2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

Automated Software Vulnerability Assessment with Concept Drift

Triet Huynh Minh Le
School of Computer Science
The University of Adelaide
Adelaide, Australia
triet.h.le@adelaide.edu.au

Bushra Sabir
School of Computer Science
The University of Adelaide
Adelaide, Australia
bushra.sabir@adelaide.edu.au

M. Ali Babar
School of Computer Science
The University of Adelaide
Adelaide, Australia
ali.babar@adelaide.edu.au

***Abstract*— Software Engineering researchers are increasingly using Natural Language Processing (NLP) techniques to automate Software Vulnerabilities (SVs) assessment using the descriptions in public repositories. However, the existing NLP-based approaches suffer from *concept drift*. This problem is caused by a lack of proper treatment of new (out-of-vocabulary) terms for the evaluation of unseen SVs over time. To perform automated SVs assessment with *concept drift* using SVs' descriptions, we propose a systematic approach that combines both character and word features. The proposed approach is used to predict seven Vulnerability Characteristics (VCs). The optimal model of each VC is selected using our customized time-based cross-validation method from a list of eight NLP representations and six well-known Machine Learning models. We have used the proposed approach to conduct large-scale experiments on more than 100,000 SVs in the National Vulnerability Database (NVD). The results show that our approach can effectively tackle the *concept drift* issue of the SVs' descriptions reported from 2000 to 2018 in NVD even without retraining the model. In addition, our approach performs competitively compared to the existing word-only method. We also investigate how to build compact *concept-drift-aware* models with much fewer features and give some recommendations on the choice of classifiers and NLP representations for SVs assessment.**

Keywords—software vulnerability, machine learning, multi-class classification, natural language processing, mining software repositories

I. Introduction

Software Vulnerability (SV) is usually defined as a flaw or weakness in software code, that can potentially result in a cybersecurity attack [1]. Cybersecurity attacks reportedly led to a loss of more than 50 billion dollars to the U.S. economy in 2016 [2]. Different types of SVs have different levels of security threats to software-intensive systems [3]. It is important to assess SVs for prioritizing actions so that more severe SVs are patched before exploitations [4, 5]. Automation of SVs analysis and assessment has become an important area of research efforts. Identification of SVs' characteristics is a critical task for automation. It is asserted that SVs' public repositories, such as the National Vulnerability Database (NVD), can help identify SVs characteristics by analyzing their descriptions using Natural Language Processing (NLP) [6-8].

However, the vulnerability data have the temporal property since many new terms appear in the descriptions of SVs. Such terms are a result of the release of new technologies/products or discovery of a zero-day attack or SV; for example, the NVD received more than 13,000 new SVs in 2017 [9]. The appearance of new concepts makes the vulnerability data and patterns change over time [10-12], which is known as *concept drift* [13]. For example, the keyword Android has only started appearing in NVD since 2008, the year when Google released Android. It is being recognized that such new SVs terms cause problems for building the vulnerability assessment models.

Some previous studies [6, 14, 15] have suffered from *concept drift* by mixing the new and old SVs in the model validation step, which can lead to biased results as such approach accidentally merges the new information with the existing one. Moreover, the previous work of SVs analysis [6, 7, 14-18] used predictive models with only word features without reporting how to handle the new or extended concepts (e.g., new versions of the same software) in the new SVs' descriptions. The research on machine translation [19-22] has shown that the unseen (Out-of-Vocabulary (OoV)) terms can make existing word-only models less robust to future prediction due to their missing information. For vulnerability prediction, Zhoubing [8] did use random embedding vectors to represent the OoV words, which still discards the relationship between the new and old concepts. Such observations motivated us to tackle the research problem "**How to handle the *concept drift* issue of the vulnerability descriptions in public repositories to improve the robustness of automated SVs assessment?"** It appears to us that it is important to address the issue of *concept drift* to enable practical applicability of automated vulnerability assessment tools. To the best of our knowledge, there has been no existing work to systematically address the *concept drift* issue in SVs assessment.

To perform SVs assessment with *concept drift* using the vulnerability descriptions in public repositories, we present a Machine Learning (ML) model that utilizes both character-level and word-level features. We also propose a customized time-based version of cross-validation method for model selection and validation. Our cross-validation method splits the data by year to embrace the temporal relationship of SVs. We evaluate the proposed model on the prediction of seven Vulnerability Characteristics (VCs), i.e., Confidentiality, Integrity, Availability, Access Vector, Access Complexity, Authentication, and Severity. Our key contributions are:

1. We demonstrate the *concept drift* issue of SVs using concrete examples from NVD.
2. We investigate a customized time-based cross-validation method to select the optimal ML models for SVs assessment. Our method can help prevent future vulnerability information from being leaked into the past in model selection and validation steps.
3. We propose and extensively evaluate an effective Character-Word Model (CWM) to assess SVs using the descriptions with *concept drift*. We also investigate the performance of low-dimensional CWM models. We provide a GitHub repository[1] containing our models and associated source code.

[1] https://github.com/lhmtriet/MSR2019

2574-3864/19/$31.00 ©2019 IEEE
DOI 10.1109/MSR.2019.00063

Paper structure. Section II introduces the vulnerability description and VCs. Section III describes our proposed approach. Section IV presents the experimental design of this work. Section V analyzes the experimental results and discusses the findings. Section VI identifies the threats to validity. Section VII covers the related works. Section VIII concludes and suggests some future directions.

II. BACKGROUND

National Vulnerability Database [9] (NVD) is a well-known public repository that contains a huge amount of SVs information that is considered trustworthy as NVD is maintained by governmental bodies (National Cyber Security and Division of the United States Department of Homeland Security). NVD inherits the unique vulnerability identifiers and descriptions from Common Vulnerabilities and Exposures (CVE) [1]. NVD also adds an evaluation to each SV using the Common Vulnerability Scoring System (CVSS) [23, 24]. Currently, there are three versions of CVSS, in which the latest version (i.e., the third version) was introduced in 2015. The second CVSS version (CVSS 2) is also maintained.

In CVSS 2, an SV is evaluated based on three main criteria: Impact, Exploitability and Severity. Impact and Exploitability refer to the threats and exploitation procedures of each SV. Severity determines the level of severity of an SV based on Impact and Exploitability. The first two criteria can be further decomposed into (Confidentiality, Integrity, Availability) and (Access Vector, Access Complexity, Authentication), respectively (cf. Fig. 1). There are three separate values for each of the seven VCs. From the perspective of ML, this is a multi-class classification problem, which can be solved readily using ML algorithms. It is noted that Access Vector, Access Complexity and Authentication characteristics suffer the most from the issue of imbalanced data, in which the number of elements in the minority class is much smaller compared to those of the other classes.

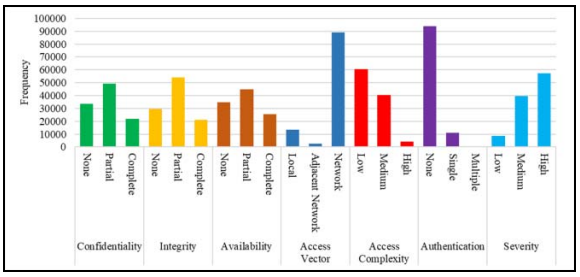

Fig. 1. Frequencies of each class of the seven VCs.

III. THE PROPOSED APPROACH

A. Approach Overview

The overall workflow of our proposed approach is given in Fig. 2. Our approach consists of three processes: model selection, model building and prediction. The first two processes work on the training set, while the prediction process performs on either a separate testing set or new vulnerability descriptions. The first model selection has two steps: Text preprocessing and Time-based k-fold cross-validation. Text preprocessing step (cf. section III.B) is necessary to reduce the noise in the text to build a better assessment model. Next, the preprocessed text enters the time-based k-fold cross-validation step to select the optimal classifier and NLP representations for each VC. It should be noted that this step only tunes the word-level models instead of the combined models of both word and character features. One reason is that the search space of the combined model is much larger than that of the word-only model since we at least have to consider different NLP representations for character-level features. The computational resource to extract character-level n-grams is also more than that of word-level counterparts. Section III.C provides more details about the time-based k-fold cross-validation method.

Next comes the model building process with four main steps: (*i*) word n-grams generation, (*ii*) character n-grams generation, (*iii*) feature aggregation and (*iv*) character-word model building. Steps (*i*) and (*ii*) use the preprocessed text in the previous process to generate word and character n-grams based on the identified optimal NLP representations of each VC. The word n-grams generation step (*i*) here is the same as the one in the time-based k-fold cross-validation of the previous process. An example of the word and character n-grams in our approach is given in Table 1. Such character n-grams increase the probability of capturing parts of OoV terms due to *concept drift* in vulnerability descriptions. Subsequently, both levels of the n-grams and the optimal NLP representations are input into the feature aggregation step (*iii*) to extract the features from the preprocessed text using our proposed algorithm in section III.D. This step also combines the aggregated character and word vocabularies with the optimal NLP representations of each VC to create the feature models. We save such models to transform the data of future prediction. In the last step (*iv*), the extracted features are trained with the optimal classifiers found in the model selection process to build the complete character-word models for each VC to perform automated vulnerability assessment with *concept drift*.

In the prediction process, new vulnerability description is first preprocessed using the same text preprocessing step. Then, the preprocessed text is transformed to create the features by the feature models saved in the model building process. Finally, the trained character-word models use such features to determine each VC.

TABLE 1. WORD AND CHARACTER N-GRAMS EXTRACTED FROM THE SENTENCE "HELLO WORLD". '_' REPRESENTS A SPACE.

n-grams	Word	Character
1	Hello, World	H, e, l, l, o, W, o, r, l, d
2	Hello World	He, el, ll, lo, o_, _W, Wo, or, rl, ld

B. Text preprocessing of vulnerability descriptions

The text preprocessing is an important step for any NLP task [25]. We use the following text preprocessing techniques: (*i*) removal of stop words and punctuations, (*ii*) conversion to lowercase and (*iii*) stemming. The stop words are combined from the default lists of *scikit-learn* [26] and *nltk* [27] libraries. We only remove the punctuations followed by at least one space or the ones at the end of a sentence. This punctuation removal method keeps the important words in the software and security contexts such as "*input.c*", "*man-in-the-middle*", "*cross-site*".

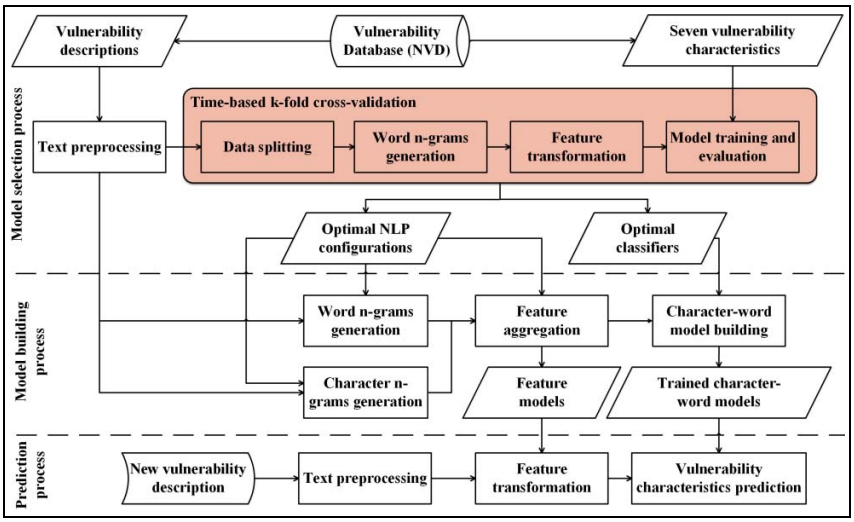

Fig. 2. Main workflow of our proposed model for vulnerability assessment with *concept drift*.

Subsequently, the stemming step is done using the Porter Stemmer algorithm [28] in *nltk* library. Stemming is needed to avoid two or more words with the same meaning but in different forms (e.g., "*allow*" vs. "*allows*"). The main goal of stemming is to retrieve consistent features (words), thus any algorithm that can return each word's root should work. Researchers may use lemmatization, which is relatively slower as it also considers the surrounding context.

C. Model selection with time-based k-fold cross-validation

We propose a time-based cross-validation method (cf. Fig. 3) to select the best classifiers and NLP representations for each VC. The idea has been inspired by the time-series domain [29]. As shown in Fig. 2, our method has four steps: (*i*) data splitting, (*ii*) word n-grams generation, (*iii*) feature transformation, and (*iv*) model training and evaluation. Data splitting explicitly considers the time order of SVs to ensure that in each pass/fold, the new information of the validation set does not exist in the training set, which maintains the temporal property of SVs. The new terms can appear in different time during a year; thus, the preprocessed text in each fold is split by the year explicitly, not by equal sample size, e.g., SVs from 1999 to 2010 are for training and those of 2011 are for validation in a pass/fold.

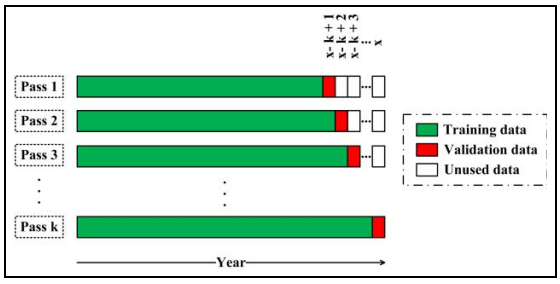

Fig. 3. Our proposed time-based cross-validation method.
Note: x is the final year in the original training set,
k is the number of cross-validation folds.

After data splitting in each fold, we use the training set to generate the word n-grams. Subsequently, with each of the eight NLP configurations in Table 2, the feature transformation step uses the word n-grams as the vocabulary to transform the preprocessed text of both training and validation sets into the features for building a model. We create the NLP configurations from various values of n-grams combined with either term frequency or tf-idf measure. Uni-gram with term frequency is also called Bag-of-Words (BoW). These NLP representations have been selected since they are popular and have performed well for SVs analysis [6, 14, 17].

TABLE 2. EIGHT CONFIGURATIONS OF NLP REPRESENTATIONS USED FOR MODEL SELECTION. NOTE: '✓' IS SELECTED, '-' IS NON-SELECTED.

Configuration	Word n-grams	tf-idf
1	1	-
2	1	✓
3	1-2	-
4	1-3	-
5	1-4	-
6	1-2	✓
7	1-3	✓
8	1-4	✓

For each NLP configuration, the model training and evaluation step trains six classifiers (cf. section IV.C) on the training set and then evaluates the models on the validation set using different evaluation metrics (cf. section IV.D). The model with the highest average cross-validated performance is selected for the current VC. The process is repeated for every VC, then the optimal classifiers and NLP representations are returned for all seven VCs.

D. Feature aggregation algorithm

We propose **Algorithm 1** to combine word and character n-grams in the model building process to create the features for our character-word model. Six inputs of the algorithm are (*i*) input descriptions, (*ii*) word n-grams, (*iii*) character n-grams, (*iv*) the minimum, (*v*) the maximum number of character n-grams, and (*vi*) the optimal NLP configuration of the current VC. The main output is a feature matrix containing the term weights of the documents transformed by the aggregated character and word vocabularies to build the character-word models. We also output the character and word feature models for future prediction.

Algorithm 1. Feature aggregation algorithm to transform the documents with the aggregated word and character-level features.

Input: List of vulnerability descriptions: D_{in}

Set of word-level n-grams: $F_w = \{f_{1w}, f_{2w}, ..., f_{mw}\}$

Set of character-level n-grams: $F_c = \{f_{1c}, f_{2c}, ..., f_{mc}\}$

The minimum and maximum character n-grams: min_{n-gram} and max_{n-gram}

The optimal NLP configuration of the current VC: *config*

Output: The aggregated data matrix: $\mathbf{X}_{agg}$

The word and character feature models: $model_w$, $model_c$

1 *slt_chars* ← empty set
2 **foreach** $f_i \in F_c$ **do**
3 *tokens* ← f_i trimmed and split by space
4 **if** (size of *tokens* = 1) **and** ((length of the first element in *tokens*) > 1) **then**
5 *slt_chars* ← *slt_chars* + {*tokens*}
6 **end if**
7 **end foreach**
8 *diff_words* ← $F_w - slt\_chars$
9 $model_w$ ← Feature_transformation (*diff_words, config)*
10 $model_c$ ← Feature_transformation (*slt_chars*, $min_{n-gram} - 1$, max_{n-gram}, *config*)
11 $\mathbf{X}_{word}$ ← D_{in} transformed with $model_w$
12 $\mathbf{X}_{char}$ ← D_{in} transformed with $model_c$
13 $\mathbf{X}_{agg}$ ← horizontal_append($\mathbf{X}_{word}$, $\mathbf{X}_{char}$)
14 **return** $\mathbf{X}_{agg}$, $model_w$, $model_c$

Steps 2-7 of the algorithm filter the character features. More specifically, step 3 removes (trims) the spaces from both ends of each feature. Then, we split such feature by space(s) to determine how many words to which the character(s) belongs. Subsequently, steps 4-6 retain only the character features that are parts of single words (size of *tokens* = 1), except the single characters such as *x*, *y*, *z* ((length of the first element in *tokens*) > 1). The n-gram characters with space(s) in between represent more than one word, which can make the classifier more prone to overfitting. Similarly, single characters are too short and they can belong to too many words, which is likely to make the model hardly generalizable. In fact, '*a*' is a meaningful single character, but it has been already removed as a stop word. The characters can even represent a whole word (e.g., "*attack*" token with $max_{n-gram} \geq 6$). In such cases, step 8 removes the duplicated word-level features ($F_w - slt\_chars$). Based on the assumption that unseen or misspelled terms can share common characters with existing words, such choice can enhance the probability of the model capturing the OoV words in the new descriptions. Retaining only the character features also helps reduce the number of features and the model overfitting. After that, steps 9-10 define the feature models $model_w$ and $model_c$ using the word (*diff_words*) and character (*slt_chars*) vocabularies, respectively, along with the NLP configurations to transform the input documents into feature matrices for building the model. Steps 11-12 then use the two defined word and character models to actually transform the input documents D_{in} into feature matrices $\mathbf{X}_{word}$ and $\mathbf{X}_{char}$, respectively. Step 13 concatenates two feature matrices by columns. Finally, step 14 returns the final aggregated feature matrix $\mathbf{X}_{agg}$ along with both word and character feature models namely $model_w$ and $model_c$.

IV. EXPERIMENT DESIGN

All the classifiers and NLP representations (n-grams, term frequency, tf-idf) in this work have been implemented in *scikit-learn* [26] and *nltk* [27] libraries in Python. Our code ran on a fourth-generation Intel Core i7-4200HQ CPU (four cores) running at 2.6 GHz with 16 GB of RAM.

A. Research Questions

Our research aims at addressing the *concept drift* issue in SVs' descriptions to improve the robustness of both model selection and prediction steps. We have evaluated our two-phase character-word model. The first phase selects the optimal models for each VC. The second phase incorporates the character features to build character-word models. We raise and answer four Research Questions (RQs):

- **RQ1**: *Is our time-based cross-validation more effective than a non-temporal method to handle concept drift in the model selection step for vulnerability assessment?* To answer RQ1, we first identify the new terms in the vulnerability descriptions. We associate such terms with their release or discovery years. We then use qualitative examples to demonstrate the information leakage in the non-temporal model selection step. We also quantitatively compare the effectiveness of the proposed time-based cross-validation method with a traditional non-temporal one to address the temporal relationship in the context of vulnerability assessment.
- **RQ2:** *Which are the optimal models for multi-classification of each vulnerability characteristic?* To answer RQ2, we present the optimal models (i.e., classifiers and NLP representations) using word features for each VC selected by a five-fold time-based cross-validation method (cf. section III.C). We also compare the performance of different classes of models (single vs. ensemble) and NLP representations to give recommendations for future use.
- **RQ3**: *How effective is our character-word model to perform automated vulnerability assessment with concept drift?* For RQ3, we first demonstrate how the OoV phrases identified in RQ1 can affect the performance of the existing word-only models. We then highlight the ability of the character features to handle the *concept drift* issue of SVs. We also compare the performance of our character-word model with those of the word-only (without handling *concept drift*) and character-only models.
- **RQ4**: *To what extent can low-dimensional model retain the original performance?* The features of our proposed model in RQ3 are high-dimensional and sparse. Hence, we evaluate the dimensionality reduction technique (i.e., Latent Semantic Analysis [30]) and the sub-word embeddings (i.e., fastText [31, 32]) to show how much information of the original model is approximated in lower dimensions. The work done for answering RQ4 facilitates the building of more efficient *concept-drift*-aware predictive models.

B. Dataset

We retrieved 113,292 SVs from NVD in JSON format. The dataset contains the SVs from 1988 to 2018. We discarded 5926 SVs that contain "** REJECT **" in their descriptions since they have been confirmed duplicated or

incorrect by experts. Seven VCs of CVSS 2 (cf. section II) were used as our SVs assessment metrics. It turned out that there are 2242 SVs without any value of CVSS 2. Therefore, we also removed such SVs from our dataset. Finally, we obtained a dataset containing 105,124 SVs along with their descriptions and the values of seven VCs indicated previously. For evaluation purposes, we followed the work in [6] to use the year of 2016 to divide our dataset into training and testing sets with the sizes of 76,241 and 28,883, respectively. The primary reason for splitting the dataset based on the time order is to consider the temporal relationship of SVs.

C. Vulnerability classification machine learning models

To solve our multi-class classification problem, we used six well-known ML models. These classifiers have achieved great results in recent data science competitions such as Kaggle [33]. We provide brief descriptions and the hyperparameters of each classifier below.

- **Naïve Bayes (NB)** [34] is a simple probabilistic model that is based on Bayes' theorem. This model assumes that all the features are conditionally independent with respect to each other. In this study, NB has no tuning hyperparameter during the validation step.
- **Logistic Regression (LR)** [35] is a linear classifier in which the logistic function is used to convert the linear output into probability. The one-vs-rest scheme is applied to split the multi-class problem into multiple binary classification problems. In this work, we select the optimal value of the regularization parameter for LR from the list of values: 0.01, 0.1, 1, 10, 100.
- **Support Vector Machine (SVM)** [36] is a classification model in which a maximum margin is determined to separate the classes. For NLP, the linear kernel is preferred because of its more scalable computation and sparsity handling [37]. The tuning regularization values of SVM are the same as LR.
- **Random Forest (RF)** [38] is a bagging model in which multiple decision trees are combined to reduce the variance and sensitivity to noise. The complexity of RF is mainly controlled by (*i*) the number of trees, (*ii*) maximum depth, and (*iii*) maximum number of leaves. (*i*) tuning values are: 100, 300, 500. We set (*ii*) to *unlimited*, which makes the model the highest degree of flexibility and easier to adapt to new data. For (*iii*), the tuning values are 100, 200, 300 and *unlimited*.
- **XGBoost - Extreme Gradient Boosting (XGB)** [39] is a variant Gradient Boosting Tree Model (GBTM) in which multiple weak tree-based classifiers are combined and regularized to enhance the robustness of the overall model. Three hyperparameters of XGB that require tuning are the same as RF. It should be noted that the *unlimited* value of the maximum number of leaves is not applicable to XGB.
- **Light Gradient Boosting Machine (LGBM)** [40] is a light-weight version of GBTM. Its main advantage is the scalability since the sub-trees are grown in a leaf-wise manner rather than depth-wise of other GBT algorithms. Three hyperparameters of LGBM that require tuning are the same as XGB.

In this work, we consider NB, LR and SVM as single models, while RF, XGB and LGBM as ensemble models.

D. Evaluation Metrics

Our multi-class classification problem can be decomposed into multiple binary classification problems. To define the standard evaluation metrics for a binary problem [6-8], we first describe four possibilities as follows.

- True positive (*TP*): The classifier correctly predicts that an SV has a particular characteristic.
- False positive (*FP*): The classifier incorrectly predicts that an SV has a particular characteristic.
- True negative (*TN*): The classifier correctly predicts that an SV does not have a particular characteristic.
- False negative (*FN*): The classifier incorrectly predicts that an SV does not have a particular characteristic.

Based on *TP*, *FP*, *TN*, *FN*, *Accuracy*, *Precision*, *Recall* and *F-Score* can be defined accordingly in (1), (2), (3), (4).

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (1)$$

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

$$F-Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (4)$$

Whilst *Accuracy* measures the global performance of all classes, *F-Score* (a harmonic mean of *Precision* and *Recall*) evaluates each class separately. Such local estimate like *F-Score* is more favorable to the imbalanced VCs such as Access Vector, Access Complexity, Authentication, Severity (cf. Fig. 1). In fact, there are several variants of *F-Score* for multi-class classification problem namely *Micro*, *Macro* and *Weighted F-Scores*. In the case of multi-class classification, *Micro F-Score* is actually the same as *Accuracy*. For *Macro* and *Weighted F-Scores*, the former does not consider class distribution (the number of elements in each class) for computing *F-Score* of each class; whereas, the latter does. To account for the balanced and imbalanced VCs globally and locally, we use *Accuracy*, *Macro,* and *Weighted F-Scores* to evaluate our models. For model selection, if there is a performance tie among models regarding *Accuracy* and/or *Macro F-Score*, *Weighted F-Score* is chosen as the discriminant criterion. The reason is that *Weighted F-Score* can be considered a compromise between *Macro F-Score* and *Accuracy*. If the tie still exists, the less complex model with the smaller number of hyperparameters is selected as per the *Occam's razor* principle [41]. In the last tie scenario, the model with shorter training time is chosen.

V. EXPERIMENTAL RESULTS AND DISCUSSION

A. RQ1: *Is our time-based cross-validation more effective than a non-temporal method to handle concept drift in the model selection step for vulnerability assessment?*

We performed both qualitative and quantitative analyses to demonstrate the relationship between *concept drift* and the model selection step of vulnerability assessment. Firstly, it is intuitive that the data of SVs intrinsically changes over time because of new products, software and attack vectors. The number of new terms appearing in the NVD description each year during the period from 2000 to 2017 is given in Fig. 4.

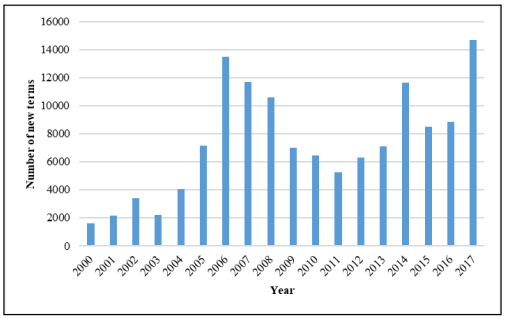

Fig. 4. The number of new terms from 2000 to 2017 of vulnerability description in NVD.

On average each year, there are 7345 new terms added to the vocabulary. Moreover, from 2015 to 2017, the number of new terms has been consistently increasing and achieved an all-time high value of 14684 in 2017. We also highlight some concrete examples about the terms appearing in the database after a particular technology, product or attack was released in Fig. 5. There seems to be a strong correlation between the time of appearance of some new terms in the descriptions and their years of release or discovery.

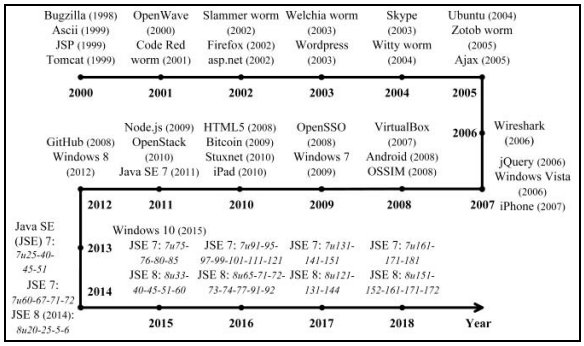

Fig. 5. Examples of new terms in NVD corresponding to new products, software, cyber-attacks from 2000 to 2018.
Note: The year of release/discovery is put in parentheses.

Such unseen terms contain many concepts about new products (e.g., *Firefox*, *Skype*, and *iPhone*), operating systems (e.g., *Android*, *Windows Vista/7/8/10*), technologies (e.g., *Ajax*, *jQuery*, and *Node.js*), attacks (e.g., *Code Red*, *Slammer*, and *Stuxnet* worms). There are also the extended forms of existing ones such as the updated versions of Java Standard Edition (Java SE) each year. These qualitative results depict that if the time property of SVs is not considered in the model selection step, then the future terms can be mixed with past ones. Such information leakage can result in the discrepancy in the real model performance.

In fact, the main goal of the validation step is to select the optimal models that can exhibit the similar behavior on unseen data. Next, our approach quantitatively compared the degree of model overfitting using our time-based cross-validation method with a stratified non-temporal one used in [6, 7]. For each method, we computed the *Weighted F-Scores* difference between the cross-validated and testing results of the optimal models found in the validation step (cf. Fig. 6). The model selection and selection criteria procedures of the normal cross-validation method are the same as our temporal one.

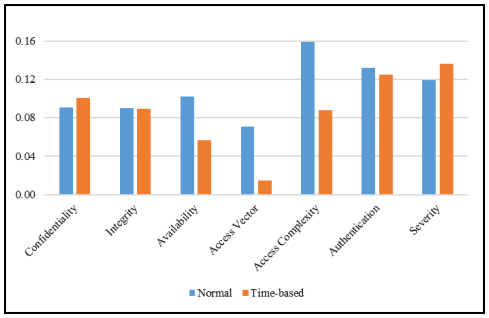

Fig. 6. Difference between the validated and testing *Weighted F-Scores* of our time-based and a normal cross-validation methods.

Fig. 6 shows that traditional non-temporal cross-validation was overfitted in four out of seven cases (i.e., Availability, Access Vector, Access Complexity, and Authentication). Especially, the degrees of overfitting of non-temporal validation method were 1.8, 4.7 and 1.8 times higher than those of the time-based version for Availability, Access Vector, and Access Complexity, respectively. For the other three VCs, both methods were similar, in which the differences were within 0.02. Moreover, on average, the *Weighted F-Scores* on the testing set of the non-temporal cross-validation method were only 0.002 higher than our approach. This value is negligible compared to the difference of 0.02 (ten times more) in the validation step. It is noted that similar comparison also holds for non-stratified non-temporal cross-validation. Overall, both qualitative and quantitative findings suggest that the time-based cross-validation method should be preferred to lower the performance overestimation and mis-selection of predictive models due to the effect of *concept drift* in the model selection step of SVs.

The summary answer to RQ1: The qualitative results show that many new terms are regularly added to NVD, after the release or discovery of the corresponding software products or cyber-attacks. Normal methods mixing these new terms can inflate the cross-validated model performance. Quantitatively, the optimal models found by our time-based cross-validation are also less overfitted, especially two to five times for Availability, Access Vector and Access Complexity. It is recommended that the time-based cross-validation should be adopted in the model selection step for vulnerability assessment.

B. **RQ2:** *Which are the optimal models for multi-classification of each vulnerability characteristic?*

The answer to RQ1 has shown that the temporal cross-validation should be used for selecting the optimal models in the context of vulnerability assessment. The work to answer RQ2 presents the detailed results of the first phase of our model. To be more specific, we have used our five-fold time-based cross-validation to select the optimal word-only model for each of the seven VCs from six classifiers (cf. section IV.C) and eight NLP representations (cf. Table 2). More specifically, we have followed the guidelines of the previous work [6] to extract only the words appearing in more than 0.1% of all descriptions as features for RQ2.

Firstly, each classifier was tuned using random VCs to select its optimal set of hyperparameters. Such selected hyperparameters are reported in Table 3.

TABLE 3. OPTIMAL HYPERPARAMETERS FOUND FOR EACH CLASSIFIER.

Classifier	Hyperparameters
NB	None
LR	Regularization value: + 0.1 for term frequency + 10 for tf-idf
SVM	Kernel: linear Regularization value: 0.1
RF	Number of trees: 100 Max. depth: unlimited Max. number of leaf nodes: unlimited
XGB	Number of trees: 100 Max. depth: unlimited Max. number of leaf nodes: 100
LGBM	Number of trees: 100 Max. depth: unlimited Max. number of leaf nodes: 100

It is worth noting that we have utilized local optimization as a filter to reduce the search space. We found that 0.1 was a consistently good value of regularization coefficient for SVM. Unlike SVM, for LR, 0.1 was suitable for term frequency representation; whereas, 10 performed better for the case of tf-idf. One possible explanation is that LR provides a decision boundary that is more sensitive to hyperparameter. Additionally, although tf-idf with l2-normalization helps model converge faster, it usually requires more regularization to avoid overfitting [42]. For ensemble models, more hyperparameters need tuning as mentioned in section IV.C. Regarding the maximum number of leaves, the optimal value for RF was *unlimited*, which is expected since it would give more flexibility to the model.

However, for XGB and LGBM, the *unlimited* value was not available. In fact, the higher value did not improve the performance, but significantly increased the computational time. As a result, we chose 100 to be the number of leaves for XGB and LGBM. Similarly, we obtained 100 as a good value for the number of trees of each ensemble model. We noticed that the maximum depth of ensemble methods was the hyperparameter that affected the validation result the most; the others did not change the performance dramatically. Finally, we got a search space of size of 336 in the cross-validation step ((six classifiers) x (eight NLP configurations) x (seven characteristics)). After using our five-fold time-based cross-validation method in section III.C, the optimal validation results are given in Table 4.

TABLE 4. OPTIMAL MODELS AND RESULTS AFTER THE VALIDATION STEP. NOTE: THE NLP CONFIGURATION NUMBER IS PUT IN PARENTHESES.

Vulnerability characteristic	Classifier (config)	*Accuracy*	*Macro F-Score*	*Weighted F-Score*
Confidentiality	LGBM (1)	0.839	0.831	0.840
Integrity	XGB (4)	0.861	0.853	0.861
Availability	LGBM (1)	0.785	0.783	0.782
Access Vector	XGB (7)	0.936	0.643	0.914
Access Complexity	LGBM (1)	0.771	0.553	0.758
Authentication	LR (3)	0.973	0.626	0.972
Severity	LGBM (5)	0.814	0.763	0.811

Besides each output, we also examined the validated results across different types of classifiers (single vs. ensemble models) and NLP representations (n-grams and tf-idf vs. term frequency).

TABLE 5. AVERAGE CROSS-VALIDATED *WEIGHTED F-SCORES* OF TERM FREQUENCY VS. TF-IDF GROUPED BY SIX CLASSIFIERS.

	Classifier					
	NB	**LR**	**SVM**	**RF**	**XGB**	**LGBM**
Term frequency	0.781	**0.833**	**0.835**	**0.843**	**0.846**	**0.846**
tf-idf	**0.786**	0.832	0.831	0.836	0.843	0.844

Since the NLP representations mostly affect the classifiers, their validated results are grouped by six classifiers in Table 5 and Table 6. The result shows that tf-idf did not outperform term frequency for five out of six classifiers. This result agrees with the existing work [6, 7]. It seemed that n-grams with $n > 1$ improved the result. We used a right-tailed unpaired two-sample t-test to check the significance of such improvement of n-grams ($n > 1$). The P-value was 0.169; that was larger than the confidence level of 0.05. Thus, we were unable to accept the improvement of n-grams over uni-gram. Furthermore, there was no performance improvement after increasing the number of n-grams. The above-reported three observations implied that the more complex NLP representations did not provide a statistically significant improvement over the simplest BoW (configuration one in Table 2). This argument helped explain why three out of seven optimal models in Table 4 were BoW.

TABLE 6. AVERAGE CROSS-VALIDATED *WEIGHTED F-SCORES* OF UNI-GRAM VS. N-GRAMS ($2 \le N \le 4$) GROUPED BY SIX CLASSIFIERS.

Classifier	1-gram	2-grams	3-grams	4-grams
NB	0.756	0.778	0.784	**0.785**
LR	0.821	0.835	**0.836**	**0.836**
SVM	0.823	0.835	0.836	**0.837**
RF	0.838	**0.840**	0.838	0.838
XGB	0.844	0.845	**0.846**	**0.846**
LGBM	**0.845**	**0.845**	**0.845**	**0.845**

Along with the NLP representations, we also investigated the performance difference between single (NB, LR, and SVM) and ensemble (RF, XGB, and LGBM) models. The average *Weighted F-Scores* grouped by VCs for single and ensemble models are illustrated in Fig. 7. Ensemble models seemed to consistently demonstrate the superior performance compared to single counterparts. It was also observed that the ensemble methods produced mostly consistent results (i.e., small variance) for Access Vector and Authentication properties.

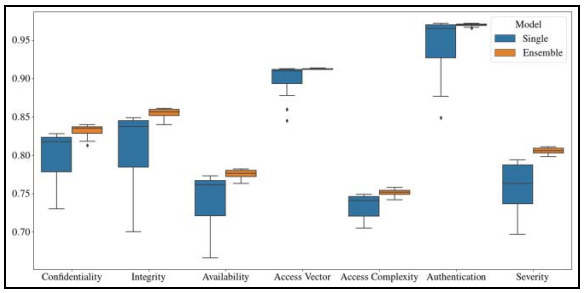

Fig. 7. Average cross-validated *Weighted F-Scores* comparison between ensemble and single models for each VC.

We performed the right-tailed unpaired two-sample *t*-tests to check the significance of the better performance of ensemble over single models. Table 7 reports the P-values of the results from the hypothesis testing.

TABLE 7. P-VALUES OF H_0: ENSEMBLE MODELS ≤ SINGLE MODELS FOR EACH VC.

Vulnerability characteristic	P-value
Confidentiality	3.261×10^{-5}
Integrity	9.719×10^{-5}
Availability	3.855×10^{-5}
Access Vector	2.320×10^{-3}
Access Complexity	1.430×10^{-5}
Authentication	1.670×10^{-3}
Severity	1.060×10^{-7}

The hypothesis testing confirmed that the superiority of the ensemble methods was significant since all P-values are smaller than the confidence level of 0.05. The validated results in Table 4 also affirmed that six out of seven optimal classifiers were ensemble (i.e., LGBM and XGB). It is noted that the XGB model usually takes more time to train than the LGBM model, especially for tf-idf representation. Our findings suggest that LGBM, XGB and BoW should be considered as baseline classifiers and NLP representations for future vulnerability-related research.

The summary answer to RQ2: LGBM and BoW are the most frequent optimal classifiers and NLP representations. Overall, the more complex NLP representations such as n-grams, tf-idf do not provide a statistically significant performance improvement than BoW. The ensemble models perform statistically better than single ones. It is recommended that the ensemble classifiers (e.g., XGB and LGBM) and BoW should be used as baseline models for vulnerability analytics.

C. RQ3: *How effective is our character-word model to perform automated vulnerability assessment with concept drift?*

The OoV terms presented in RQ1 actually directly have an impact on the word-only models. Such missing features can make the model unable to produce reliable results. Especially when no existing term is found (i.e., all features are zero), the model would have the same output regardless of the context. To answer RQ3, we first tried to identify such all-zero cases in the vulnerability descriptions from 2000 to 2018. For each year from 2000 to 2018, we split the dataset into (*i*) training set (data from the previous year backward) for building the vocabulary, and (*ii*) testing set (data from the current year to 2018) for checking the vocabulary existence. We found 64 cases from 2000 to 2018 in the testing data, in which all the features were missing (cf. Appendix). We used the terms appearing at least 0.1% in all descriptions. It should be noted that the number of all-zero cases may be reduced using a larger vocabulary with the trade-off for larger computational time. We also investigated the descriptions of these vulnerabilities and found several interesting patterns. The average length of these abnormal descriptions was only 7.98 words compared to 39.17 of all descriptions. It turned out that the information about the threats and sources of such SVs was limited. Most of them just included the assets and attack/vulnerability types. For example, the vulnerabilities with ID of CVE-2016-10001xx had nearly the same format "Reflected XSS in WordPress plugin" with the only differences were the name and version of the plugin. This format made the model hard to evaluate the impact of each vulnerability separately. Another issue was due to the specialized or abbreviated terms such as */redirect?url= XSS*, *SEGV*, *CSRF* without proper explanation. The above issues suggest that the vulnerability descriptions should be written with sufficient information to enhance the comprehensibility of SVs.

For RQ3, the solution to the issue of the word-only model using character level features is evaluated. We considered the non-stop-words with high frequency (i.e., appearing in more than 10% of all descriptions) to generate the character features. Using the same 0.1% value as RQ2 increased the dimensions more than 30 times, but the performance only changed within 0.02. According to **Algorithm 1**, the output minimum number of character n-grams was chosen to be two. We first tested the robustness of the character-only models by setting the maximum number of characters to only three. For each year y from 1999 to 2017, we used such character model to generate the characters from the data of the considering year y backward. We then verified the existence of such features using the descriptions of the other part of data (i.e., from year $y + 1$ towards 2018). Surprisingly, the model using only two-to-three-character n-grams could produce at least one non-zero feature for all the descriptions even using only training data in 1999 (i.e., the first year in our dataset based on the vulnerability identification). Such finding shows that our approach is stable to vulnerability data changes (*concept drift*) in testing data from 2000 to 2018 even with the limited amount of data and without retraining.

Next, to increase the generalizability of our approach, three to ten were considered for selecting the maximum number of character n-grams based on their corresponding vocabulary sizes (cf. Fig. 8). Using the elbow method in cluster analysis, six was selected since vocabulary size did not increase dramatically after this point. The selected minimum and maximum values of character n-grams turned out to match the minimum and average word lengths of all NVD descriptions in our dataset, respectively.

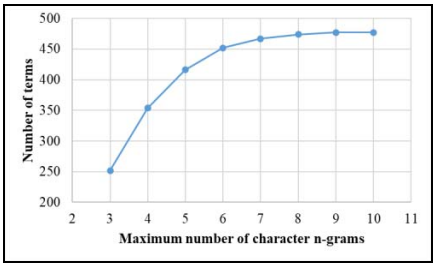

Fig. 8. The relationship between the size of vocabulary and the maximum number of character n-grams.

We then used the feature aggregation algorithm (cf. section III.D) to create the aggregated features from the character n-grams ($2 \leq n \leq 6$) and word n-grams to build the final model set and compared it with two baselines: Word-only Model (WoM) and Character-only Model (CoM). It should be noted that WoM is the model in which *concept drift* is not handled. Unfortunately, a direct comparison with the existing WoM [6] was not possible since they used older NVD dataset and they did not produce their source code for reproduction. However, we tried to set up the experiments based on the guidelines and results in the previous paper.

TABLE 8. PERFORMANCE (*ACCURACY*, *MACRO F-SCORE*, *WEIGHTED F-SCORE*) OF OUR CHARACTER-WORD, WORD-ONLY AND CHARACTER-ONLY MODELS.

Vulnerability chacteristic	Our optimal model (CWM)			Word-only model (WoM)			Character-only model (CoM)		
	Accuracy	*Macro F-Score*	*Weighted F-Score*	*Accuracy*	*Macro F -Score*	*Weighted F-Score*	*Accuracy*	*Macro F-Score*	*Weighted F-Score*
Confidentiality	**0.727**	**0.717**	**0.728**	0.722	0.708	0.723	0.694	0.683	0.698
Integrity	0.763	**0.749**	0.764	0.763	0.744	0.764	0.731	0.718	0.734
Availability	**0.712**	**0.711**	**0.711**	0.700	0.696	0.702	0.660	0.657	0.660
Access Vector	**0.914**	**0.540**	**0.901**	0.904	0.533	0.894	0.910	0.538	0.899
Access Complexity	0.703	0.468	0.673	**0.718**	**0.476**	**0.691**	0.700	0.457	0.668
Authentication	**0.875**	**0.442**	**0.844**	0.864	0.425	0.832	0.866	0.441	0.840
Severity	0.668	**0.575**	0.663	**0.686**	0.569	**0.675**	0.661	0.549	0.652

To be more specific, we used BoW predictors and random forest (the best of their three models used) with the following hyperparameters: the number of trees was 100 and the number of features for splitting was 40. For CoM, we used the same optimal classifiers of each VC. The comparison results are given in Table 8. CWM performed slightly better than the WoM for four out of seven VCs regarding all evaluation metrics. Also, 4.98% features of CWM were non-zero, which was nearly five-time denser than 1.03% of WoM. Also, CoM was the worst model among the three, which had been expected since it contained the least information (smallest number of features). Although CWM does not significantly outperform WoM, its main advantage is to effectively handle the OoV terms (*concept drift*), except new terms without any matching parts. We hope that our solution to *concept drift* will be integrated into the practitioner's existing framework and future research work to perform more robust SVs analytics.

The summary answer to RQ3: The WoM does not handle the new cases well, especially those with all zero-value features. Without retraining, the tri-gram character features can still handle the OoV words effectively with no all-zero features for all testing data from 2000 to 2018. Our CWM performs comparably well with the existing WoM and provides nearly five-time richer information. Hence, our CWM is better for automated vulnerability assessment with *concept drift*.

D. RQ4: *To what extent can low-dimensional model retain the original performance?*

The n-gram NLP models usually have an issue with the high-dimensional and sparse feature vectors [25]. The large feature sizes of our CWMs in Table 8 were 1649 for Confidentiality, Availability and Access Complexity; 4154 for Integrity and Access Vector; 3062 for Authentication; and 5104 for Severity. To address such challenge in RQ4, we investigated the dimensionality reduction method (i.e., Latent Semantic Analysis (LSA) [30]) and recent sub-word embeddings (e.g., fastText [31, 32]) for vulnerability classification. fastText is an extension of Word2Vec [43] word embeddings, in which the character-level features are also considered. fastText is different to traditional n-grams in the sense that it determines the meaning of a word/subword based on the surrounding context. Here, we computed the sentence representation as an average fastText embedding of its constituent words and characters. We implemented fastText using Gensim [44] library in Python.

For LSA, using the elbow method and total explained variances of the principal components, we selected 300 for the dimensions and called it LSA-300.

TABLE 9. *WEIGHTED F-SCORES* OF OUR ORIGINAL CWM (GREEN-COLORED BASELINE), 300-DIMENSION LATENT SEMANTIC ANALYSIS (LSA-300), FASTTEXT TRAINED ON VULNERABILITY DESCRIPTION (FASTTEXT-300) AND FASTTEXT TRAINED ON ENGLISH WIKIPEDIA PAGES (FASTTEXT-300W).

Vulnerability characteristic	Our CWM	LSA-300	fastText-300	fastText-300W
Confidentiality	**0.728**	0.656	**0.679**	0.648
Integrity	**0.764**	0.695	**0.719**	0.672
Availability	**0.711**	0.656	**0.687**	0.669
Access Vector	**0.901**	0.892	**0.893**	0.866
Access Complexity	**0.673**	0.611	**0.679**	0.678
Authentication	**0.844**	**0.842**	0.815	0.765
Severity	**0.663**	**0.656**	0.654	0.635

Table 9 shows that LSA-300 retained from 90% to 99% performance of the original model, but used only 300 dimensions (6-18% original model sizes). More remarkably, with the same 300 dimensions, the fastText model trained on the vulnerability descriptions was on average better than LSA-300 (97% vs. 94.5%). fastText model even slightly outperformed our original CWM for Access Complexity. Moreover, for all seven cases, the fastText model using vulnerability knowledge (fastText-300) had higher *Weighted F-Scores* than that trained on English Wikipedia pages (fastText-300W) [45]. The result implied that vulnerability descriptions contain specific terms that do not frequently appear in the general domains. The domain relevance turns out to be not only applicable to word embeddings [8], but also to character/sub-word embeddings for vulnerability analysis and assessment. Overall, our findings show that LSA and fastText are capable of building efficient models without too much performance trade-off.

The summary answer to RQ4: The LSA model with 300 dimensions (6-18% of the original size) retains from 90% up to 99% performance of the original model. With the same dimensions, the model with fastText sub-word embeddings provide even more promising results. The fastText model with the vulnerability knowledge outperforms that trained on the general context (e.g., Wikipedia). LSA and fastText can help build efficient models for vulnerability assessment.

VI. THREATS TO VALIDITY

Internal validity. We used well-known tools such as *scikit-learn* [26] and *nltk* [27] libraries for ML and NLP. Our optimal models may not guarantee the highest performance for every SV since there are infinite values of hyperparameters to tune. However, even when the optimal values change, a time-based cross-validation method should still be preferred since we have considered the general trend of all SVs. Our model may not provide the state-of-the-art results, but at least it gives the baseline performance for handling the *concept drift* of SVs.

External validity. Our work used NVD – one of the most comprehensive public repositories of SVs. The size of our dataset is more than 100,000 with the latest vulnerabilities in 2018. Our character-word model has also been demonstrated to consistently handle the OoV words well even with very limited data for all years in the dataset. It is recognized that the model may not work for extreme rare terms in which no existing parts can be found. However, our model is totally re-trainable to deal with such cases or incorporate more sources of SVs' descriptions.

Conclusion validity. We mitigated the randomness of the results by taking the average value of five-fold cross-validation. The performance comparison of different types of classifiers and NLP representations was also confirmed using a statistical hypothesis test with P-values that were much lower than the confidence level of 5%.

VII. RELATED WORKS

A. Software Vulnerability Analytics

It is important to patch the critical-first vulnerabilities [46]. Thus, besides CVSS, there have been many effective evaluation schemes for SVs [47-49]. Recently, there is a detailed Bayesian analysis of various vulnerability scoring systems [50], which highlights the good overall performance of CVSS. Therefore, we used the well-known CVSS as the ground truth for our approach. We assert that our approach can be generalizable to other vulnerability rating systems following the same scheme of multi-class classification.

Regarding the predictive analytics of SVs, Bozorgi [15] pioneered the use of ML models for SVs analysis. Their paper used an SVM model and various features (e.g., NVD description, CVSS, published and modified dates) to estimate the likelihood of exploitation and *time-to-exploit* of SVs. Another piece of work analyzed the VCs and trends of SVs by incorporating different vulnerability information from multiple vulnerability repositories [11, 14], security advisories [18, 51], darkweb/deepnet [14, 52] and social network (Twitter) [53]. These efforts assumed that all VCs have been available at the time of analysis. However, our work relaxes this assumption by using only the vulnerability description – one of the first information about new SVs. Our model can be used for both new and old SVs.

Actually, the descriptions are also utilized for vulnerability analysis and prediction. Yamamoto [17] used Linear Discriminant Analysis, Naïve Bayes and Latent Semantic Indexing combined with annual effect estimation to determine the VCs of more than 60,000 SVs in NVD. The annual effect focused on the recent SVs, but still could not explicitly handle the OoV terms in the descriptions. Spanos [6] worked on the same task using a multi-target framework with Decision Tree, Random Forest and Gradient Boosting Tree. Our approach also contains the word-only model, but we select the optimal models using our time-based cross-validation to better address the *concept drift* issue. The vulnerability descriptions were also used to evaluate the vulnerability severity [7], associate the frequent terms with each VC [16], determine the type of each SV using topic modeling [12] and show vulnerability trends [11]. Recently, Zhuobing [8] have applied deep learning to predict vulnerability severity. The existing literature has demonstrated the usefulness of description for vulnerability analysis and assessment, but has not mentioned how to overcome its *concept drift* challenge. Our work is the first of its kind to provide a robust treatment for SVs' *concept drift*.

B. Temporal modeling of software vulnerabilities

Regarding the temporal relationship of SVs, Roumani [54] proposed a time-series approach using autoregressive integrated moving average and exponential smoothing methods to predict the number of vulnerabilities in the future. Another time-series work [55] was described to model the trend in disclosing SVs. A group of researchers led by Tsokos published a series of work [56-58] on stochastic models such as Hidden Markov Models, Artificial Neural Network, and Support Vector Machine to estimate the occurrence and exploitability of vulnerabilities. The focus of the above studies was on the determination of the occurrence of SVs over time. In contrast, our work aims to handle the temporal relationship to build more robust predictive models for SVs assessment.

VIII. CONCLUSION AND FUTURE WORK

We observe that the existing works suffer from *concept drift* in the vulnerability descriptions that affect both the traditional model selection and prediction steps of SVs assessment. We assert that *concept drift* can degrade the robustness of existing predictive models. We show that the time-based cross-validation should be used for vulnerability analysis to better capture the temporal relationship of SVs. Then, our main contribution is the Character-Word Models (CWMs) to improve the robustness of automated SVs assessment with *concept drift*. CWMs have been demonstrated to handle *concept drift* of SVs effectively for all the testing data from 2000 to 2018 in NVD even in the case of data scarcity. Our approach has also performed comparably well with the existing word-only models. Our CWMs are also much less sparse and thus less prone to overfitting. We have also found that Latent Semantic Analysis and sub-word embeddings like fastText help build compact and efficient CWM models (up to 94% reduction in dimension) with the ability to retain at least 90% of the performance for all VCs. Besides the good performance, the implications on the use of different models are also given to support practitioners and researchers with vulnerability analytics. Hopefully, this work can open up various research avenues to develop more sophisticated *concept-drift*-aware models in SVs and related areas.

In the future, we plan to investigate the performance of deep learning models to embed the dependency of both character and word features in low-dimensional space for vulnerability prediction. Alongside *concept drift*, handling imbalanced data is also a concern for future SVs research.

APPENDIX

64 vulnerabilities (CVD-ID) with all-zero features of word-only model (V.C) from 2000 to 2018: CVE-2013-6647, CVE-2015-1000004, CVE-2016-1000113, CVE-2016-1000114, CVE-2016-1000117, CVE-2016-1000118, CVE-2016-1000126, CVE-2016-1000127, CVE-2016-1000128, CVE-2016-1000129, CVE-2016-1000130, CVE-2016-1000131, CVE-2016-1000132, CVE-2016-1000133, CVE-2016-1000134, CVE-2016-1000135, CVE-2016-1000136, CVE-2016-1000137, CVE-2016-1000138, CVE-2016-1000139, CVE-2016-1000140, CVE-2016-1000141, CVE-2016-1000142, CVE-2016-1000143, CVE-2016-1000144, CVE-2016-1000145, CVE-2016-1000146, CVE-2016-1000147, CVE-2016-1000148, CVE-2016-1000149, CVE-2016-1000150, CVE-2016-1000151, CVE-2016-1000152, CVE-2016-1000153, CVE-2016-1000154, CVE-2016-1000155, CVE-2016-1000217, CVE-2017-10798, CVE-2017-10801, CVE-2017-14036, CVE-2017-14536, CVE-2017-15808, CVE-2017-16760, CVE-2017-16785, CVE-2017-17499, CVE-2017-17703, CVE-2017-17774, CVE-2017-6102, CVE-2017-7276, CVE-2017-8783, CVE-2018-10030, CVE-2018-10031, CVE-2018-10382, CVE-2018-11120, CVE-2018-11405, CVE-2018-12501, CVE-2018-13997, CVE-2018-14382, CVE-2018-5285, CVE-2018-5361, CVE-2018-6467, CVE-2018-6834, CVE-2018-8817, CVE-2018-9130

REFERENCES

[1] M. Corporation. *Common Vulnerabilities and Exposures* [Online]. Available: https://cve.mitre.org/ [Accessed: 25/12/2018].

[2] *CEA Report: The Cost of Malicious Cyber Activity to the U.S. Economy* [Online]. Available: https://www.whitehouse.gov/articles/cea-report-cost-malicious-cyber-activity-u-s-economy/ [Accessed: 25/12/2018].

[3] K. Nayak, D. Marino, P. Efstathopoulos, and T. Dumitraş, "Some Vulnerabilities Are Different Than Others," Cham, 2014, pp. 426-446.

[4] S. Khan and S. Parkinson, "Review into State of the Art of Vulnerability Assessment using Artificial Intelligence," in *Guide to Vulnerability Analysis for Computer Networks and Systems: An Artificial Intelligence Approach*, S. Parkinson, A. Crampton, and R. Hill, Eds., ed Cham: Springer International Publishing, 2018, pp. 3-32.

[5] V. Smyth, "Software vulnerability management: how intelligence helps reduce the risk," *Network Security,* vol. 2017, pp. 10-12, 2017.

[6] G. Spanos and L. Angelis, "A multi-target approach to estimate software vulnerability characteristics and severity scores," *Journal of Systems and Software,* vol. 146, pp. 152-166, 2018.

[7] G. Spanos, L. Angelis, and D. Toloudis, "Assessment of Vulnerability Severity using Text Mining," in *21st Pan-Hellenic Conference on Informatics*, Larissa, Greece, 2017, pp. 1-6.

[8] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, "Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 125-136.

[9] *National Vulnerability Database* [Online]. Available: https://nvd.nist.gov/ [Accessed: 25/12/2018].

[10] M. A. Williams, S. Dey, R. C. Barranco, S. M. Naim, M. S. Hossain, and M. Akbar, "Analyzing Evolving Trends of Vulnerabilities in National Vulnerability Database," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 3011-3020.

[11] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and A. B. Bener, "Mining trends and patterns of software vulnerabilities," *Journal of Systems and Software,* vol. 117, pp. 218-228, 2016.

[12] S. Neuhaus and T. Zimmermann, "Security Trend Analysis with CVE Topic Models," in *2010 IEEE 21st International Symposium on Software Reliability Engineering*, 2010, pp. 111-120.

[13] B. L. Bullough, A. K. Yanchenko, C. L. Smith, and J. R. Zipkin, "Predicting Exploitation of Disclosed Software Vulnerabilities Using Open-source Data," in *3rd ACM on International Workshop on Security And Privacy Analytics*, Scottsdale, Arizona, USA, 2017, pp. 45-53.

[14] M. Almukaynizi, E. Nunes, K. Dharaiya, M. Senguttuvan, J. Shakarian, and P. Shakarian, "Patch Before Exploited: An Approach to Identify Targeted Software Vulnerabilities," in *AI in Cybersecurity*, L. F. Sikos, Ed., ed Cham: Springer International Publishing, 2019, pp. 81-113.

[15] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, USA, 2010, pp. 105-114.

[16] D. Toloudis, G. Spanos, and L. Angelis, "Associating the Severity of Vulnerabilities with their Description," Cham, 2016, pp. 231-242.

[17] Y. Yamamoto, D. Miyamoto, and M. Nakayama, "Text-Mining Approach for Estimating Vulnerability Score," in *2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, 2015, pp. 67-73.

[18] M. Edkrantz, S. Truvé, and A. Said, "Predicting Vulnerability Exploits in the Wild," in *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, 2015, pp. 513-514.

[19] M.-T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba, "Addressing the rare word problem in neural machine translation," *arXiv preprint arXiv:1410.8206,* 2014.

[20] C.-C. Huang, H.-C. Yen, P.-C. Yang, S.-T. Huang, and J. S. Chang, "Using sublexical translations to handle the OOV problem in machine translation," *ACM Transactions on Asian Language Information Processing (TALIP),* vol. 10, p. 16, 2011.

[21] A. Liu and K. Kirchhoff, "Context Models for OOV Word Translation in Low-Resource Languages," *arXiv preprint arXiv:1801.08660,* 2018.

[22] M. Razmara, M. Siahbani, R. Haffari, and A. Sarkar, "Graph propagation for paraphrasing out-of-vocabulary words in statistical machine translation," in *51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2013, pp. 1105-1115.

[23] FIRST. *Common Vulnerability Scoring System v3.0: Specification Document* [Online]. Available: https://www.first.org/cvss/specification-document [Accessed: 25/12/2018].

[24] FIRST. *Common Vulnerability Scoring System* [Online]. Available: https://www.first.org/cvss/ [Accessed: 25/12/2018].

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

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[27] S. Bird and E. Loper, "NLTK: the natural language toolkit," in *ACL 2004 on Interactive Poster and Demonstration Sessions*, 2004, p. 31.

[28] M. F. Porter, "An algorithm for suffix stripping," *Program,* vol. 14, pp. 130-137, 1980.

[29] C. Bergmeir and J. M. Benítez, "On the use of cross-validation for time series predictor evaluation," *Information Sciences,* vol. 191, pp. 192-213, 2012.

[30] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science,* vol. 41, pp. 391-407, 1990.

[31] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606,* 2016.

[32] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759,* 2016.

[33] *Kaggle* [Online]. Available: https://www.kaggle.com/ [Accessed: 25/12/2018].

[34] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*: Malaysia; Pearson Education Limited, 2016.

[35] S. H. Walker and D. B. Duncan, "Estimation of the probability of an event as a function of several independent variables," *Biometrika,* vol. 54, pp. 167-179, 1967.

[36] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning,* vol. 20, pp. 273-297, 1995.

[37] A. Basu, C. Walters, and M. Shepherd, "Support vector machines for text categorization," in *2003 36th Annual Hawaii International Conference on System Sciences*, 2003, p. 7 pp.

[38] T. K. Ho, "Random decision forests," in *1995 3rd International Conference on Document Analysis and Recognition*, 1995, pp. 278-282.

[39] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785-794.

[40] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017, pp. 3146-3154.

[41] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Information Processing Letters,* vol. 24, pp. 377-380, 1987.

[42] T. H. M. Le, T. T. Tran, and L. K. Huynh, "Identification of hindered internal rotational mode for complex chemical species: A data mining approach with multivariate logistic regression model," *Chemometrics and Intelligent Laboratory Systems,* vol. 172, pp. 10-16, 2018.

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

[44] R. Rehurek and P. Sojka, "Software framework for topic modelling with large corpora," in *LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010.

[45] *Pre-trained word vectors of fastText* [Online]. Available: https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md [Accessed: 25/12/2018].

[46] Y. Hou, X. Ren, Y. Hao, T. Mo, and W. Li, "A Security Vulnerability Threat Classification Method," in *Advances on Broad-Band Wireless Computing, Communication and Applications*, Cham, 2018, pp. 414-426.

[47] Q. Liu and Y. Zhang, "VRSS: A new system for rating and scoring vulnerabilities," *Computer Communications,* vol. 34, pp. 264-273, 2011.

[48] G. Spanos, A. Sioziou, and L. Angelis, "WIVSS: a new methodology for scoring information systems vulnerabilities," in *17th Panhellenic Conference on Informatics*, Thessaloniki, Greece, 2013, pp. 83-90.

[49] R. Sharma and R. K. Singh, "An Improved Scoring System for Software Vulnerability Prioritization," in *Quality, IT and Business Operations: Modeling and Optimization*, P. K. Kapur, U. Kumar, and A. K. Verma, Eds., ed Singapore: Springer Singapore, 2018, pp. 33-43.

[50] P. Johnson, R. Lagerstrom, M. Ekstedt, and U. Franke, "Can the Common Vulnerability Scoring System be Trusted? A Bayesian Analysis," *IEEE Transactions on Dependable and Secure Computing,* pp. 1-1, 2018.

[51] C.-C. Huang, F.-Y. Lin, F. Y.-S. Lin, and Y. S. Sun, "A novel approach to evaluate software vulnerability prioritization," *Journal of Systems and Software,* vol. 86, pp. 2822-2840, 2013.

[52] E. Nunes, A. Diab, A. Gunn, E. Marin, V. Mishra, V. Paliath*, et al.*, "Darknet and deepnet mining for proactive cybersecurity threat intelligence," in *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, 2016, pp. 7-12.

[53] C. Sabottke, O. Suciu, and T. Dumitras, "Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits," in *USENIX Security Symposium*, 2015, pp. 1041-1056.

[54] Y. Roumani, J. K. Nwankpa, and Y. F. Roumani, "Time series modeling of vulnerabilities," *Computers & Security,* vol. 51, pp. 32-40, 2015.

[55] M. Tang, M. Alazab, and Y. Luo, "Exploiting Vulnerability Disclosures: Statistical Framework and Case Study," in *2016 Cybersecurity and Cyberforensics Conference (CCC)*, 2016, pp. 117-122.

[56] S. M. Rajasooriya, C. P. Tsokos, and P. K. Kaluarachchi, "Cyber Security: Nonlinear Stochastic Models for Predicting the Exploitability," *Journal of Information Security,* vol. 8, p. 125, 2017.

[57] P. K. Kaluarachchi, C. P. Tsokos, and S. M. Rajasooriya, "Non-Homogeneous Stochastic Model for Cyber Security Predictions," *Journal of Information Security,* vol. 9, p. 12, 2017.

[58] N. R. Pokhrel, H. Rodrigo, and C. P. Tsokos, "Cybersecurity: Time Series Predictive Modeling of Vulnerabilities of Desktop Operating System Using Linear and Non-Linear Approach," *Journal of Information Security,* vol. 8, p. 362, 2017.