Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 238 (2024) 618–622

Procedia Computer Science

www.elsevier.com/locate/procedia

The 7th International Conference on Emerging Data and Industry (EDI40), April 23-25, 2024, Hasselt, Belgium

Triage Software Update Impact via Release Notes Classification

Solomon Berhe[a,*], Vanessa Kan[a], Omhier Khan[a], Nathan Pader[a], Ali Zain Farooqui[a], Marc Maynard[b], Foutse Khomh[c]

[a] *University of the Pacific, Stockton, CA 95211, USA*
[b] *Data Independence LLC, Ellington CT 06029, USA*
[c] *Polytechnique Montréal, Montréal, QC, H3C 3A7, Canada*

Abstract

In the rapidly evolving domain of Industry 4.0, effective management of software updates is crucial for maintaining system continuity and security. This paper presents a novel machine learning-based approach for a prompt and effective triage of software updates, leveraging an evaluation of six release note classifiers to categorize updates by component type, release type, and security risk. Our methodology, tested on a dataset of 1,000 release notes commonly encountered in Industry 4.0 ecosystems, demonstrates Logistic Regression as the most accurate classifier. The findings not only highlight the practical applicability of our approach in real-world data but also set the foundation for future enhancements to streamline the machine learning triage process further.

Keywords: Triage, Software, Update, Release Notes, Classifier, Evaluation

1. Introduction

Software ecosystems today comprise independent software teams that release and install updates autonomously, and these updates wield significant impact over the Software Engineering (SWE) process. The challenge lies in promptly assessing their impact, given the unpredictable nature of update behavior. Triage, the process of evaluating a software update impact, presents difficult challenges, including factors such as the frequency of daily updates, unclear technical release notes description, varied update practices, and as a result, the potential for cascading impact. This challenge is particularly common in Industry 4.0 ecosystems, where a lot of inter-dependent software is updated independently by multiple stakeholders, including product teams and actual customers[1]. For example, in safety-critical ecosystems like healthcare, prompt software update triage is essential to ensure patient safety [2, 3]. Previous research, including our own, primarily relied on graph-based impact analysis, leveraging version dates, numbers, and

* Corresponding author. Tel.: +1-772-202-3743 ;
E-mail address: sberhe@pacific.edu

1877-0509
10.1016/j.procs.2024.06.069

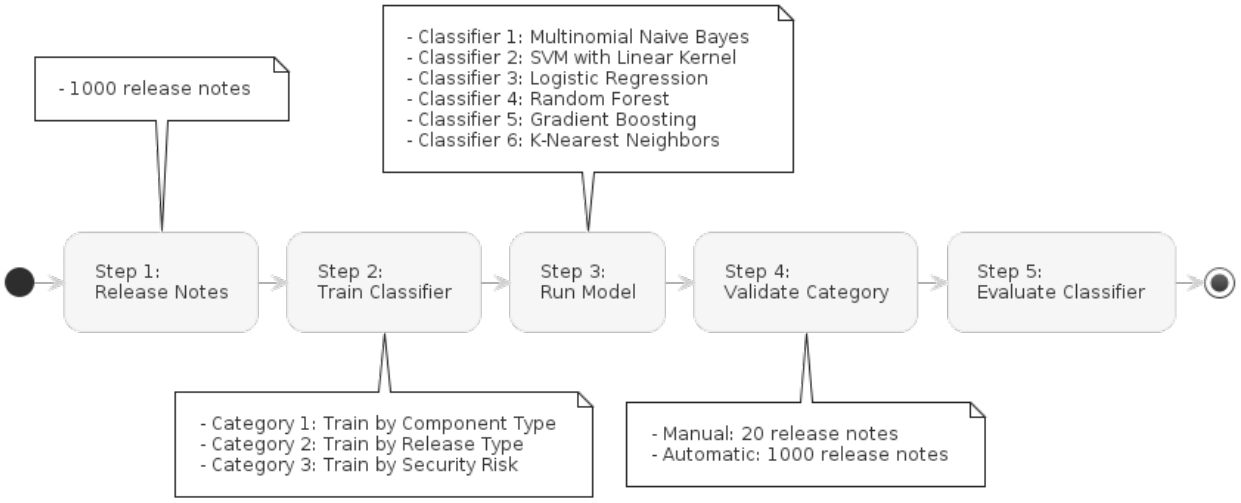

Fig. 1. Release Note Classification Process

description tags [4, 5]. However, this approach has many limitations. For example, it often overlooks dependencies, fails to strictly adhere to semantic versioning principles, and relies on manual triage, as crucial details are often embedded in release notes details. To address these limitations, our proposed work evaluates the classification of release notes to categories, with the aim to support prompt and a more automatic triage process. Towards that objective, we will evaluate the following three common software application ecosystem impact-critical category types:

- Classification based on software release note component type, allows for fast triage of high-impact component updates. This is particularly critical in Industry 4.0 ecosystems with central server components interfacing many adjacent dependent components. [6].
- Classification based on software release type, with a focus on major updates that necessitate rapid assessment due to their expected breaking impact. This is particularly important, since not all release notes adhere to semantic versioning and the major breaking update information is often included in the release notes details [7].
- Classification based on software security risk severity presented by the update, enabling the fast triage of zero-day updates to mitigate potential zero-day attacks. This is particularly important, since zero-day updates are often released as patch versions and unless the release notes details are manually reviewed, the significance of the impact of the patch update is often unknown [8].

To illustrate this approach, Figure 1 illustrates a systematic process for training and evaluating classifiers on a set of 1,000 release notes. The process begins with the release notes being fed into the training phase where six different classifiers, namely: Multinomial Naive Bayes, Support Vector Machine (SVM) with Linear Kernel, Logistic Regression, Random Forest, Gradient Boosting, and K-Nearest Neighbors—are trained across three distinct categories: Component Type, Release Type, and Security Risk. Once trained, each classifier model is run and its category predictions are validated, with a subset of 20 random release notes undergoing manual expert validation and the remainder being validated automatically. Finally, the performance of each classifier is evaluated, completing the classification workflow. Our primary focus is on an unexplored area in software update classification, with an explicit emphasis on high impact category classifier. The goal is to determine the suitability of classifiers for different impact categories. The following sections will address the related work, experimental setup, result analysis, and concluding remarks.

2. Related Work

Recent research has focused on utilizing metadata from software release notes to address various challenges in software engineering. Abebe et al. in [9] utilize machine learning to classify software release notes, analyzing 85

notes from 15 different systems to enhance understanding of their informational content. This approach is similar to Bi et al. in [10], who conduct a case study involving an empirical analysis of 32,425 release notes from 1,000 GitHub projects, aiming to augment release note details.

Several studies, including those by Nath et al. [11], Moreno et al. [12], Cseker et al. [13], and Khalfallah et al. [14], focus on generating comprehensive release notes. In support of this, Ma et al. [15] propose an automatic classification of software artifacts, while Linares et al. [16] apply machine learning for domain categorization of software applications, an approach that could extend our work into safety relevant domains like health care.

Araujo et al. in [17] apply machine learning for decision support in planning future releases, similar to our work in aiding stakeholders' decision-making. However, our research diverges by concentrating on the impact of existing software updates rather than future releases. Alsolai et al. in [18] provide a systematic literature review on software maintenance, underscoring the growing impact of release notes in software ecosystem engineering.

Contrasting these approaches, our work specifically aims to apply and evaluate classifiers to concisely summarize release notes, particularly focusing on their impact to support a prompt triage. The objective is to support the rapid triage of release notes based on their impact, a critical task for maintaining the operations of Industry 4.0 ecosystems.

3. Data and Experimental Setup

For data collection (see Figure 1, Step 1), we randomly selected 1000 release notes from the year 2023, sourced from a pool of 55,232 release notes spanning 10369 components and 20 repositories, including GitHub, Mitre, and Wikipedia. This ensured the variety of our data set, including recent software component updates, various software release types, security-related patch updates, as well as open source and closed source software updates. Below are three examples of release notes, illustrating an update to a central server component, a major disruptive update, and a security patch for a browser.

1. *"Next generation frontend tooling. It's fast! vite, hmr, frontend, build-tool, dev-server, vite, production, 20230103, 4.0.4*
2. *"Expressive middleware for node.js using ES2017 async functions koa, koa, beta, 20230102, 3.0.0, node.js"*
3. *"Inappropriate implementation in HTML parser in Google Chrome prior to 99.0.4844.51 allowed a remote attacker to bypass XSS preventions via a crafted HTML page. (Chrome security severity: Medium) mitre cve, cve, 20230102, 99.0.4844, xss, bypass, html, chrome"*

To arrive at this standard release notes format, all differently supported properties (brand, license, tags, timestamp, version, channel, etc.) from different repositories were merged into a single description field. For researchers seeking to replicate our entire process, we have provided a repository on GitHub[1]. This repository contains the requisite data, software, and documentation enabling a reproduction of our analyses and results.

For the second step (see Figure 1, Step 2) we proceeded to train the release notes of all classifiers. The classifier were trained to categorize release notes based on three high impact dimensions: component type, release type, and security risk.

4. Results and Analysis

In this section, we present an analysis of our methodology, encompassing three critical stages: running the model, validating the categories, and evaluating the classifier performance (see Figure 1, Steps 3-5). Firstly, the 'Run Model' stage involved applying our trained classifiers to the entire data set, where each classifier predicted categories based on the training data. The subsequent 'Validate Category' phase entailed an assessment of the predictions. This step combined manual and automated validation. Finally, in the 'Evaluate Classifier' stage, we conducted a evaluation of each classifier. This evaluation was anchored on key performance metrics such as accuracy and precision, providing an understanding of each classifier's efficacy in categorizing the release notes. This overall classification process, from model execution to validation and evaluation, is instrumental in demonstrating the effectiveness and applicability of our classifiers in real-world scenarios.

[1] GitHub repository link: https://github.com/SEI40E/ReleaseNotesClassification

	Component Type Category	Release Type Category	Security Risk Category
Multinomial Naive Bayes	772 (72%)	Incorrect	992 (92%)
SVM with Linear Kernel	771 (77%)	Incorrect	949 (94%)
Logistic Regression	889 (88%)	Incorrect	997 (99%)
Random Forest	776 (77%)	Incorrect	949 (94%)
Gradient Boosting	708 (70%)	Incorrect	944 (94%)
K-Nearest Neighbors	583 (58%)	Incorrect	811 (81%)

Table 1. Results of Classifier by Category Combinations of 1000 Release Notes

4.1. Run Model

In this study, when running the models across 1,000 release notes, the classification process for each note, encompassing all three category types, was performant, averaging approximately one second per release note. It underscores the practicality of the model in real-world applications, where rapid categorization aligns well with the aim of a prompt triage.

4.2. Validate Category

Initially, a manual expert review and iterative training and validation process was established for the first random 20 release notes to ensure accuracy in the classifier's categorization. The most complex release note involved a cryptocurrency trading API.

```
{
    "A JavaScript / Python / PHP cryptocurrency trading API with support
    for more than 100 bitcoin/altcoin exchanges altcoin, api, arbitrage,
    bitcoin, bot, cryptocurrency, crypto, e-commerce, ethereum, library,
    strategy, trading, btc, eth, trade, merchant, ccxt, production, 20230115,
    2.6.19, python, javascript, php"
}
```

It necessitates the creation of a custom 'bitcoin' category due to its coverage of multiple component types, including programming languages and APIs. Following this, a hybrid validation method, combining partial manual and full automatic processes, was applied to the data set of 1,000 release notes, aiming for a relative majority correctness in classifications. This approach was replicated across different category types. However, challenges were observed in achieving majority accuracy across all classifiers, particularly in the initial set of 20 notes for the release note type category. For the security risk category, the specific labels achieved a clear categorization, resulting in a majority consensus on the correct categorization.

4.3. Evaluate Classifier

In the classification of software component types, applying a relative majority category vote yields accurate classifier results. In particular, Logistic Regression showed the most accuracy. As illustrated in Table 1, Logistic Regression excelled in classifying software release notes due to its adeptness in handling binary (security risk) and multi-class (component type) classification [19]. However, future work will focus on refining the classifier training for high impact component types to enhance precision. Classifying software release types presents a challenge, particularly with labels like [".0.0", "major", "breaking", "dependency"]. The difficulty lies in categorically defining non-major updates, which often leads to either overfitting or underfitting in the classification process. Regarding security risks, the labels are specific, including terms like ["mitre", "cve", "bitdefender", "certificate", "xss", "untrusted", "attacker", "vulnerable", "vulnerability", "firewall", "encryption", "OWASP", "escalation", "privilege"]. This specificity significantly improves the accuracy of all classifiers offering release notes in this domain to promptly triage zero-day patch updates.

This overall promising result allows exploring combinations of categories, such as updates that simultaneously fall into high impact component types and security risks. Future work will further improve accuracy by enriching software release notes with additional contextual data. For instance, incorporating classifications from the release

notes repository for certain categories could provide more precision. Overall, the results, particularly in the areas outside of release type classification, are promising and offer an opportunity for future research directions in this field.

5. Conclusion

In this work we introduce a method to promptly triage software updates in Industry 4.0 ecosystems, focusing on the impact evaluation of updates using machine learning to classify 1,000 release notes across component type, release type, and security risk. Utilizing classifiers like Logistic Regression, the study demonstrates effective categorization of updates, incorporating a comprehensive evaluation process that combines manual iterative training and validation with an automated approach. This research offers a structured methodology for a prompt triage of software updates, crucial for complex ecosystems. Future work will focus on the development of an algorithm that integrates the output from predicted categories with larger data sets derived from our previous graph-based impact assessment model. This integrated approach with the objective to further improve the triage process. Lastly, we plan to validate the algorithm's effectiveness within a real Industry 4.0 ecosystem environment.

6. Acknowledgement

We would like to thank NIST NVD, GitHub, Linux Distrowatch Feed, and Wikipedia for providing public access to the software update release meta data.

References

[1] Mugarza, I., Flores, J. L., & Montero, J. L. (2020). Security Issues and Software Updates Management in the Industrial Internet of Things (IIoT) Era. *Sensors*, 20(24), 7160.
[2] Scquizzato, T., Landoni, G., Semeraro, F., & Zangrillo, A. (2019). Smartphone software update could potentially affect the efficiency of lay first-responders networks in out-of-hospital cardiac arrests. *Resuscitation*, 144, 15-16.
[3] Tervonen, L. (2019). Efficient Distribution of Software Updates - A Case Study in Healthcare.
[4] Berhe, S., Maynard, M., & Khomh, F. (2020). Software Release Patterns: When Is It a Good Time to Update a Software Component?. *Procedia Computer Science*, 170, 618-625.
[5] Berhe, S., Maynard, M., & Khomh, F. (2023). Maintenance Cost of Software Ecosystem Updates. *Procedia Computer Science*, 220, 608-615.
[6] Muccini, H., & Moghaddam, M. T. (2018). IoT architectural styles: A systematic mapping study. In *Software Architecture: 12th European Conference on Software Architecture, ECSA 2018, Madrid, Spain, September 24–28, 2018, Proceedings* (Vol. 12, pp. xxx-xxx). Springer International Publishing.
[7] Lam, P., Dietrich, J., & Pearce, D. J. (2020). Putting the semantics into semantic versioning. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (pp. xxx-xxx).
[8] Sawadogo, D. D. A. (2022). Towards Overcoming Zero-Day Vulnerabilities in Open Source Software: An Automatic Approach for Security Patches Identification.
[9] Abebe, S. L., Ali, N., & Hassan, A. E. (2016). An Empirical Study of Software Release Notes. *Empirical Software Engineering*, 21, 1107-1142.
[10] Bi, T., Xia, X., Lo, D., Grundy, J., & Zimmermann, T. (2020). An Empirical Study of Release Note Production and Usage in Practice. *IEEE Transactions on Software Engineering*, 48(6), 1834-1852.
[11] Nath, S. S., & Roy, B. (2021). Automatically Generating Release Notes with Content Classification Models. *International Journal of Software Engineering and Knowledge Engineering*, 31(11n12), 1721-1740.
[12] Moreno, L., Bavota, G., Di Penta, M., Oliveto, R., Marcus, A., & Canfora, G. (2014). Automatic Generation of Release Notes. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 484-495).
[13] Çeker, A., Yeşilyurt, S., Ardahan, İ. C., & Çınar, B. (2021). Prediction of Development Types from Release Notes for Automatic Versioning of OSS Projects. In *The International Conference on Artificial Intelligence and Applied Mathematics in Engineering* (pp. 399-407). Springer.
[14] Khalfallah, M. (2019). Generation and Visualization of Release.
[15] Ma, Y., Fakhoury, S., Christensen, M., Arnaoudova, V., Zogaan, W., & Mirakhorli, M. (2018). Automatic Classification of Software Artifacts in Open-Source Applications. In *Proceedings of the 15th International Conference on Mining Software Repositories* (pp. 414-425).
[16] Linares-Vásquez, M., McMillan, C., Poshyvanyk, D., & Grechanik, M. (2014). On Using Machine Learning to Automatically Classify Software Applications into Domain Categories. *Empirical Software Engineering*, 19, 582-618. Springer.
[17] Araújo, A. A., Paixao, M., Yeltsin, I., Dantas, A., & Souza, J. (2017). An Architecture Based on Interactive Optimization and Machine Learning Applied to the Next Release Problem. *Automated Software Engineering*, 24, 623-671. Springer.
[18] Alsolai, H., & Roper, M. (2020). A Systematic Literature Review of Machine Learning Techniques for Software Maintainability Prediction. *Information and Software Technology*, 119, 106214. Elsevier.
[19] Harrell, F. E. (2015). Binary Logistic Regression. In *Regression Modeling Strategies* (Springer Series in Statistics). Springer, Cham.