906, 0.534 and 0.084 points higher than one-class SVM, positive-similarity filtering, and one-stage PU models on unseen testing posts, respectively. PUMiner also performs well with an MCC of 0.745 for scenarios where string matching totally fails. Even when the ratio of the labelled positive posts to the unlabelled ones is only 1:100, PUMiner still achieves a strong MCC of 0.65, which is 160% better than fully-supervised learning. Using PUMiner, we provide the largest and up-to-date security content on Q&A websites for practitioners and researchers.

KEYWORDS

Mining Software Repositories, Positive Unlabelled Learning, Machine Learning, Natural Language Processing, Software Security

MSR '20, October 5–6, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7517-7/20/05…$15.00
https://doi.org/10.1145/3379597.3387443

ACM Reference Format:

Triet Huynh Minh Le, David Hin, Roland Croft and M. Ali Babar. 2020. PUMiner: Mining Security Posts from Developer Question and Answer Websites with PU Learning. In *Proceedings of Mining Software Repositories (MSR'20), October 5-6, 2020, Seoul, Republic of Korea*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3379597.3387443

1 INTRODUCTION

Security incidents and their damaging effects are increasing at an unprecedented rate [1]. There are several platforms reporting security vulnerabilities and attacks such as Common Vulnerabilities and Exposures (CVE) [2], National Vulnerability Database (NVD) [3], Common Weakness Enumeration (CWE) [4], Common Attack Pattern Enumeration and Classification (CAPEC) [5], and Open Web Application Security Project (OWASP) [6]. These platforms provide definitions and examples of known security attacks and countermeasures. However, they do not support on-the-fly discussion about the up-to-date usage and implementation of such security in real-life scenarios. For example, CWE suggests using the *mysql_real_escape_string()* function to mitigate SQL injection vulnerability in PHP [7], but this function was deprecated and one should use PDO or MySQLi instead for better protection [8]. Conversely, developer Question and Answer (Q&A) websites provide an abundance of such discussions [9]. Until 2019, Stack Overflow [10], the largest developer Q&A site, has had around seventeen million posts and more than ten million users. This is an enormous data source for mining patterns in secure software development.

Several studies [11-14] have proposed different heuristics to retrieve security content from open sources. Such predefined rules require considerable domain expertise and human effort. More importantly, the rules are not exhaustive since security concepts keep evolving over time [15]. There is a need for an automatic and accurate way of retrieving security content. It is observed that recent unsupervised word/document embeddings [16-19] in Natural Language Processing (NLP) have achieved state-of-the-art results for various textual information retrieval tasks. Such NLP methods can then be combined with a Machine Learning (ML) classifier to automatically identify security-related posts on developer Q&A websites without human-defined rules.

Building a traditional binary classification model requires a large amount of labelled data containing both positive (security) and negative (non-security) classes [20, 21]. However, it is very challenging to obtain such high-quality and labelled datasets. Even

with the knowledge from multiple security sources (e.g., NVD, CVE, CWE, CAPEC, and OWASP), we are still unsure of the labels of more than 98% of the posts, leaving them unlabelled. Human inspection also does not scale to such a large extent. Fortunately, there are still related yet much smaller sources that are likely to be related to the domain of interest for learning the patterns of the positive class. In the context of security, Security StackExchange [22] contains mostly security-related discussion, and this site also follows the same format as Stack Overflow. Such observations have motivated us to formulate and address a research problem "**How to retrieve security-related content from developer Q&A websites using only a small quantity of labelled positive posts and a large number of unlabelled posts?**"

We present **PU**Miner, a novel learning framework to mine the posts belonging to a topic of interest from developer Q&A websites using only **P**ositive and **U**nlabelled posts. Specifically, we represent the posts with a context-aware feature model and then build a two-stage PU learning model to discern security-related posts on Q&A websites. Such security posts help researchers and practitioners study security practices/guidelines in a software development lifecycle as well as improve the discovery and mitigation of software vulnerabilities. Our key contributions are:

- We are the first to formulate the retrieval of security-related posts as a PU learning problem.
- We propose a novel framework, PUMiner, to retrieve security posts using unlabelled data.
- We systematically evaluate the performance of PUMiner using more than 17.3 million posts on both Stack Overflow and Security StackExchange.
- We demonstrate that PUMiner outperformed one-class SVM [23], positive-similarity filtering [24], one-stage PU models. PUMiner even detected posts that keyword matching totally failed. PUMiner was also better than a fully supervised approach treating all unlabelled data as negative class even with only 1% positive data.
- We provide the largest security-related datasets extracted from Q&A websites and code at [25].

2 BACKGROUND AND MOTIVATIONS

2.1 Security Discussion on Q&A Websites

A discussion thread on Stack Overflow developer Question and Answer (Q&A) website is called a post [26]. A post as illustrated in Figure 1 contains a unique id, owner, title, question, tag(s) and answer(s) along with additional information such as comments, the dates when it was created, edited, last-active and closed, the score (upvotes minus downvotes), the number of views and favorites. A complete schema of a post on Q&A websites can be found at [27]. Security StackExchange, a security Q&A website, also follows a similar format as Stack Overflow.

To identify security-related posts, we consider the title, tags, question and answer bodies following the previous practices [11, 12, 26]. The security relevance of a post is then based on the explicit or implicit reference or discussion about cybersecurity and/or any related subtopics mentioned in every part of a post.

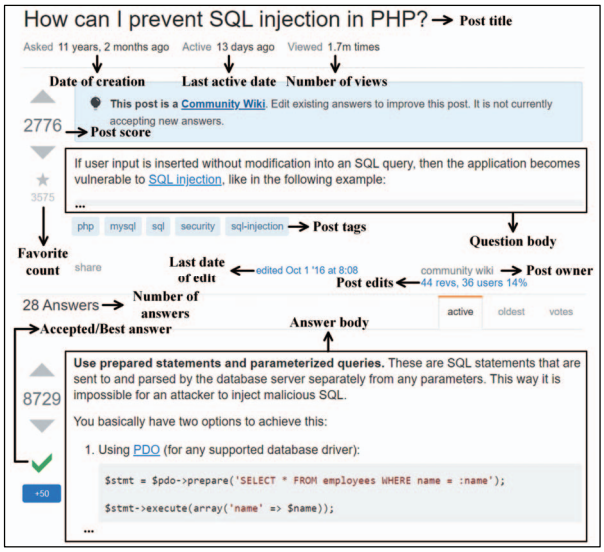

Figure 1: Format of a security post 60174 on Stack Overflow

A post that asks about how to use and/or develop security tools, functions and/or frameworks (e.g., login/authentication) is still related to security. However, discussing a non-security task (e.g., post id 35275411 on Stack Overflow about integrating third-party APIs) that may indirectly involve security requirements (e.g., user's authentication/permission) is not security-relevant.

In the literature, the number of security posts identified on Stack Overflow is still very limited. From our manual inspection of 385 random posts (i.e., 95% confidence level and 5% error [28]), we found sixteen (around 4%) posts were security-related. However, the largest set [12] available in the literature just contains around thirty thousand posts (only about 0.1% of 17.2 million Stack Overflow posts). This set of posts was created with a keyword-matching approach using only the tags (i.e., *encryption*, *cryptography*, *passwords*, *security*, *sql-injection*, *web-security*, *xss*) of a post. There are many cases in which a security post does not have any of these seven tags. One such example is the post id 3226374 on Stack Overflow, discussing a cross-site scripting issue in PHP using only the tag *php*. More limitations of keyword matching are discussed in the next section.

2.2 Keyword Matching and its Limitations

A straightforward way to determine the type of a post is by matching it with predefined keywords in the domain of interest (i.e., security). However, there are two major drawbacks of such a keyword-matching approach. Firstly, it requires considerable domain expertise and a tedious trial-and-error process to select appropriate keywords. For example, to the best of our knowledge, the most extensive list has 127 security keywords proposed by Pletea et al. [14]. However, based on our expertise, we have found that such list is still missing many important security keywords. With further investigation, we found out that missing the keyword "*ssh*" (a cryptographic network protocol) alone would result in at least 32,216 potential posts being overlooked.

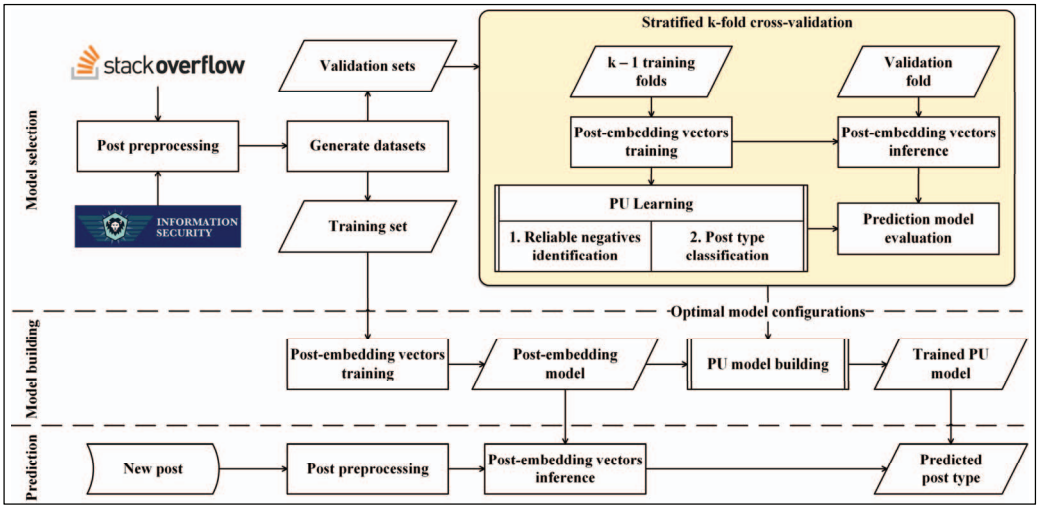

Figure 2: Workflow of our proposed PUMiner framework for mining security-related posts from developer Q&A websites

Another important challenge of keyword matching is to balance between Precision and Recall of information retrieval. With fewer keywords, it is likely to miss many posts of interest (i.e., low recall) as demonstrated above. On the contrary, even with more relevant keywords, it does not guarantee a better matching accuracy. In fact, more keywords may result in a higher false-positive rate due to the multiple meanings of a word. For example, the word "*exploit*" in the previous security list [14] also means "*take advantage of*" some techniques/tools in general, which is not necessarily related to a security attack. Another example is the word "*signed*", which is also a matching subword of non-security words like "*assigned*". Therefore, a more reliable and generalized technique is required for addressing such drawbacks.

2.3 The need of PU Learning for Retrieving Security Posts

Machine Learning (ML)-based binary classifiers can automatically identify important keywords to determine a type of interest. To build a binary classifier, we need to have both positive (i.e., security-related) and negative (i.e., non-security related) posts. Unfortunately, there has not been such a large and reliable dataset containing both positive and negative classes in the literature. We have only identified a small number of positive posts with high confidence from Stack Overflow and Security StackExchange. Retrieving non-security posts is also challenging since it should not contain any security context, which requires significant human effort to define and verify. Hence, there are many posts still left unlabelled (i.e., containing a mixture of positive and negative posts). This would violate the prerequisite condition of developing a traditional binary classifier since the negative class is unknown [21, 29]. Nonetheless, this is a suitable case for **PU** learning to utilize both available **P**ositive and **U**nlabelled samples [30]. Such observations have motivated us to formulate the identification of security posts on Q&A websites as a PU binary learning problem.

Input representation is important for building an ML model. In NLP, a textual post on a Q&A website can be represented using Bag-of-Words (BoW), n-grams, or tf-idf. However, these traditional techniques only support the exact matching of each word without considering its semantic meaning [31, 32]. For instance, "*penetration-testing*" or "*pen-test*" is usually used for vulnerability assessment (i.e., security-related context), but BoW, n-grams or tf-idf would treat "*pen-test*" and "*security*" as two totally different words. To address such drawback, word/document embeddings [16, 19] are utilized to incorporate the context (neighboring words) of a considering word. Therefore, our work would use context-aware word/document embeddings to represent the content of a discussion post to determine its security relevance.

3 THE PUMINER FRAMEWORK

3.1 Approach Overview

We propose **PU**Miner as illustrated in Figure 2 to automatically perform large-scale mining of security content from developer Q&A websites using **P**ostive and **U**nlabelled posts. PUMiner framework contains: (*i*) model selection, (*ii*) model building, and (*iii*) prediction. The model selection and building processes develop the optimal PU learning models, and the prediction process uses such models to predict the security relevance for a new post. It is noted that the workflow of PUMiner can be customized to retrieve other topics of interest from developer Q&A websites.

Model selection. There are three steps: (*i*) preprocess posts from Q&A websites, (*ii*) generate datasets for validation and training, and (*iii*) perform stratified k-fold cross-validation to select the best hyperparameters for our PU models. Step (*i*) accepts and preprocesses raw posts from Stack Overflow and Security StackExchange, see section 3.2 for more preprocessing details. Step (*ii*) takes the preprocessed posts to create a reliable positive (security) dataset for PU learning. Although PU learning does not

require the negative class, it still needs to learn the patterns from the positive class. We propose different heuristics in section 3.3 to obtain a positive dataset using both the tags and content of a post. Such heuristics have been defined based on our expertise and references from CVE, NVD, CWE, CAPEC and OWASP. It should be noted that these heuristics are insufficient for retrieving security posts at full scale (see section 2.2). Besides the positive set, step (*ii*) also obtains an unlabelled set to build a complete dataset for validating/training PU models. The validation set is separated into *k* folds, each containing both security and unlabelled posts.

The folds enter the last step (*iii*) to perform stratified k-fold cross-validation. Stratification ensures that the ratio of each input source is kept throughout the cross-validation step, avoiding different data distribution of the folds. In each iteration, $k - 1$ folds are used for training a model, while the remaining one is used for validating such model. In the training step of an iteration, textual posts and their tags are first fed into a doc2vec [19] embedding model to obtain fixed-length context-aware feature vectors. The embedding model is also stored temporarily for inferring the vectors of the posts in both the training and validation folds. The trained feature vectors of the training folds then enter a two-stage PU learning model (see section 3.4). The first step of PU learning identifies the reliable negative (non-security) posts in the training folds to approximately transform this PU learning problem into a fully-supervised learning one. After that, both positive (security) and reliable negative (non-security) posts in the training folds are used to train a binary classifier to predict the type of each post. Such trained model is then evaluated on the validation fold. This whole training-validation process is repeated *k* times. The validation model performance is the average value of *k* runs. The optimal model configurations with the highest performance metric would be selected for the next model building process.

Model building. This process has two steps: (*i*) train a doc2vec feature model to obtain feature vectors, and (*ii*) train a PU learning model to predict security-related posts with the optimal configurations from the previous process. In step (*i*), doc2vec embedding is trained on the whole dataset generated in the model selection process to obtain the context-aware feature vectors of each word and post. This feature model is also saved to disk for looking up the vectors of future posts. Using the obtained features, step (*ii*) trains a two-stage PU learning model with the optimal configurations of the model selection process. Lastly, such prediction model is then saved to disk for future inference.

Prediction. This process first reuses the same preprocessing module to clean a new post. The cleaned content is then converted into context-aware vectors by the feature model developed in the model building process. The trained PU learning model uses such features to determine whether the current post is related to security. We do not use tags for prediction since they may not capture the true topic of a post (e.g., the post 3226374 in section 2).

3.2 Preprocessing of Discussion Posts

Discussion posts on Q&A websites contain too much noise for a learning model to be applied directly. The main problem with raw posts is the large vocabulary of the post content, which can make a learning model overfit [33]. Following the practices of the previous works [12, 13, 26], we first remove code snippets and outputs (within <pre><code> and </pre></code> tags) and HTML tags (e.g., and). Then, we remove stop words as well as punctuations. We leave security and software engineering keywords intact, e.g., "*sql-injection*", "*private-key*", or "*x.509*". Moreover, we convert text to lowercase and perform Porter stemming [34] to remove duplicate forms of a word (e.g., "*attack*" vs. "*attacks*" vs. "*Attacker*"). We apply the same process for the title, question, tags, and answers of a post.

3.3 Heuristics to Build a Security-related Dataset

Heuristics may miss many security posts (see section 2.2), but they are still useful for gathering a decent positive dataset for PU learning. In this work, we propose to build a security-related dataset using both **tag-based** and **content-based filtering**. Regarding the **tag-based filtering**, inspired by Yang et al. [12], we select the tags whose frequency in the security context ($|tag_i \cap security| / |tag_i|$, $|.|$ is the tag count) and popularity ($|tag_i \cap security| / |security|$) are larger than *Thre1* and *Thre2* thresholds, respectively. We consider all tags with the subword "*security*" (e.g., *spring-security*, *firebase-security* or *android-security*) along with the *security* tag to select co-occurrent tags. We have adopted *Thre1* = 0.1 and *Thre2* = 0.01 as in [12] and obtained five tags: *cryptography*, *csrf*, *passwords*, *sql-injection* and *xss*. We then retrieve all the posts with at least one of such tags to create the security set$_{tag}$. However, there are still security posts with no security tag (e.g., post 3226374 in section 2), requiring the examination of the content of other parts besides tags as well.

For the **content-based filtering** on the remaining posts, we present an up-to-date list of 234 security keywords (see APPENDIX), which is two times larger than [14]. Such keywords are inherited from the existing list and gathered from CVE, NVD, CWE, CAPEC and OWASP. We consider different matching forms (i.e., American/British English and with/without hyphen/space/stemming) to cope with various (mis-)spellings. For instance, "*improper synchronization*" includes: improper/improp(-)synchronisation/synchronization/synchron/synchronis. Like [14], to reduce false positives, we perform exact matching for short (three-character) keywords (e.g., *md5*) and subword matching otherwise. For each post, we then extract the ratio of security words (*kw_ratio*), and how many security keywords appear in total (*kw_count*). *kw_ratio* ensures that a security word is not an outlier in the discussion, e.g., the post 477816 on Stack Overflow has the word "*security*" in the question, but it is mainly about JSON MIME type. A keyword may also have several meanings, some of which are unrelated to security. For example, "*hash*" may refer to a data structure instead of cryptography. *kw_count* increases the coverage of security discussion in a post. We have obtained the thresholds $a = 0.053$ and $b = 8$ for *kw_ratio* and *kw_count*, respectively, from unclosed Security StackExchange posts. Such posts help to ensure the security relevance. The content-based filtering then selects Stack Overflow posts whose $kw\_ratio \geq a$ and $kw\_count \geq b$ to obtain set$_{content}$ of security posts.

We combine set_{tag} and $set_{content}$ to form our security dataset from Stack Overflow. We have randomly sampled 385 posts (statistically significant size [28]) from each set_{tag} and $set_{content}$ for two authors to independently examine the security relevance. The disagreement proportions were only 1.6% and 0.3% for set_{tag} and $set_{content}$, respectively. When there was a conflict, the third author would be involved in the decision-making process. For set_{tag}, we encountered only two unclear security cases (< 1%), i.e., the posts 46178245 and 10727000 did not explicitly mention the security purposes for sanitizing input. For $set_{content}$, all 385 posts checked were security-related. To extend PUMiner to other domains, relevant anchor tags of tag-based filtering and keywords of content-based filtering need to be updated accordingly.

3.4 Context-aware Two-stage PU Learning Model

Traditional feature extraction methods like BoW, n-grams, tf-idf do not consider the semantic relationship of a word [35, 36]. In contrast, word embeddings such as word2vec [16] incorporate the context of a word by maximizing the following likelihood: $\prod_{t}^{N} \prod_{-ws \le i \le ws,\ i \ne t} P(w_{t+i} \mid w_t; \theta)$. Specifically, word2vec optimizes its parameters (θ) to maximize the probability of the context words appearing within a window size (*ws*) of the current word (w_t). Doc2vec [19] extends word2vec to jointly learn the representation of a document (paragraph vector) with its constituent words. Our work adopts the distributed memory architecture to train doc2vec model as suggested in [19]. Regarding the label of a document/post, a post may contain multiple tags to describe complex concepts, e.g., *php* and *security* tags represent security issues (sql-injection) in PHP. Therefore, besides single tags, we also combine all tags to handle the topic mixture. For example, a post with *php* and *security* tags would have *php_security* alongside *php*, *security* and *post id* as labels. We also sort labels alphabetically to avoid duplicates (e.g., *php_security* and *security_php* are the same).

Next, we present the context-aware two-stage PU learning model (see Algorithm 1) to retrieve security-related post – the core of our PUMiner framework. This algorithm requires a list of discussion posts with their respective labels (positive or unlabelled), along with the configurations of doc2vec (size of embeddings and window size) and classification models (model hyperparameters). Details of Algorithm 1 are given hereafter.

Lines 1-8: **Learning doc2vec context-aware feature vectors.** Lines 1-4 tokenize text and extract tags of each post to prepare the data for training doc2vec models. Line 5 trains a doc2vec embedding model with stochastic gradient descent to learn the context of words and posts. Line 6 then obtains the embedding vector of each post using the trained doc2vec model. Lines 7 and 8 extract the features of both positive and unlabelled posts, respectively.

Lines 9-13: Stage one of PU learning model. Inspired by PU learning in other domains [37-40], we assume that the context-aware doc2vec embeddings can make posts of the same class stay in close proximity in the embedding space. Such assumption has been shown to hold in section 5.1.

Algorithm 1. Context-aware two-stage PU learning model building.

Input: List of posts: P_{in}

Labels of posts: $labels \in \{positive, unlabelled\}$

Size of embeddings and Window size: *sz*, *ws*

Classifier and its model configurations: *C*, *config*

Output: The trained feature and PU models: feature_model, $model_{PU}$

1 $word\_list, tag\_list \leftarrow$ empty list, empty list
2 **foreach** $p_i \in P_{in}$ **do**
3 $words, tags \leftarrow$ tokenize (p_i) , extract_tags (p_i)
4 $word\_list, tag\_list \leftarrow word\_list + \{words\}, tag\_list + \{tags\}$
5 feature_model $\leftarrow$ train_doc2vec(*word_list*, *tag_list*, *sz*, *ws*)
6 $\mathbf{X}_{in} \leftarrow$ obtain_feature(feature_model, *word_list*)
7 $P \leftarrow \{\mathbf{x}_p \mid \mathbf{x}_p \in \mathbf{X}_{in} \wedge label(p) = positive\}$
8 $U \leftarrow \{\mathbf{x}_p \mid \mathbf{x}_p \in \mathbf{X}_{in} \wedge label(p) = unlabelled\}$
9 $\mathbf{centroid}_P$, $\mathbf{centroid}_U \leftarrow \frac{\sum_{p \in P} \mathbf{x}_p}{|P|}$, $\frac{\sum_{p \in U} \mathbf{x}_p}{|U|}$
10 $RN \leftarrow$ empty list
11 **foreach** $\mathbf{x}_i \in \mathbf{X}_{in}$ **do** // Stage-1 PU: Identify reliable negatives
12 **if** $d(\mathbf{x}_i, \mathbf{centroid}_U) < \alpha * d(\mathbf{x}_i, \mathbf{centroid}_P)$ **do**
13 $RN \leftarrow RN + \{\mathbf{x}_i\}$
14 $model_{PU} \leftarrow$ train_classifier(*C*, *P*, *RN*, *config*) // Stage-2 PU
15 **return** feature_model, $model_{PU}$

Stage-one PU learning first identifies (reliable/pure) negative (non-security) posts in the unlabelled set that are as different as possible from the positive set. A traditional binary classifier would work poorly in this case since the negative class is not pure [39]. This has been confirmed in section 5.3. In line 10 of Algorithm 1, we propose to approximately locate unknown negative posts using the centroid (average) vectors of the known positive ($\mathbf{centroid}_P$) and unlabelled ($\mathbf{centroid}_U$) sets, respectively. Since the number of non-security posts is dominant (i.e., up to 96% on Stack Overflow as in section 2), $\mathbf{centroid}_U$ would represent the negative class more than the positive one. Lines 11-13 compute and compare the cosine distances [16, 19, 39] (see Eq. (1)) from each post to the two centroids. If the current post is closer to $\mathbf{centroid}_U$ (i.e., more towards the negative class), it would be selected as a reliable negative.

$$\text{cosine\_distance} = d(i,j) = 1 - \frac{\mathbf{p}_i \bullet \mathbf{p}_j}{\|\mathbf{p}_i\| \times \|\mathbf{p}_j\|} \quad (1)$$

where $\mathbf{p}_i$ and $\mathbf{p}_j$ are the embedding vectors of the posts i^{th} and j^{th}, respectively. The range of cosine_distance is [0, 2].

We also propose a scaling factor (α) to increase the flexibility of our centroid-based approach, which would be jointly optimized with other hyperparameters of binary classifiers in the second stage. Besides having only one hyperparameter for tuning, this centroid-based approach can incrementally learn new post(s) very fast with $O(1)$ complexity as given in Eq. (2).

$$\mathbf{centroid}_{new} = \frac{\mathbf{centroid}_{old} * N + \mathbf{x}_{new}}{N + \text{size}(\mathbf{x}_{new})} \quad (2)$$

where $\mathbf{centroid}_{old}$, $\mathbf{centroid}_{new}$ are the centroid vectors before and after learning new post(s) ($\mathbf{x}_{new}$), while N is the original number of posts in the positive or unlabelled set.

Lines 14-15: Stage two of PU learning model. Using positive and (reliable) negative posts from the first stage, the second stage of PU learning (line 14) trains a binary classifier with its hyperparameters. In the model building process, PU model is trained with the optimal classifier and its configurations obtained from stratified k-fold cross-validation (see Figure 2). Finally, line 15 saves the trained feature and PU models to disk for future inference.

4 EXPERIMENTAL DESIGN AND SETUP

4.1 Dataset

We gathered 17,278,709 posts on Stack Overflow (SO) as of July 2019 from BigQuery [41], and 52,611 posts on Security StackExchange (SSE) as of October 2019 from StackExchange Data Explorer [42]. Table 1 summarizes the number of tags and the word length of each part of posts from the two Q&A sites. The data distributions of SSE and SO nearly resemble, except answers. We obtained 102,046 and 27,424 security posts for set_{tag} and $set_{content}$ (see section 3.3), respectively. Set_{tag} was three times larger than the largest (30,054) previous set [12], and $set_{content}$ also had nearly the same size as [12], proving that our new heuristics and keywords are useful. We then randomly sampled 181,696 unlabelled posts from the remaining ones to build the *development set* along with the security posts from set_{tag}, $set_{content}$ and SSE (52,226 posts). 385 random SSE posts with $kw\_count = 0$ were put aside for a separate testing set, which will be explained later in this section. The *development set* had a 1:1 ratio between security and unlabelled posts. 90% of the *development set* was then randomly selected for model selection/building, and the other 10% ($Test_{dev}$) was for model prediction (see Figure 2). Stratified sampling ensures that the proportion of each source would be kept.

We used the whole *development set* to test our models on two more sets ($Test_{norm}$ and $Test_{hard}$), each containing 385 (95% confidence level and 5% error [28]) manually-selected posts for each of the classes (security and non-security). The manual selection was conducted by one author and then carefully verified by two other authors. If no agreement could be reached, another post would be selected for examination until the size of each testing set was met. It is noted that the posts in $Test_{norm}$ were not in the *development set*. $Test_{norm}$ was necessary since it had manually-checked label for each post, allowing an evaluation using both positive and negative classes. $Test_{hard}$ contained 385 security-related posts with $kw\_count = 0$ on SSE (see section 3.3) and 385 non-security posts with $kw\_count > 0$ on SO that were not in the *development set*, set_{tag}, $set_{content}$ or $Test_{norm}$. $Test_{hard}$ had labelled and extreme cases where our tag-based and content-based keyword heuristics in section 3.3 totally failed. One such borderline security post is a concern about an app tracking user's location on smartphone, which affects user's privacy[1]. We have released all datasets at [25].

[1] https://security.stackexchange.com/questions/190107/

Table 1: Statistics of posts on SO and SSE

Source (no. of posts)	Statistic	Tag	Title	Question	Answer
Stack Overflow (17,278,709)	Average	3.0	8.6	159.6	140.8
	Median	3.0	8.0	119.0	87.0
	Min	1.0	1.0	1.0	0.0
	Max	6.0	45.0	11,025.0	45,004.0
Security StackExchange (52,611)	Average	2.6	9.2	140.0	341.4
	Median	3.0	9.0	106.0	225.0
	Min	1.0	1.0	1.0	0.0
	Max	5.0	30.0	3,714.0	9,069.0

4.2 Implementation of the PUMiner Framework

We implemented our heuristics and PU learning models using *scikit-learn* [43], *nltk* [44] and *Gensim* [45] libraries in Python. The hyperparameters we considered for optimizing each model are given in Table 2. For doc2vec, we just used the default configurations since they do not make much of a difference as found in the previous studies [15, 35]. In addition, the six classifiers were selected based on the common practices in the literature [15, 46, 47] and in real data science competitions (e.g., Kaggle [48]). The first three (LR, SVM and KNN) are single models, while the other three (RF, XGB and LGBM) are ensemble ones. There were 260 model configurations for tuning in total. To select the optimal configurations for each model, we performed stratified 10-fold cross-validation. Due to the large scale of the dataset, we ran each fold as a computing job on a supercomputing cluster with at least ten cores and 400 MB of RAM. After that, we aggregated the results from multiple jobs to compute the performance of each cross-validation run. Such performance metrics are described in the next section.

Table 2: Hyperparameter tuning for PUMiner

Modeling step	Model	Hyperparameters
Context-aware feature embedding	Doc2vec [19]	Embedding size: 300 Window size: 5
Identification of reliable negatives	Centroid-based approach (section 3.4)	Alpha (α): 0.8, 0.9, 1.0, 1.1, 1.2
Prediction of the security relevance of a post	Logistic Regression (LR) [49]	Regularization coefficient: 0.01, 0.1, 1, 10, 100
	Support Vector Machine (SVM) [50]	
	K-Nearest Neighbors (KNN) [51]	Number of neighbors: 11, 31, 51 Distance weight: uniform, distance-based Distance norm: 1, 2
	Random Forest (RF) [52]	Number of estimators: 100, 300, 500 Max depth: unlimited Maximum number of leaf nodes: 100, 200, 300, unlimited (RF)
	Extreme Gradient Boosting (XGB) [53]	
	Light Gradient Boosting Machine (LGBM) [54]	

4.3 Evaluation Metrics

Table 3: Evaluation metrics for models of PUMiner

Metric and Formula	Description
1. $\text{Precision}_{PU-LB} = \frac{TP_P}{TP_P + TP_U + FP}$, TP_P & TP_U : $\lvert\hat{y}=1\rvert$ in the positive and unlabelled sets, $\hat{y}$: predicted label, y : true label.	Minimum Precision when only *TP* in the positive set is counted. *TP* in *U* is unknown, and thus not counted.
2. $\text{Precision}_{PU-UB} = \frac{TP_P + \min(r \times \lvert U\rvert, TP_U + FP)}{TP_P + TP_U + FP}$, r: the ratio of security posts in U, $\lvert U\rvert$: size of U.	Maximum Precision when a model is assumed to obtain all security posts in *U*.
3. $\text{Recall}_{PU} = \frac{TP_P}{TP_P + FN_P}$ ($\text{Recall}_{PU} \approx \text{Recall}_{PN}$)	Approximated Recall for PU learning [55]
4. $\text{F1-Score}_{PU-LB} = \frac{2 \times \text{Precision}_{PU-LB} \times \text{Recall}_{PU}}{\text{Precision}_{PU-LB} + \text{Recall}_{PU}}$	Minimum F1-Score for PU learning
5. $\text{F1-Score}_{PU-UB} = \frac{2 \times \text{Precision}_{PU-UB} \times \text{Recall}_{PU}}{\text{Precision}_{PU-UB} + \text{Recall}_{PU}}$	Maximum F1-Score for PU learning
6. $\text{G-mean}_{PU} = \frac{\lvert\hat{y}\rvert \times \text{R}^2}{\lvert\hat{y}=1\rvert}$	G-mean for PU Learning (Eq. (3))
7. $\text{Precision}_{PN} = \frac{TP}{TP + FP}$	Precision on ground-truth posts
8. $\text{Recall}_{PN} = \frac{TP}{TP + FN}$	Recall on ground-truth posts
9. $\text{F1-Score}_{PN} = \frac{2 \times \text{Precision}_{PN} \times \text{Recall}_{PN}}{\text{Precision}_{PN} + \text{Recall}_{PN}}$	F1-Score on ground-truth posts
10. $\text{G-mean}_{PN} = \sqrt{\text{Precision}_{PN} \times \text{Recall}_{PN}}$	G-mean on ground-truth posts
11. $\text{MCC}_{PN} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$	Matthews correlation coefficient on ground-truth posts

Retrieving security posts is a binary classification problem, which can be evaluated using: True Positive (*TP*), False Positive (*FP*), True Negative (*TN*), and False Negative (*FN*) [20]. There are also unlabelled posts; thus, we present evaluation metrics for two different scenarios in Table 3: the *development set* (only Positive and Unlabelled (PU) posts are available), and the testing sets (Positive and Negative (PN) ground truths are known). We estimated r = 0.025 in $\text{Precision}_{PU\text{-}UB}$ by manually examining 385 posts in the unlabelled set. Eq. (3) correlates G-mean_{PN} and G-mean_{PU}.

$$\text{G-mean}_{PN} = \sqrt{\text{P} \times \text{R}} \propto \frac{\text{P} \times \text{R}}{P(y=1)} = \frac{\text{P} \times \text{R}^2}{\text{R} \times P(y=1)}$$
$$= \frac{P(y=1 \mid \hat{y}=1) \times \text{R}^2}{P(\hat{y}=1 \mid y=1) \times P(y=1)} \text{ (Definition of P and R)} \quad (3)$$
$$= \frac{\text{R}^2}{P(\hat{y}=1)} = \frac{\lvert\hat{y}\rvert \times \text{R}^2}{\lvert\hat{y}=1\rvert} = \text{G-mean}_{PU} \text{ (Bayes' rules, } \lvert.\rvert\text{:size)}$$

where P and R are Precision and Recall, respectively.

G-mean is directly proportional to Precision and Recall, similar to F1-Score. Also, *P(y=1)* is a constant for the same dataset; hence, the model ranking using G-mean_{PN} is the same as G-mean_{PU}.

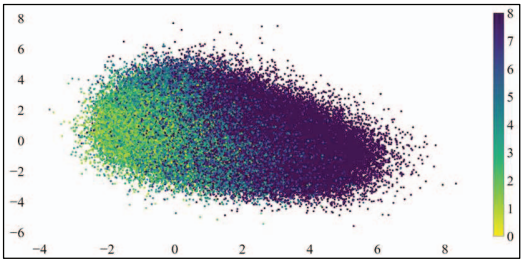

Figure 3: Embeddings of security posts in our *development set*. Note: Color bar shows the number of security keywords.

Therefore, we used G-mean_{PU} as the determinant metric to select the optimal PU learning model. G-mean_{PU} is the best metric for PU learning, but its value can be arbitrarily larger than one. Thus, we needed other metrics in Table 3 to provide a more human-interpretable yet less accurate evaluation of PUMiner. For evaluating on the ground-truth datasets, we also used MCC since it considers the results of both the positive and negative classes.

5 RESEARCH QUESTIONS AND RESULTS

5.1 *RQ1:* Is PUMiner effective for mining security posts from developer Q&A websites?

Motivation. Our work formulates the retrieval of security posts on Q&A websites as a PU learning problem. RQ1 evaluates the first attempt to address this new problem with PUMiner. Our framework allows researchers and practitioners to gather security information without the negative class.

Approach. We first visualize the doc2vec feature vectors using Principal Component Analysis (PCA) [21]. We then perform stratified 10-fold cross-validation and report the results of different configurations of PUMiner (see Table 2) with respect to the first six evaluation metrics in Table 3.

Results. We first showed that doc2vec context-aware embeddings could capture the transition from the non-security to security regions (i.e., yellow to purple based on the security keyword count (*kw_count*)) in Figure 3 and Figure 4. This confirmed our assumption in section 3.4 that the same-class posts would be close to each other in their respective regions, making them distinguishable. Figure 4 also demonstrated that there were only few security (purple) posts in the unlabelled set.

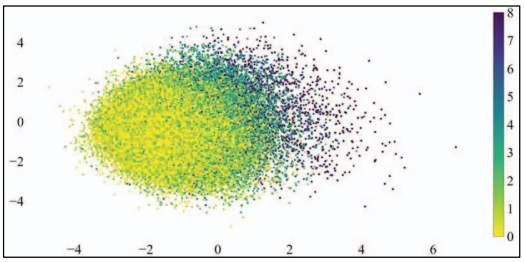

Figure 4: Embeddings of unlabelled posts in our *development set*. Note: Color bar shows the number of security keywords.

Table 4: Cross-validation performance of the optimal models of six classifiers

Metric	LR	SVM	KNN	RF	XGB	LGBM
$Recall_{PU}$	0.869	0.869	**0.893**	0.862	**0.893**	**0.893**
$Precision_{PU\text{-}LB}$	0.876	0.876	0.815	0.853	**0.903**	0.902
$Precision_{PU\text{-}UB}$	0.901	0.901	0.838	0.878	**0.928**	0.927
$F1\text{-}Score_{PU\text{-}LB}$	0.873	0.873	0.852	0.857	**0.898**	0.897
$F1\text{-}Score_{PU\text{-}UB}$	0.885	0.885	0.864	0.870	**0.910**	**0.910**
$G\text{-}mean_{PU}$	1.521	1.521	1.454	1.468	**1.611**	1.609

This finding showed that the unlabelled centroid could approximately represent the negative class.

Next, Table 4 gives the optimal validation results of each classifier. XGB with number of estimators = 500, max_depth = unlimited, and max_leaf_nodes = 300 performed the best in every metric. No common optimal configuration was found for all models, suggesting that tuning process is important [15, 56, 57]. Also, the ratio of security to unlabelled posts was 1:1, and about 2.5% of the unlabelled set were security (see section 4.3), implying that 51.25% of the whole set was security. Thus, $G\text{-}mean_{PN}$ (XGB) = $\sqrt{G\text{-}mean_{PU} \times P(y=1)} = \sqrt{1.611 \times 0.5125} = 0.909$ (see Eq. (3)). The worst approximated $G\text{-}mean_{PN}$ (KNN) was still 0.863, meaning that PUMiner performed well in overall (> 0.85 $G\text{-}mean_{PN}$) for retrieving security posts. $G\text{-}mean_{PU}$ of ensemble models (RF, XGB, and LGBM) were confirmed significantly better than that of single ones (LR, SVM, and KNN) using one-tailed non-parametric Mann-Whitney U-test [58] with P-value of 6.17×10^{-4}.

We also investigated the effect of alpha (α) in the first stage of PUMiner on the performance (see Figure 5). Higher values of alpha resulted in higher Precision and lower Recall. Intuitively, increasing alpha would move the selected negatives more towards the real positive class. It would then make the negative class noisier (i.e., more security posts in it), and produce more false negatives (low Recall) yet fewer false positives (high Precision). Alpha ≥ 1 also gave more stable (less changing) results than alpha < 1, and produced better average F1-Score and G-mean. This was confirmed when five out of six classifiers used alpha = 1.1 in their optimal models; whereas, KNN used alpha = 0.9. It might be because KNN is instance-based learning [21, 29], and thus it requires more refined/reliable negatives to learn the patterns. We recommend trying alpha ≥ 1 first for better baseline performance.

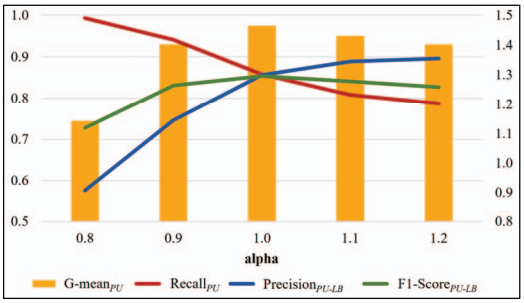

Figure 5: Effect of alpha on the average performance of PU learning models. Notes: Right axis is for $G\text{-}mean_{PU}$. Left axis is for the other three metrics.

Table 5: Hyperparameter tuning for the baseline models. Note: The optimal configurations are in bold.

Model	Hyperparameter	Values
1-PU	alpha	0.8, 0.9, **1.0**, 1.1, 1.2
OC-SVM	kernel	**linear**, rbf
	nu (Proportion of non-security posts)	**0.01**, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9
PSF	similarity threshold	**0.5**, 0.6, 0.7, 0.8, 0.9

5.2 *RQ2:* How does PUMiner perform compared to other learning approaches for mining security posts from developer Q&A websites?

Motivation. We can retrieve security posts using positive-only learning, a.k.a. novelty detection. One-class SVM (OC-SVM) [23, 59] is widely used for novelty detection by building a positive-class boundary with a max-margin SVM, while the region outside the boundary would be negative. Another novelty-detection method, Positive-Similarity Filtering (PSF) [24], classifies a new document as positive if the cosine similarity between its feature vector and that of any existing labelled positive document is higher than a threshold. RQ2 compares PUMiner with OC-SVM, PSF, and one-stage (centroid-based learning) PU (1-PU) models.

Approach. RQ2 compares the performance of the optimal PUMiner model in RQ1 on three unseen testing sets ($Test_{dev}$, $Test_{norm}$ and $Test_{hard}$ in section 4.1) against OC-SVM, PSF and 1-PU models. We optimize the baseline models on 90% data of the *development set* similarly to PUMiner. OC-SVM and PSF are only trained on positive samples and the same context-aware features are used for each model.

Results. Table 5 shows the optimal hyperparameters of the three baseline models. We compared such models with the optimal PUMiner model (XGB) reported in RQ1. Table 6 reports the performance on the labelled $Test_{norm}$ and $Test_{hard}$ sets, where the true estimation of all classification metrics can be computed. PUMiner obtained the best overall performance for both testing sets. On $Test_{norm}$, PUMiner outperformed the three baseline models in all metrics, except $Recall_{PN}$ for OC-SVM and $Precision_{PN}$ for 1-PU model. MCC_{PN} of PUMiner was 0.084, 0.534 and 0.906 points higher than those of 1-PU, PSF and OC-SVM models, respectively. OC-SVM had the best $Recall_{PN}$, but its MCC_{PN} was negative and even worse than a random classifier.

Table 6: Performance comparison on the $Test_{norm}$ and $Test_{hard}$ sets. Note: The results on the $Test_{hard}$ set are in parentheses.

Model	$Recall_{PN}$	$Precision_{PN}$	$F1\text{-}Score_{PN}$	$G\text{-}mean_{PN}$	MCC_{PN}
PUMiner	**0.951** (0.745)	0.943 (0.973)	**0.947** (**0.844**)	**0.947** (**0.852**)	**0.894** (**0.745**)
1-PU	0.849 (0.600)	**0.951** (**0.991**)	0.897 (0.748)	0.899 (0.771)	0.810 (0.647)
OC-SVM	**0.951** (**0.948**)	0.499 (0.489)	0.654 (0.645)	0.688 (0.681)	-0.012 (-0.13)
PSF	0.377 (0.758)	0.833 (0.692)	0.519 (0.724)	0.560 (0.724)	0.360 (0.423)

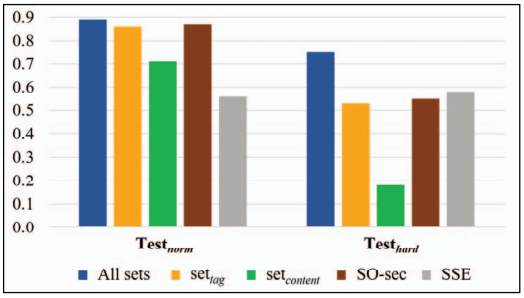

Figure 6: MCC$_{PN}$ of different positive datasets on the Test$_{norm}$ and Test$_{hard}$ sets. Note: SO-sec is set$_{tag}$ + set$_{content}$.

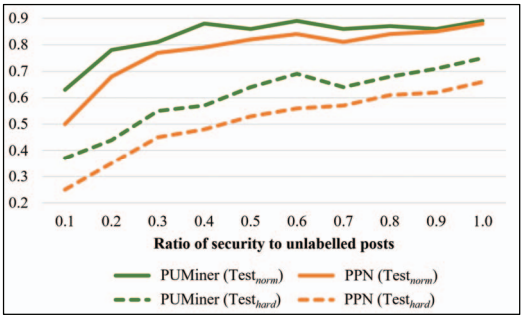

Figure 7: MCC$_{PN}$ of PUMiner and PPN on the Test$_{norm}$ and Test$_{hard}$ sets with different ratios of security to unlabelled posts in the *development set*.

1-PU model had the best Precision$_{PN}$, yet its Recall$_{PN}$ was much lower than PUMiner. On Test$_{hard}$, PUMiner produced the best MCC$_{PN}$ value of 0.745, which was 0.0918, 0.322 and 0.875 points superior to those of 1-PU, PSF and OC-SVM models, respectively. We also recorded that OC-SVM had the highest Recall$_{PN}$ yet poor Precision$_{PN}$, and vice versa for 1-PU model on Test$_{hard}$. Also, PSF was the worst model with the lowest F1-Score and G-mean for identifying security posts. We obtained the same patterns of results on Test$_{dev}$. In addition, we observed that Test$_{norm}$ > Test$_{dev}$ and Test$_{hard}$ < Test$_{dev}$ in terms of performance, which is reasonable since Test$_{dev}$ contains both normal and borderline test cases, proving that our sample selection was not biased. These findings have also reinforced the potential value of using unlabelled data and the second stage of PUMiner. Importantly, the heuristics in section 3.3 could not correctly detect the type of any post in Test$_{hard}$, which highlights the usefulness of PUMiner for reliable and large-scale security posts mining from Q&A websites.

5.3 *RQ3:* How do the source and size of training data affect the performance of PUMiner?

Motivation. The major advantage of PU learning is working with partially labelled positive samples and a large amount of unlabelled data, which cannot be handled by traditionally fully-supervised learning. RQ3 investigates the impact of data source and size on the performance of PUMiner.

Approach. We first investigate set$_{tag}$, set$_{content}$ and Security StackExchange security datasets. We then change the amount of labelled positive and unlabelled data to evaluate our PUMiner. We also compare the optimal PUMiner model with a fully supervised model that considers all unlabelled data as pseudo-negatives (PPN) to investigate the impact of the unlabelled set on the model performance. We use the same optimal model configurations of PUMiner and features for PPN. We would only report the results on the Test$_{norm}$ and Test$_{hard}$ sets as Test$_{dev}$ changes with each value of the positive ratio, which makes the results incomparable.

Results. Figure 6 shows that Stack Overflow (SO) was more helpful for identifying the security posts in Test$_{norm}$ set, while Security StackExchange (SSE) contributed more information to predict the posts in Test$_{hard}$ set. The security posts from SO (SO-sec) using our heuristics were more refined, while discussion on SSE was less restricted due to the informal nature of Q&A forums. Thus, the normal security posts were better captured using the patterns of SO-sec. In contrast, the security context of the posts in Test$_{hard}$ was less explicit; hence, the more diverse content of SSE could better cover such borderline cases. Overall, combining SO-sec and SSE sources has been useful for retrieving security posts in both normal and extreme scenarios. Set$_{tag}$ also outperformed set$_{content}$ in both cases, probably due to the nearly four-time difference in size (102,046 vs. 27,424).

Next, when varying the ratio of security to unlabelled posts, PUMiner consistently had higher MCC$_{PN}$ than PPN (see Figure 7). This has demonstrated that fully-supervised learning is not as effective as PUMiner when there is unlabelled data, confirming our statement made in section 3.4. PPN approached PUMiner in Test$_{norm}$ as the ratio increased. A reason might be that the results became more dominated by the increasing size of the positive set than the same positive posts in the unlabelled set. Note that PUMiner also outperformed 1-PU, OC-SVM and PSF models for each value of the positive ratio.

We also examined the cases when unlabelled set increased by ten and 100 times (i.e., 1,346,926 and 12,999,226 posts). For the 1:10 case, PUMiner achieved MCC$_{PN}$ of 0.78 and 0.46 for Test$_{norm}$ and Test$_{hard}$ sets, respectively. For the 1:100 case, MCC$_{PN}$ values of PUMiner were still 0.65 (Test$_{norm}$) and 0.35 (Test$_{hard}$), significantly surpassing PPN by 160% and 106%, respectively. These findings demonstrate the robustness of PUMiner for information retrieval on Q&A sites even with huge unlabelled data.

6 DISCUSSION

6.1 PUMiner Framework and Beyond

PUMiner is the best security-retrieval model as shown in section 5. Moreover, PUMiner is efficient as it took only 1,400 seconds to learn 360,000 posts in the *development set*, and less than four seconds to predict the security relevance for 770 posts in each of the sets (Test$_{norm}$ and Test$_{hard}$). PUMiner also allows incremental training on new posts, making it suitable to keep up with the fast and large-scale update of Q&A websites. Using the optimal PUMiner model in RQ1 and our heuristics with *kw_count* ≥ 5 and *kw_ratio* ≥ 0.03, we have identified and released 104,024 new

security posts on Stack Overflow (only 1% *FP* in 385 random posts) at [25] for future software security study. PUMiner can be extended to 170+ other topics on StackExchange [60], many of which are related to Software Engineering (SE).

We still found several misclassification patterns that can improve PUMiner in future work. For *FP*s, PUMiner struggled with the non-security posts about tools appearing in the security context, e.g., SO-1967980/44775539 about *Spring (boot)* framework were confused with *Spring security*. For *FN*s, PUMiner tended to miss security posts with implicit security context (e.g., SSE-34394 about *REcon* security conference or SO-37183524 about *OAuth* yet embedded in links and function calls), or about infrequent (< 1%) security tools or techniques (e.g., SSE-134794 about *Snort* or SO-16455365 about *Diffie-Hellman*). *Snort* and *Diffie-Hellman* only appeared in 432 and 1,426 posts, respectively, out of 181,696 total posts in our security dataset in section 4.1.

6.2 Threats to Validity

One potential threat to validity is with our data selection. We did use random sampling to avoid any bias. Our testing sets still involved human judgment, but we performed cross-checking with three people to make sure that the selected ones were as accurate and objective as possible.

Another threat is that our optimal hyperparameters of PUMiner might not produce the absolute highest results. However, it is nearly impossible to try all the combinations. To minimize this threat, we tried to follow previous practices, and then ran our models with stratified 10-fold cross-validation. We also showed that PUMiner performed much better than other existing models.

Our generalizability is also a concern. We used the largest training dataset, statistically significant size of testing sets [28], and confirmed our results with a statistical confidence level > 95%. We also published our code and models at [25] to support future reuse and extension of PUMiner.

7 RELATED WORK

7.1 Stack Overflow for SE Research

Stack Overflow (SO) has long been used to support various SE tasks such as API usage/documentation [61-63], code comment generation [64], code debugging/fixing [65, 66] and code clone detection [67, 68]. Among these studies, [61] was evaluated in the context of unlabelled data, but their labelled dataset contained both the positive and negative classes. Thus, it was less challenging than our study since we only have labelled positive posts and no negative class. Other studies identified topics of different domains [26, 69-71] on SO using unsupervised models (e.g., Latent Dirichlet Allocation (LDA) [72]), requiring no label. However, it is nearly impossible to directly guide such topic models to retrieve domain-specific posts. In addition, tag recommendation on Q&A websites is related to our task, but most of the existing techniques [73-75] required fully labelled data (post-tag pairs). It is also very difficult to obtain all relevant tags of a domain.

7.2 Mining Security Content from Open Sources

Yang et al. [12] identified and analyzed thirty security topics on SO using LDA [72]. However, they only obtained 30,054 security posts, which were at least six times fewer than ours. Such missing posts may lead to an overlook of current security issues and countermeasures. There is also active research on software vulnerabilities analytics using CVE/NVD [15, 76, 77], but these sources do not support online discussion for developers, and security issues and countermeasures may not be updated in time [78, 79]. Pletea et al. [14] proposed a list of security keywords to perform sentimental analysis of security discussions on GitHub. However, this previous list still misses many security keywords and we have proposed an updated list (see APPENDIX) that has two times more security keywords than [14]. Recently, [80] and [24] have used novelty detection methods for security retrieval. PUMiner utilizing unlabelled data has performed much better than one-class SVM and positive-similarity filtering used in these two studies.

8 CONCLUSIONS AND FUTURE WORK

We have proposed a novel framework, PUMiner, to automatically mine security content from Q&A websites where only some positive posts are known and the rest is unlabelled. Extensive experiments on more than 17.3 million posts from Stack Overflow and Security StackExchange have shown that PUMiner is a promising approach by outperforming one-class SVM, positive-similarity filtering and one-stage PU models on unseen posts. PUMiner could also predict the cases where keyword matching totally missed with an MCC of 0.745. Notably, with only 1% labelled positive posts, PUMiner was still 160% better than fully-supervised learning. Overall, PUMiner significantly reduces the human effort to retrieve information from large-scale and mostly unlabelled open sources in the SE community.

In the future, we would like to extend our PUMiner to other related domains such as security vulnerability analytics. We also plan to develop more sophisticated models and incorporate them into PUMiner to improve the effectiveness of information retrieval on developer Q&A websites.

ACKNOWLEDGMENTS

The work was supported by the Cyber Security Research Centre Limited whose activities are partially funded by the Australian Government's Cooperative Research Centres Programme. This work was also supported with supercomputing resources provided by the Phoenix HPC service at the University of Adelaide.

APPENDIX

234 security keywords used in this work: access control, access role, adware, adversarial, malware, spyware, ransomware, aes, antivirus, asset, audit, authority, authorise, availability, bitlocker, biometric, blacklist, botnet, buffer overflow, burp, ctf, capture the flag, cbc, certificate, checksum, cipher, clearance, confidential, clickfraud, clickjacking, cloudflare, cookie, crc, credential, crypt, csrf, ddos, danger, data exfiltrate, data exfiltration, data breach, decode, defence, defense, defensive programming, delegation, denial of service, diffie hellman, directory traversal, disclose, disclosure, dmz, dotfuscator, dsa, ecdsa, encode, encrypt, escrow, exploit, evil twin, fingerprint, firewall, forge, forgery, fuzz, fraud, gnupg, gss api, hack, hash, hijacking, hmac, honeypot, honey pot, hsm, inject, insecure, integrity, intrusion, intruder, ipsec, kerberos, ldap, login, metasploit, meterpreter, malicious, md5, nessus, nonce, nss, oauth, obfuscate, openssl, openssh, openvas, open auth, open redirect, openid, owasp, password, pbkdf2, pci dss, pgp, phishing, pki, privacy, private key, privilege, privilege escalation, permission escalation, public key, pcidss, pen test, penetration test, protect, rainbow table. rbac. rc4. repudiation. rfc 2898. riindael. rootkit. rsa. safe. salt. saml. sanitise. sandbox. scam. script kiddie, scripting, security, sftp, sha, shell code, shib boleth, signature, signed, signing, single sign on, smart assembly, smartassembly, snif, spam, spnego, spoof, ssh, ssl, sso, steganography, tampering, theft, threat, tls, transport, tunneling, tunnelling, trojan, trust, two factor, user account, user name, violate, validate, virus, white list, worm, x 509, x.509, xss, xxe, ssrf, zero day, 0 day, zombie computer, attack, vulnerability, attack vector, authentication, cross site, sensitive information, leak, information exposure, path traversal, use after free, double free, man in the middle, man in middle, mitm, poisoning, unauthorise, dot dot slash, bypass, session fixation, forced browsing, nvd, cwe, cve, capec, cpe, common weakness enumeration, common platform enumeration, crack, xml entity expansion, http parameter pollution, eavesdropping, cryptanalysis, http flood, xml flood, udp flood, tcp flood, tcp syn flood, steal, ssl flood, j2ee misconfiguration, asp.net misconfiguration, improper neutralisation, race condition, null pointer dereference, untrusted pointer dereference, trap door, back door, time bomb, xml bomb, logic bomb, captcha, deadlock, missing synchronisation, incorrect synchronisation, improper synchronisation, illegitimate, breach, sql injection

REFERENCES

[1] Security breaches in 2019. Retrieved 1/12/2019 from https://us.norton.com/internetsecurity-emerging-threats-2019-data-breaches.html.

[2] Common Vulnerabilities and Exposures. Retrieved 1/12/2019 from https://cve.mitre.org/.

[3] National Vulnerability Database. Retrieved 1/12/2019 from https://nvd.nist.gov/.

[4] Common Weakness Enumeration. Retrieved 1/12/2019 from https://cwe.mitre.org/.

[5] Common Attack Pattern Enumeration and Classification. Retrieved 1/12/2019 from https://capec.mitre.org/.

[6] Open Web Application Security Project Retrieved 1/12/2019 from https://www.owasp.org/index.php/Main_Page.

[7] SQL injection mitigations in CWE. Retrieved 1/12/2019 from https://cwe.mitre.org/data/definitions/89.html#oc_89_Potential_Mitigations.

[8] Deprecation of mysql_real_escape_string(). Retrieved 1/12/2019 from https://www.php.net/manual/en/intro.mysql.php.

[9] Discussion on SQL injection mitigations on Stack Overflow. Retrieved 1/12/2019 from https://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php.

[10] Stack Overflow. Retrieved 1/12/2019 from https://stackoverflow.com/.

[11] Rahman, M. S. 2016. An empirical case study on Stack Overflow to explore developers' security challenges (2016).

[12] Yang, X.-L., Lo, D., Xia, X., Wan, Z.-Y. and Sun, J.-L. 2016. What security questions do developers ask? A large-scale study of Stack Overflow posts. *J. Comput. Sci. Tech-ch.*, 31, 5 (2016), 910-924.

[13] Zahedi, M., Ali Babar, M. and Treude, C. 2018. An empirical study of security issues posted in open source projects. In *Proceedings of the 51st Hawaii International Conference on System Sciences*.

[14] Pletea, D., Vasilescu, B. and Serebrenik, A. 2014. Security and emotion: sentiment analysis of security discussions on GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, Hyderabad, India. ACM, 348-351.

[15] Le, T. H. M., Sabir, B. and Babar, M. A. 2019. Automated software vulnerability assessment with concept drift. In *Proceedings of the 16th International Conference on Mining Software Repositories*, Montreal, Quebec, Canada. IEEE Press, 371-382.

[16] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of Advances in Neural Information Processing Systems*, 3111-3119.

[17] Pennington, J., Socher, R. and Manning, C. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532-1543.

[18] Efstathiou, V., Chatzilenas, C. and Spinellis, D. 2018. Word embeddings for the software engineering domain. In *Proceedings of the 15th International Conference on Mining Software Repositories*, Gothenburg, Sweden. ACM, 38-41.

[19] Le, Q. and Mikolov, T. 2014. Distributed representations of sentences and documents. In *Proceedings of International Conference on Machine Learning*, 1188-1196.

[20] Han, J., Pei, J. and Kamber, M. 2011. *Data mining: concepts and techniques*. Elsevier.

[21] Friedman, J., Hastie, T. and Tibshirani, R. 2001. *The elements of statistical learning*. Springer series in statistics New York.

[22] Security StackExchange. Retrieved 1/12/2019 from https://security.stackexchange.com/.

[23] Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J. and Platt, J. C. 2000. Support vector method for novelty detection. In *Proceedings of Advances in neural information processing systems*, 582-588.

[24] Mendsaikhan, O., Hasegawa, H., Yamaguchi, Y. and Shimada, H. 2019. Identification of cybersecurity specific content using the doc2vec language model. In *Proceedings of 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, 396-401.

[25] Reproduction package. Retrieved 2/3/2020 from https://github.com/lhmtriet/PUMiner_MSR.

[26] Barua, A., Thomas, S. W. and Hassan, A. E. 2014. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empir. Softw. Eng.*, 19, 3 (2014), 619-654.

[27] Database schema of Stack Overflow. Retrieved 1/12/2019 from https://data.stackexchange.com/stackoverflow/query/new.

[28] Cochran, W. G. 2007. *Sampling techniques*. John Wiley & Sons.

[29] Murphy, K. P. 2012. *Machine learning: a probabilistic perspective*. MIT press.

[30] Bekker, J. and Davis, J. 2018. Learning from positive and unlabeled data: A survey. *arXiv preprint arXiv:1811.04820* (2018).

[31] Dam, H. K., Tran, T., Pham, T. T. M., Ng, S. W., Grundy, J. and Ghose, A. 2018. Automatic feature learning for predicting vulnerable software components. *IEEE Trans. Softw. Eng.* (2018).

[32] Hoang, T., Dam, H. K., Kamei, Y., Lo, D. and Ubayashi, N. 2019. DeepJIT: an end-to-end deep learning framework for just-in-time defect prediction. In *Proceedings of the 16th International Conference on Mining Software Repositories*, Montreal, Quebec, Canada. IEEE Press, 34-45.

[33] Kao, A. and Poteet, S. R. 2007. *Natural language processing and text mining*. Springer Science & Business Media.

[34] Porter, M. F. 1980. An algorithm for suffix stripping. *Program*, 14, 3 (1980), 130-137.

[35] Han, Z., Li, X., Xing, Z., Liu, H. and Feng, Z. 2017. Learning to predict severity of software vulnerability using only vulnerability description. In *Proceedings of 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 125-136.

[36] White, M., Vendome, C., Linares-Vásquez, M. and Poshyvanyk, D. 2015. Toward deep learning software repositories. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, Florence, Italy. IEEE Press, 334-345.

[37] Elkan, C. and Noto, K. 2008. Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 213-220.

[38] Mordelet, F. and Vert, J. P. 2014. A bagging SVM to learn from positive and unlabeled examples. *Pattern Recognit. Lett.*, 37 (2014), 201-209.

[39] Li, X. and Liu, B. 2003. Learning to classify texts using positive and unlabeled data. In *IJCAI*, 587-592.

[40] Hernández Fusilier, D., Montes-y-Gómez, M., Rosso, P. and Guzmán Cabrera, R. 2015. Detecting positive and negative deceptive opinions using PU-learning. *Inf. Process. Manag.*, 51, 4 (2015), 433-443.

[41] Google BigQuery. Retrieved 1/12/2019 from https://cloud.google.com/bigquery/.

[42] Security StackExchange queries. Retrieved 1/12/2019 from https://data.stackexchange.com/security/queries.

[43] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R. and Dubourg, V. 2011. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12, Oct (2011), 2825-2830.

[44] Bird, S. and Loper, E. 2004. NLTK: the natural language toolkit. In *ACL 2004 on Interactive Poster and Demonstration Sessions*. Association for Computational Linguistics, 31.

[45] Rehurek, R. and Sojka, P. 2010. Software framework for topic modelling with large corpora. In *Proceedings of LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer.

[46] Majumder, S., Balaji, N., Brey, K., Fu, W. and Menzies, T. 2018. 500+ times faster than deep learning: a case study exploring faster methods for text mining stackoverflow. In *Proceedings of the 15th International Conference on Mining Software Repositories*, Gothenburg, Sweden. ACM, 554-563.

[47] Ma, Y., Fakhoury, S., Christensen, M., Arnaoudova, V., Zogaan, W. and Mirakhorli, M. 2018. Automatic classification of software artifacts in open-source applications. In *Proceedings of the 15th International Conference on Mining Software Repositories*, Gothenburg, Sweden. ACM, 414-425.

[48] Kaggle. Retrieved 1/12/2019 from https://www.kaggle.com/.

[49] Walker, S. H. and Duncan, D. B. 1967. Estimation of the probability of an event as a function of several independent variables. *Biometrika*, 54, 1-2 (1967), 167-179.

[50] Cortes, C. and Vapnik, V. 1995. Support-vector networks. *Mach. Learn.*, 20, 3 (1995), 273-297.

[51] Altman, N. S. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *Am. Stat.*, 46, 3 (1992), 175-185.

[52] Ho, T. K. 1995. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*. IEEE, 278-282.

[53] Chen, T. and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 785-794.

[54] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. and Liu, T.-Y. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of Advances in Neural Information Processing Systems*, 3146-3154.

[55] Lee, W. S. and Liu, B. 2003. Learning with positive and unlabeled examples using weighted logistic regression. In *ICML*, 448-455.

[56] Treude, C. and Wagner, M. 2019. Predicting good configurations for GitHub and stack overflow topic models. In *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press, 84-95.

[57] Tantithamthavorn, C., McIntosh, S., Hassan, A. E. and Matsumoto, K. 2018. The impact of automated parameter optimization on defect prediction models. *IEEE Trans. Softw. Eng.* (2018).

[58] Mann, H. B. and Whitney, D. R. 1947. On a test of whether one of two random variables is stochastically larger than the other. *Annals of mathematical statistics* (1947), 50-60.

[59] Manevitz, L. M. and Yousef, M. 2001. One-class SVMs for document classification. *J. Mach. Learn. Res.*, 2, Dec (2001), 139-154.

[60] Stack Exchange sites. Retrieved 1/12/2019 from https://stackexchange.com/sites.

[61] Wang, S., Phan, N., Wang, Y. and Zhao, Y. 2019. Extracting API tips from developer question and answer websites. In *Proceedings of the 16th International Conference on Mining Software Repositories*, Montreal, Quebec, Canada. IEEE Press, 321-332.

[62] Treude, C. and Robillard, M. P. 2016. Augmenting api documentation with insights from stack overflow. In *Proceedings of the 38th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 392-403.

[63] Ye, X., Shen, H., Ma, X., Bunescu, R. and Liu, C. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, Austin, Texas. Association for Computing Machinery, 404–415.

[64] Wong, E., Yang, J. and Tan, L. 2013. Autocomment: Mining question and answer sites for automatic comment generation. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 562-567.

[65] Campos, E. C., Monperrus, M. and Maia, M. A. 2016. Searching stack overflow for API-usage-related bug fixes using snippet-based queries. In *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 232-242.

[66] Thiselton, E. and Treude, C. 2019. Enhancing Python compiler error messages via Stack Overflow. In *Proceedings of 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1-12.

[67] Ragkhitwetsagul, C., Krinke, J., Paixao, M., Bianco, G. and Oliveto, R. 2019. Toxic code snippets on stack overflow. *IEEE Trans. Softw. Eng.* (2019).

[68] Nishi, M. A., Ciborowska, A. and Damevski, K. 2019. Characterizing duplicate code snippets between stack overflow and tutorials. In *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press, 240-244.

[69] Bangash, A. A., Sahar, H., Chowdhury, S., Wong, A. W., Hindle, A. and Ali, K. 2019. What do developers know about machine learning: a study of ML discussions on StackOverflow. In *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press, 260-264.

[70] Bagherzadeh, M. and Khatchadourian, R. 2019. Going big: a large-scale study on what big data developers ask. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 432-442.

[71] Rosen, C. and Shihab, E. 2016. What are mobile developers asking about? a large scale study using stack overflow. *Empir. Softw. Eng.*, 21, 3 (2016), 1192-1223.

[72] Blei, D. M., Ng, A. Y. and Jordan, M. I. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3, Jan (2003), 993-1022.

[73] Wang, S., Lo, D., Vasilescu, B. and Serebrenik, A. 2018. EnTagRec++: An enhanced tag recommendation system for software information sites. *Empir. Softw. Eng.*, 23, 2 (2018), 800-832.

[74] Wang, S., Lo, D., Vasilescu, B. and Serebrenik, A. 2014. EnTagRec: An Enhanced Tag Recommendation System for Software Information Sites. In *Proceedings of 2014 IEEE International Conference on Software Maintenance and Evolution*, 291-300.

[75] Xia, X., Lo, D., Wang, X. and Zhou, B. 2013. Tag recommendation in software information sites. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 287-296.

[76] Ghaffarian, S. M. and Shahriari, H. R. 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM Comput. Surv.*, 50, 4 (2017), 1-36.

[77] Shahzad, M., Shafiq, M. Z. and Liu, A. X. 2012. A large scale exploratory analysis of software vulnerability life cycles. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 771-781.

[78] Feutrill, A., Ranathunga, D., Yarom, Y. and Roughan, M. 2018. The effect of common vulnerability scoring system metrics on vulnerability exploit delay. In *Proceedings of 2018 International Symposium on Computing and Networking (CANDAR)*, 1-10.

[79] Ruohonen, J. 2017. A look at the time delays in CVSS vulnerability scoring. *Applied Computing and Informatics* (2017).

[80] Le, B., Wang, G., Nasim, M. and Babar, M. A. 2019. Gathering cyber threat intelligence from Twitter using novelty classification. In *Proceedings of 2019 International Conference on Cyberworlds (CW)*, 316-323.