Machine Learning for the Identification and Classification of Technical Debt Types on StackOverflow Discussions

Eliakim Gama
Universidade Estadual do Ceará
Fortaleza, Ceará, Brasil
eliakim.gama@aluno.uece.br

Mariela I. Cortés
Universidade Estadual do Ceará
Fortaleza, Ceará, Brasil
mariela.cortes@uece.br

Matheus Paixao
Universidade Estadual do Ceará
Fortaleza, Ceará, Brasil
matheus.paixao@uece.br

Adson Damasceno
Universidade Estadual do Ceará
Fortaleza, Ceará, Brasil
adson.damasceno@uece.br

ABSTRACT

In today's fast-paced software industry, understanding and managing Technical Debt (TD) is crucial for software development. TD can compromise the long-term quality of software systems. The occurrence of TD is commonly reported and discussed by practitioners on Question and Answers (Q&A) platforms, such as Stack Overflow (SO). Data from Q&A platforms has been leveraged by the TD research community, most prominently regarding knowledge extraction. However, manual analyses of such data not only require considerable effort but also suffer from biases. Hence, this paper aims to propose an automated approach for identifying and classifying types of TD in SO discussions using machine learning (ML) and natural language processing. We divided our methodology into four main steps: i) data preprocessing, ii) application of natural language processing, iii) application of ML algorithms, and iv) computing the evaluation metrics for the proposed models. Our results indicate that ML algorithms have the potential to be successfully applied to automatically identify and classify TD types on SO discussions. We achieved a recall of 85% for test debt and a precision of 78% for design debt. Furthermore, the results of automated TD identification on SO benefit the software development community by enhancing solution quality, raising awareness of best practices, and facilitating collaboration among developers. This leads to more efficient development and the promotion of consistent standards. We make our entire dataset and pre-trained models available to encourage future research directions.

CCS CONCEPTS

• **General and reference** → **Empirical studies**; • **Software and its engineering** → **Maintaining software**.

KEYWORDS

Technical Debt, Classification, Stack Overflow, Machine Learning

Reference Format:
Eliakim Gama, Mariela I. Cortés, Matheus Paixao, and Adson Damasceno. 2023. Machine Learning for the Identification and Classification of Technical Debt Types on StackOverflow Discussions. In *Brazilian Workshop on Intelligent Software Engineering (ISE'23), September, 2023, Campo Grande, Brazil.* Campo Grande - Brazil, 6 pages. https://doi.org/10.5753/ise.2023.235840

https://doi.org/10.5753/ise.2023.235840

1 INTRODUCTION

The choice of shortcuts during software development can generate Technical Debt (TD), which needs to be managed to keep its accumulation under control [8] . Nowadays, TD has been divided into several types, each of which refer to a certain debt that can be accumulated during different phases of the software development lifecycle. Common types of TD are code debt, infrastructure debt, architectural debt and testing debt. Recent studies have aimed to acquire knowledge and provide insights regarding TD by identifying TD elements and classifying their types [15, 21, 28, 31]. Managing TD is essential for the long-term health of software systems. However, many organizations lack established TD management practices. Project managers and developers need tools and methods to strategically address TD, which can be challenging due to the complexity of software development practices and the diverse data involved [2].

The quest for practical knowledge about TD has driven investigations to Question and Answer platforms (Q&A). Through both quantitative and qualitative approaches, several studies [6, 12, 13, 19, 33] have investigated Q&A platforms to identify elements of TD, most of them through manual analysis. For instance, Gama *et al.* [12] analysed discussions on SO to understand how practitioners debate issues regarding TD. They performed a manual analysis of 140 discussions. Despite being a widely used technique in scientific work, manual analysis tends to be labor-intensive and can introduce bias in result interpretation. It requires the involved methodology to be well-defined and followed rigorously to ensure the reliability of the process. Nevertheless, even when used following the best possible practices, manual analysis does not scale to large amounts of data. Without a strategy to automatically identify and filter TD-related data from large text collections, the research contributions will be limited.

Hence, this paper aims to propose an automated approach for the identification and classification of TD types in SO discussions through ML and natural language processing. By leveraging labelled data from previous studies, we trained different ML models to not only identify an instance of TD discussion but also the type of TD being discussed. We considered a thorough experimental setup to identify the best performing model.

The main contributions of this work are listed as follows:

- Providing pre-trained models to automatically identify and classify TD types in OS discussions.
- An analysis of the words most likely to refer to different types of TD.

We believe that our contribution has the potential to boost the research involving knowledge extraction regarding TD on SO and Q&A platforms in general. Future work can leverage our models to filter TD-related data from large text collections.

The remainder of this paper is organized as follows: Section 2 provides the background, Section 3 discusses related works, Section 4 describes the experimental setup, Section 5 presents the results, Section 6 addresses implications for future research, and Section 7 concludes the paper.

2 BACKGROUND

In order to substantiate the concepts applied to this work, in this section, the fundamental components that provide the structure for this research are presented.

2.1 Technical Debt

TD describes the long-term consequences of shortcuts taken during the software development process to achieve short-term goals [8]. Throughout the software's lifecycle, TD may occur in different artefacts depending on when it is incurred and which activity it is associated with. Considering these aspects, TD is classified into the following types [31]: design, code, architecture, tests, documentation, defects, infrastructure, requirements, people, build, process, automated tests, usability, service, and versioning.

2.2 Stack Overflow

SO is a Q&A platform that has gained prominence among computing professionals for facilitating discussions primarily related to the software development phase. Questions can be answered by multiple users, leading to discussions regarding the topic. Considering SO users [35], the discussions held on the platform not only represent a vast community of practitioners but also contain valuable information for advancing knowledge in software engineering and computer science as a whole [3]. The data generated through SO's questions and answers has been utilised in various areas such as microservices [4], automated documentation enhancement [36], IDE (Integrated Development Environment) improvement [25], and mobile development [32], to mention only a few [1].

2.3 Natural Language Processing

Natural Language Processing (NLP) is related to the development of computational models for performing tasks that rely on information expressed in natural languages [7]. This branch of artificial intelligence is divided into three main aspects: Sound (related to morphology), Structure (related to morphology and syntax), and Meaning (related to semantics and pragmatics). NLP techniques can be leveraged to enhance ML classifiers by providing efficient data preprocessing techniques and vectorisation strategies, for instance.

2.4 Predictive Analysis and ML

Predictive analysis is a type of analysis performed on large databases that involves extracting information from data to forecast trends and behaviour patterns. The aim of this technique is to determine the potential future outcome of an event or even the probability of a condition occurring [26].

Predictive analysis encompasses ML algorithms, which include predictive classification and regression models, including algorithms such as *Random Forest, XGBoosting, Gradient Boosting, Multinomial*, and SVC. These models have been selected due to their extensive usage within the software engineering and ML community [5, 14, 16, 18, 23, 24, 34, 37, 38]. Furthermore, the selected algorithms exhibit diverse profiles, ranging from algorithms that use a randomly extracted subset of attributes to algorithms that combine models to enhance effectiveness. A brief description of these algorithms is shown below.

- *Random Forest* algorithm is a classification algorithm that employs decision trees to perform data mining on a given dataset. This algorithm creates multiple decision trees using a randomly extracted subset of attributes from the original dataset. It has been used for prediction of student learning effectiveness in software engineering teamwork [24].
- *Gradient Boosting* is a ML model formed through the combination of weak models, such as shallow decision trees [17]. In comparison to *Random Forest*, this algorithm becomes more robust against overfitting. It has been used in software engineering for test code reuse considering object-oriented parameters [34].
- *XGBoosting* is an enhanced, faster, and better-performing version of *Gradient Boosting* [14]. This algorithm can was applied in the identification of blocking bugs in software development [5].
- MultinomialNB is an algorithm commonly used for text classification [16, 18]. This multi-class classifier produces accurate results within a short time frame, utilising a minimal percentage of training data. This algorithm was used to predict the severity of defect reports in software maintenance [38].
- The SVC algorithm (C-Support Vector Classification) is implemented based on SVM (Support Vector Machine). The concept behind this algorithm is to draw a hyperplane that separates the two sets with a margin [23]. SVC has been used to predict software quality [37].

3 RELATED WORK

The work more related to the one reported in this paper is the one by Kozanidis *et al.* (2022) [19], where the authors applied NLP and ML techniques to 415 SO discussions about TD. The study aimed to deepen the understanding of TD by focusing on types of TD, question duration, perceived urgency and sentiments. However, the models considered within the evaluation did not achieve reliable values, with an average precision, recall, and F1-Score of around 50%. Such results invalidate the usage of the models in subsequent research efforts in the topic.

A combination of NLP and static analysis for early detection of TD was used in Rantala's work (2020) [29]. The authors aimed to understand TD from developers' perspectives by discovering themes and topics in messages related to TD. The work also proposed a tool for automatic detection of TD using only NLP, but the study's data is not related to TD discussions on SO.

Maldonado *et al.* (2017) [9] presented an approach to automatically identify SATD in requirements and design using NLP. The authors achieved accurate identification of SATD, obtaining 90% performance in classification using only 23% of comments related to requirements debt and 80% using only 9% and 5% of comments for design and requirements, respectively. In spite of the good results, the study contemplated only two types of TD.

With the goal of identifying SATD through NLP, Ren *et al.* (2019) [30] identified features in source code comments within software projects using neural networks. The application of neural networks revealed patterns within the comments that had not been identified through human analysis. Unlike our study, Ren *et al.* in addition to using source code comments as a data source, also applied NLP through neural networks.

Table 1 provides an overview of the main differences and similarities among the related works.

Table 1: Comparison between related work and this paper

Reference	Approach	Data Source	Related Words?
This work	Automated	Stack Overflow	Yes
Kozanidis et al. [19]	Manual and Automated	Stack Overflow	Yes
Rantala et al. [29]	Automated	Source code	No
Maldonado et al. [9]	Automated	Source code	Partially yes
Ren et al.[30]	Automated	Source code	Yes

4 EXPERIMENTAL SETUP

This section describes the experimental setup that we employed to carry on our study. The setup is divided into the following stages: data collection, data preprocessing and ML execution. The entire dataset used in this article along with the project containing the models are available in our replication package [11].

4.1 Data Collection

The database used in this work stems from the manual analysis performed by Gama et al. [10], where the authors performed a manual analysis of 372 discussions from SO. For each discussion, the authors labelled elements such as the *type of TD*, *TD management activity*, and *indicators*. SO discussions consist of a title, a question, tags, and answers. Each of these are separate text components that belong to the same discussion. To build the dataset for this study, we had to first filter some discussions and then split the discussions into parts. This process is detailed next.

4.1.1 Phase 1: Filtering discussions without a TD type. For this study, we want to not only identify the presence of TD in the text but also classify the type of TD being discussed. Hence, for all the 372 originally labeled discussions, we filtered all discussions in which no TD type was identified by the authors. After this phase, 320 discussions remained.

4.1.2 Phase 2: Filtering discussions with low-frequency TD types. The original labelled dataset includes 13 different TD types. However, some of these types of debt have low frequency in the discussions where they were found. For instance, while code debt was identified in 170 discussions, defect debt was identified in only 4 discussions. Through a pilot experiment, we observed that TD types with an incidence of fewer than 10 discussions yielded poor results. Hence, only TD types that appeared in more than 10 discussions were considered. After this phase, the dataset contained 301 discussions related to five TD types: code, infrastructure, architecture, testing, and design, as detailed in Table 2.

Table 2: TD types considered in this study

Debt Type	Number of Discussions	Considering All Components
Code	170	1287
Infrastructure	56	410
Architecture	42	256
Testing	21	224
Design	12	78

4.1.3 Phase 3: Splitting the discussion's components. In this study, we chose to split the discussion's components into individual records. Hence, each discussion was split into title, question, tags and answers. The TD type labelled for the discussion was considered for its individual components. The final dataset reached a total of 2,255 records. Our dataset is comprised of 3 columns (Id, Category, and Text), where Id represents the identification of the discussion on the SO platform, Category indicates the TD type found in the discussion, and Text represents a title, question, tags, or response belonging to the discussion.

4.2 Data Preprocessing

Initially, the text was subjected to **tokenization**, a process that employs algorithms to split sentences into words. This helps with analysis as each word can be evaluated individually. Following tokenization, the **removal of stopwords** was carried out. This step involves eliminating words from the text that do not significantly contribute to its interpretation. SO discussions may contain alphanumeric characters within their content, therefore, it was necessary to apply the **removal of alphanumeric characters.** When transformed into tokens, these characters can make it difficult the identification of patterns within the data. Lastly, **removal of uppercase characters** was performed. To enhance the effectiveness of converting categorical and numerical data, words with the same spelling must be uniform. Consequently, tokens were transformed to lowercase to ensure consistent representation.

4.3 Machine Learning Execution

As this is an initial study, we chose simple algorithms that have have been used in previous studies. The ML algorithms were implemented using the *scikit-learn* library[1]. The selected algorithms are: Random Forest, Gradient Boosting, XGBoosting, MultinomialNB and SVC. The results for each algorithm are evaluated through standard evaluation metrics for ML models: Precision, Recall, and F1-Score.

The data split between training and testing in this work follows the standard percentage used by the data analysis community [20],

[1] *scikit-learn*: https://scikit-learn.org/stable/

which is 70% for training and 30% for testing. All tokens were transformed into a vector space, converting categorical data into numerical data using TF-IDF (Term Frequency – Inverse Document Frequency) [27]. Discussions with labelled TD type were separated, and all other discussions were labelled as "not", indicating that the label was different from the specified TD.

Considering that the quantity of discussions for each TD type can vary, we balanced the dataset separately. Data balancing involves randomly selecting a "not" labelled set with the same number of records as the TD type. Figure 1 provides an example of the balancing performed for discussions categorised as infrastructure-related. When analysing the entire dataset, one can observe that there are 410 records categorised as infrastructure debt and 1845 records categorised as "not infrastructure debt". As a balancing alternative, a subset containing 410 discussions was randomly chosen from the "not infrastructure debt" set, ensuring that both sets have the same size.

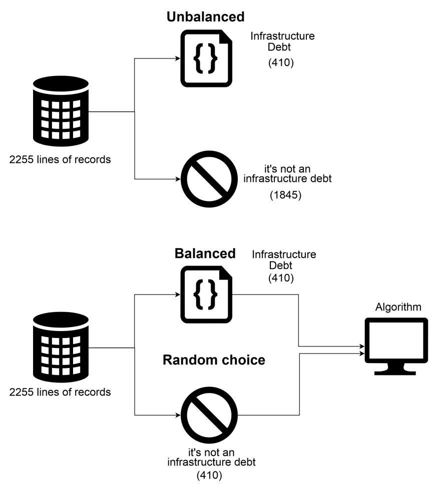

Figure 1: Example of discussion balancing for the experiment

5 RESULTS

In the following subsections, we present the results obtained in our experiment. First, we present the results for different TD types. Next, we present an analysis of the most important terms used to identify and classify each TD type.

5.1 Results for different TD types

Table 3 presents the results found for each different TD type, we display the results achieved by the best performing ML algorithm for the TD type.

The results reveal that the ability to automatically identify TD types varies based on the TD type being considered. While we observed a F1-Score of 77% for Tests debt, we also observed a F1-Score of 60% for code debt. This indicates that some TD types are easier to identify than others. For the other TD types, such as Design, Architecture and Infrastructure, we observed F1-Scores of 66%, 61% and 60%, respectively. Overall, considering the best performing algorithms for each TD type, we achieved Precision, Recall and F1-Score of 66%, 67% and 66%, respectively.

Table 3: Results by each TD type

TD Type	Precision	Recall	F1-Score
Tests	71%	85%	77%
Design	78%	61%	68%
Architecture	63%	70%	66%
Infrastructure	63%	58%	61%
Code	57%	63%	60%
Average	66%	67%	66%

Table 4 presents a comparison between the results found in this study and their corresponding values in Kozanidis *et al.*'s work. Through this comparison, it becomes evident that the results in this study achieved higher percentages in all metrics (Precision, Recall, and F1-Score). In certain cases, such as the Recall and F1-Score for code debt, the results achieved by our models are two-fold better than the ones reported reported in the related work.

Upon observing the results, it is possible to identify that the highest percentages concerning the evaluation metrics are associated with the less frequently occurring debt types within the discussions. In Kozanidis *et al.*'s work (2022), the debt type with the highest average percentage was *Build* debt, with an average of 68.3%. This debt type had the fewest occurrences, being found in only 10 discussions. A similar phenomenon occurs in the results of this study, where testing debt is the second least occurring type (see Table 2), and it is the type of debt that achieved the highest percentage concerning the evaluation metrics.

Table 4: Comparison of results between this study and Kozanidis *et al.* (2022)

TD Type	Precision		Recall		F1-Score	
	This Study	Kozanidis et al (2022)	This Study	Kozanidis et al (2022)	This Study	Kozanidis et al (2022)
Tests	71%	58%	85%	75%	77%	66%
Design	78%	36%	61%	36%	68%	36%
Architecture	63%	54%	70%	33%	66%	41%
Infrastructure	63%	42%	58%	55%	61%	48%
Code	57%	28%	63%	23%	60%	25%

5.2 Analysis of Words Related to TD Types

In addition to the results regarding the ML algorithms performance in classifying TD types, we also performed an analysis of the words that are more related to certain TD types. Our preprocessing steps facilitated the generation of word clouds for each TD type. Due to space constraints, only the word cloud pertaining to "design" debt is presented in this paper, while the remaining content is available in our replication package.

The word cloud related to "design" debt is presented in Figure 2. In the figure, the size of each word is proportional to its frequency, meaning that larger words are more frequently mentioned in the text. In this case, words like: "use", "project", "need", "war", "system", "architecture", "time", "will", "work", "spring", and "code" were more prominent, that is, they are common words in discussions of design TD in SO.

Figure 2: Word Cloud for Design Debt

Figure 3 presents the Sankey diagram [22] containing the synthesis of the top 40 most frequent words in discussions related to the TD types investigated in this study. In this work, the diagram was adapted to represent the relationship of each word to its corresponding TD type, where the lines depict the link between the TD type and the associated word. The thickness of the line indicates the strength of the relationship.

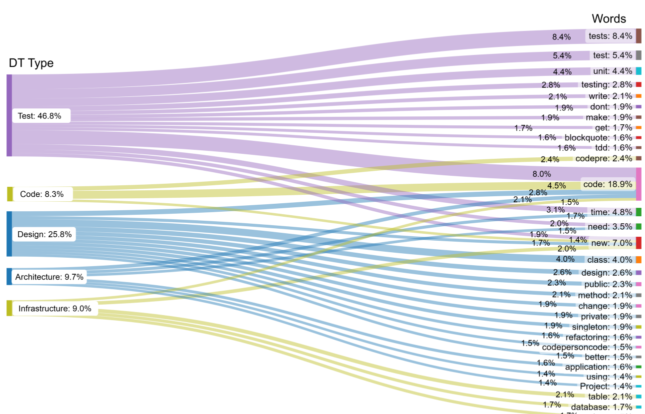

Figure 3: Sankey diagram for different TD types

From Figure 3, it can be observed that test debt has words with the highest percentage by TD type, such as *tests*, *unit*, and *testing*, where the total percentage of testing debt reaches 46.8%. Concerning design debt, the most frequent words within this TD are *code*, *class*, and *design*, with a total percentage of relevant words in design debt being 25.8%. The Architecture, Infrastructure, and Code debt types have a total word percentage of 9.7%, 9%, and 8.3%, respectively.

6 IMPLICATIONS

Implications are typically two-pronged: implications for research or theory and implications for practice. In this section, we discuss the contribution of our findings for researchers and practitioners.

6.1 For Researchers

Researchers can use our study results to support efforts in the field of TD research in software engineering. The proposed approach for automated analysis can serve as a basis for adaptations and improvements in the process. In addition, the approach can contribute to the agility of the analysis process, enabling the examination of large volumes of data that would be unfeasible through manual analysis.

The tests carried out with the proposed algorithms facilitated the identification of the best algorithm for each type of debt. Therefore, community efforts to determine the optimal ranking algorithm for a specific type of debt can be reduced, mitigating the need of evaluating several algorithms.

6.2 For Practitioners

Industry professionals can benefit from our study's findings when considering implementing TD-related practices in their software engineering projects. The approach can be applied to streamline the TD identification and approach process, making it possible to effectively manage technical debt in real-world projects and increasing the efficiency of software development and maintenance processes.

In addition, professionals can take advantage of our study results to optimize their strategies to deal with specific types of TD, since our tests identified the best algorithm for each type of debt. This can lead to more effective TD management and reduced development costs. Finally, practitioners can explore relationships between words associated with different types of TD and refine their quality criteria using the these words, which can help improve the overall quality and maintainability of the software.

7 CONCLUSIONS

In this paper, we proposed an automated approach for identifying and classifying different types of TD in SO discussions using ML and natural language processing. The approach enabled the identification of code, infrastructure, architecture, testing, and design TD types, achieving results ranging from 60% to 77% in F1-Score, these values outweigh the results of the baseline work results. The ML algorithms performance were higher than those in related studies with similar goals. In addition, we generated word clouds for TD type, revealing the words most frequently associated with each specific TD. These words can serve as input for searching for specific TD types in other data sources.

The study's limitations include potential issues with construct validity due to limited metrics, biases in internal validity during data preprocessing, and a risk of subjectivity in result analysis, as it was conducted by a single researcher. Additionally, generalizing conclusions may be challenging beyond our considered dataset.

As future work, we intend to experiment the proposed models in other datasets not yet used for training automated TD classifiers. In addition, we plan to leverage the most recurrent words for each TD type to assist in the combined automated classification with the proposed algorithms. We also plan to expand our dataset by combining data from other studies that performed manual analysis and apply the methodology proposed in this work to include the remaining TD types in addition to subjecting the experiment to deep learning methods and advanced vectorization techniques.

ACKNOWLEDGMENTS

This work was conducted with the support of the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Funding Code 001.

REFERENCES

[1] Arshad Ahmad, Chong Feng, Shi Ge, and Abdallah Yousif. 2018. A survey on mining stack overflow: question and answering (Q&A) community. *Data Technologies and Applications* 52, 2 (2018), 190–247.

[2] Danyllo Albuquerque, Everton Guimarães, Graziela Tonin, Pilar Rodríguez, Mirko Barbosa Perkusich, Hyggo O. Almeida, Angelo Perkusich, and Ferdinandy Chagas. 2023. Managing Technical Debt Using Intelligent Techniques - A Systematic Mapping Study. *IEEE Trans. Software Eng.* 49, 4 (2023), 2202–2220. https://doi.org/10.1109/TSE.2022.3214764

[3] Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, and Jure Leskovec. 2012. Discovering value from community activity on focused question answering sites. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12*. ACM Press.

[4] Alan Bandeira, Carlos Alberto Medeiros, Matheus Paixao, and Paulo Henrique Maia. 2019. We Need to Talk about Microservices: an Analysis from the Discussions on StackOverflow. *International Conference on Mining Software Repositories* 5 (2019), 255–259.

[5] Xiaoyun Cheng, Naming Liu, Lin Guo, Zhou Xu, and Tao Zhang. 2020. Blocking Bug Prediction Based on XGBoost with Enhanced Features. In *44th IEEE Annual Computers, Software, and Applications Conference, COMPSAC 2020, Madrid, Spain, July 13-17, 2020*. IEEE, 902–911. https://doi.org/10.1109/COMPSAC48688.2020.0-152

[6] Diego Costa, Mariela Inés Cortés, and Eliakim Gama. 2021. On the relation between technical debt indicators and quality criteria in Stack Overflow discussions. In *SBES '21: 35th Brazilian Symposium on Software Engineering, Joinville, Santa Catarina, Brazil, 27 September 2021 - 1 October 2021*. ACM, 432–441.

[7] Michael A. Covington, Ken Barker, and Stan Szpakowicz. 1994. Natural Language Processing for Prolog Programmers.

[8] Ward Cunningham. 1992. The WyCash Portfolio Management System. In *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*. 29–30.

[9] Everton da S. Maldonado, Emad Shihab, and Nikolaos Tsantalis. 2017. Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt. *IEEE Trans. Software Eng.* 43, 11 (2017), 1044–1062.

[10] Eliakim Gama. 2021. Investigando como os profissionais do Stack Overflow discutem sobre dívida técnica e a sua identificação em projetos de software. In *Dissertação (Mestrado Acadêmico em Ciências da Computação) - Programa De Pós-Graduação Em Ciência Da Computação - Universidade Estadual do Ceará*.

[11] Eliakim Gama, Mariela I. Cortes, Matheus Paixao, and Adson Damasceno. 2023. Replication package for the paper: "Machine Learning for the Identification and Classification of Technical Debt Types on StackOverflow Discussions". https://zenodo.org/record/8331281

[12] Eliakim Gama, Sávio Freire, Manoel Mendonça, Rodrigo O. Spínola, Matheus Paixao, and Mariela I. Cortés. 2020. Using Stack Overflow to Assess Technical Debt Identification on Software Projects. In *Proceedings of the 34th Brazilian Symposium on Software Engineering (SBES '20)*. Association for Computing Machinery, New York, NY, USA, 730–739.

[13] Eliakim Gama, Matheus Paixao, Emmanuel Sávio Silva Freire, and Mariela Inés Cortés. 2019. Technical Debt's State of Practice on Stack Overflow: A Preliminary Study. In *Proceedings of the XVIII Brazilian Symposium on Software Quality (SBQS'19)*. Association for Computing Machinery, New York, NY, USA, 228–233.

[14] Jiankun Ge, Linfeng Zhao, Zihui Yu, Huanhuan Liu, Lei Zhang, Xuewen Gong, and Huaiwei Sun. 2022. Prediction of Greenhouse Tomato Crop Evapotranspiration Using XGBoost Machine Learning Model. *Plants* 11, 15 (2022).

[15] M. Firdaus Harun and Horst Lichter. 2015. Towards a Technical Debt Management Framework based on Cost-Benefit Analysis. In *Proceedings of The Tenth International Conference on Software Engineering Advances*. 70–73.

[16] Khalil M.El Hindi, Reem R.Aljulaidan, and HussienAlSalman. 2020. Lazy fine-tuning algorithms for naïve Bayesian text classification. In *Journal Elsevier*.

[17] Ibrahim Karabayir, Samuel M. Goldman, Suguna Pappu, and Oguz Akbilgic. 2020. Gradient boosting for Parkinson's disease diagnosis from voice recordings. In *MC Medical Informatics and Decision Making, vol. 20*.

[18] Anshu Khurana and Om Prakash Verma. 2020. Novel approach with nature-inspired and ensemble techniques for optimal text classification. In *Multimed Tools Appl 79*.

[19] Nicholas Kozanidis, Roberto Verdecchia, and Emitzá Guzmán. 2022. Asking about Technical Debt Characteristics and Automatic Identification of Technical Debt Questions on Stack Overflow. In *International Symposium on Empirical Software Engineering and Measurement (ESEM) 2022*.

[20] Max Kuhn and Kjell Johnson. 2013. Applied predictive modeling. http://www.amazon.com/Applied-Predictive-Modeling-Max-Kuhn/dp/1461468485/

[21] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A Systematic Mapping Study on Technical Debt and Its Management. *J. Syst. Softw.* 101, C (March 2015).

[22] R.C. Lupton and J.M. Allwood. 2017. Hybrid Sankey diagrams: Visual analysis of multidimensional data for understanding resource use. *Resources, Conservation and Recycling* 124 (2017), 141–151. https://doi.org/10.1016/j.resconrec.2017.05.002

[23] Amit Kumar Nandanwar and Jaytrilok Choudhary. 2021. Semantic Features with Contextual Knowledge-Based Web Page Categorization Using the GloVe Model and Stacked BiLSTM. *Symmetry* 13, 10 (2021). https://www.mdpi.com/2073-8994/13/10/1772

[24] Dragutin Petkovic, Marc Sosnick-Pérez, Kazunori Okada, Rainer Todtenhoefer, Shihong Huang, Nidhi Miglani, and Arthur Vigil. 2016. Using the random forest classifier to assess and predict student learning of Software Engineering Teamwork. In *2016 IEEE Frontiers in Education Conference, FIE 2015, Eire, PA, USA, October 12-15, 2016*. IEEE Computer Society, 1–7.

[25] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Mining StackOverflow to turn the IDE into a self-confident programming prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 102–111.

[26] S. Poornima and M. Pushpalatha. 2018. A survey of predictive analytics using big data with data mining.

[27] Anand Rajaraman and Jeffrey David Ullman. 2011. Data Mining. In *Cambridge University Press*.

[28] Narayan Ramasubbu and Chris Kemerer. 2018. Integrating Technical Debt Management and Software Quality Management Processes: A Framework and Field Tests. In *Proceedings of the 40th International Conference on Software Engineering*.

[29] Leevi Rantala. 2020. Towards better technical debt detection with NLP and machine learning methods. In *ICSE '20: 42nd International Conference on Software Engineering, Companion Volume, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 242–245.

[30] Xiaoxue Ren, Zhenchang Xing, Xin Xia, David Lo, Xinyu Wang, and John Grundy and. 2019. Neural Network-based Detection of Self-Admitted Technical Debt: From Performance to Explainability. In *ACM Transactions on Software Engineering and Methodology*.

[31] Nicolli Rios, Manoel Gomes de Mendonça Neto, and Rodrigo Oliveira Spínola. 2018. A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information and Software Technology* 102 (2018), 117 – 145.

[32] Christoffer Rosen and Emad Shihab. 2016. What are mobile developers asking about? A large scale study using stack overflow. *Empirical Software Engineering* 21, 3 (jun 2016), 1192–1223.

[33] Riccardo Rubei, Claudio Di Sipio, Phuong Thanh Nguyen, Juri Di Rocco, and Davide Di Ruscio. 2020. PostFinder: Mining Stack Overflow posts to support software developers. *Inf. Softw. Technol.* 127 (2020), 106367.

[34] Amandeep Kaur Sandhu and Ranbir Singh Batth. 2021. Software reuse analytics using integrated random forest and gradient boosting machine learning algorithm. *Softw. Pract. Exp.* 51, 4 (2021), 735–747. https://doi.org/10.1002/spe.2921

[35] StackOverflow. [n.d.]. Stack Overflow Annual Developer Survey. https://insights.stackoverflow.com/survey/ Acessado: 2019-07-23.

[36] Christoph Treude and Martin P Robillard. 2016. Augmenting api documentation with insights from stack overflow. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 392–403.

[37] Fei Xing, Ping Guo, and Michael R. Lyu. 2005. A Novel Method for Early Software Quality Prediction Based on Support Vector Machine. In *16th International Symposium on Software Reliability Engineering (ISSRE 2005), 8-11 November 2005, Chicago, IL, USA*. IEEE Computer Society, 213–222.

[38] Cheng-Zen Yang, Chun-Chi Hou, Wei-Chen Kao, and Ing-Xiang Chen. 2012. An Empirical Study on Improving Severity Prediction of Defect Reports Using Feature Selection. In *19th Asia-Pacific Software Engineering Conference, APSEC 2012, Hong Kong, China, December 4-7, 2012*, Karl R. P. H. Leung and Pornsiri Muenchaisri (Eds.). IEEE, 240–249.