Applying Machine Learning in Technical Debt Management: Future Opportunities and Challenges

Angeliki-Agathi Tsintzira[1], Elvira-Maria Arvanitou[1], Apostolos Ampatzoglou[1] and Alexander Chatzigeorgiou[1]

[1] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
angeliki.agathi.tsintzira@gmail.com, e.arvanitou@uom.edu.gr, a.ampatzoglou@uom.edu.gr, achat@uom.gr

Abstract. Technical Debt Management (TDM) is a fast-growing field that in the last years has attracted the attention of both academia and industry. TDM is a complex process, in the sense that it relies on multiple and heterogeneous data sources (e.g., source code, feature requests, bugs, developers' activity, etc.), which cannot be straightforwardly synthesized; leading the community to using mostly qualitative empirical methods. However, empirical studies that involve expert judgement are inherently biased, compared to automated or semi-automated approaches. To overcome this limitation, the broader (not TDM) software engineering community has started to employ machine learning (ML) technologies. Our goal is to investigate the opportunity of applying ML technologies for TDM, through a Systematic Literature Review (SLR) on the application of ML to software engineering problems (since ML applications on TDM are limited). Thus, we have performed a broader scope study, i.e., on machine learning for software engineering, and then synthesize the results so as to achieve our high-level goal (i.e., possible application of ML in TDM). Therefore, we have conducted a literature review, by browsing the research corpus published in five high-quality SE journals, with the goal of cataloging: (a) the software engineering practices in which ML is used; (b) the machine learning technologies that are used for solving them; and (c) the intersection of the two: developing a problem-solution mapping. The results are useful to both academics and industry, since the former can identify possible gaps, and interesting future research directions, whereas the latter can obtain benefits by adopting ML technologies.

Keywords: Machine learning, software quality, literature review, technical debt, technical debt management

1. Introduction

Software quality is a multidisciplinary topic, in the sense that quality is about: (a) how well software meets users' needs, (b) how well software conforms to its specifications from the developers' point of view, (c) how well inherent, structural characteristics of the software are achieved from the product point of view, and (d) how much the end-user is willing to pay for it from the value point of view [20]. In recent years, the structural view of software quality is discussed through a metaphor, termed Technical Debt

(TD), which valuates poor software quality and the incurred maintainability problems [21]. Technical Debt Management (TDM) refers to all activities that can be performed for guaranteeing the efficient handling of TD, e.g., identifying, measuring, prioritizing, repaying, etc. A significant portion of TDM research is nowadays performed through qualitative empirical studies. However, inherently qualitative studies are subject to bias, in the sense that they heavily rely on expert judgement.
To alleviate such subjectivity, in traditional software quality research, researchers are nowadays exploiting the large amount of data that are available through software repositories. Such data enable researchers to perform large-scale quantitative studies, and adopt modern techniques, such as machine learning to effectively carry out a specific task without relying on explicit instructions or rules. For example, supervised machine learning techniques have been used to build models that can predict the number of defects in software systems. Based on the aforementioned applicability of ML technologies, we believe that there is an opportunity to apply ML in technical debt management. Nevertheless, to the best of our knowledge in the current TDM state-of-the-art there are limited studies that propose the use of ML explicitly for TDM (e.g., [6] [26]). Despite the fact that for some constituents of TD, e.g., code smell detection or change proneness assessment, some unsupervised or supervised ML approaches have been applied (e.g., [10]) these studies do not focus on the financial perspective of TD (e.g., economics of code smells, refactorings, changes), but only on the technical view of the phenomenon. The goal of this study is to investigate how ML can be applied for TDM, by studying existing literature. Since, the state-of-the-art lacks a substantial amount of studies, we conducted a broader secondary study, i.e., on how machine learning approaches have been used in software engineering (SE) practices, by conducting a systematic literature review (SLR). Next, we interpret these findings in the context of TDM. We note that the nature of this study is exploratory, in the sense that it aims at providing a panorama of the intersection of the two fields (ML and SE), without going into details. For instance, we do not aim to provide trend analysis, or explore the benefits obtained by the use of ML (this would require an explanatory research setting). The reasons for this decision is the fact that ML and SE are quite broad and a single study would not be able to cover both goals: therefore we believe that an exploratory study is first required so as to setup the research scene. Thus, the main outcome of this study is the provision of:

c1: The current ***status*** of research on combining ML and software engineering. In particular, we investigate which software engineering practices are approached through ML technologies.

c2: The ***opportunities*** of applying ML in TDM. To achieve this goal, we map software engineering practices, in which ML has already been applied, to TDM activities and concepts.

c3: The ***challenges*** for the adoption of ML in TDM research.

Section 2 presents related work (i.e., secondary studies on ML and SE) and background concepts of TDM. Next, Section 3 provides the literature review protocol, whereas, Section 4 presents the results of the study. Section 5 discusses the status, opportunities and challenges of applying ML to TDM research, whereas Section 6 displays threats to validity. Finally, Section 7 concludes the paper, and provides the implications to researchers and software development industry.

2. Related Work and Background Information

Related Work. In the literature we have been able to identify only one secondary study that summarizes the use of machine learning in software engineering. In particular, Zhang et al. [28] have surveyed the literature to identify the most commonly used ML technologies that have been applied in software engineering, and provide some guidelines on how to perform ML in software engineering. The main differences of this study compared to ours are: (a) we use a more systematic approach for obtaining and analyzing studies—i.e., a survey instead of a SLR; and (b) that our study is mapping the obtained results in the context of TDM. In addition to that, we have identified secondary studies that focus on specific software engineering practices, and underline the importance of using ML technologies. More specifically, Sharma and Spinellis [25] and Azeem et al. [5] performed secondary studies on code smell detection technologies and acknowledged that many modern approaches employ machine learning algorithms. In a similar context Heckman et al. [11] performed a SLR on approaches for providing bad design alerts, through static analysis. Finally, various studies that de-livered overviews of cost / effort estimation approaches emphasize the popularity of ML technologies for providing more accurate estimates [13][24][27].

Background Information. The TD metaphor relies on two concepts borrowed from economics: namely principal and interest. TD principal refers to the effort required to eliminate all inefficiencies that are identified in the current version of the software [2]. Whereas, TD interest refers to the extra maintenance effort required to modify the software, due to the presence of debt. For example, when an artifact needs to be maintained for the introduction of a new feature, additional effort needs to be spent in resolving it, due to inferior design quality [7]. Another concept related to TDM is interest probability. In TD literature, instability (i.e., the susceptibility of an artifact to change) is considered as a proxy of interest probability. In particular, artifacts of high instability are more probable to accumulate interest, since it manifests only during maintenance activities [4]. According to Li et al. [22], TDM can be decomposed to eight activities, synthesized as follows to four categories: (a) Visualizing TD—TD representation, communication that reflect the way that TD can be presented among stakeholders, and monitoring which follows the evolution of TD; (b) Quantifying TD—TD identification (i.e., finding which artifacts suffer from TD) and measurement (i.e., mapping the extent of the problem to some numerical value); (c) Prioritizing TD—The process of TD prioritization ranks identified TD items, according to certain predefined rules to support deciding which TD items should be repaid first and which TD items can be tolerated until later releases; and (d) Reducing TD—To reduce TD, two activities can be performed, namely TD prevention and TD repayment.

3. Study Design

This section presents the design of the systematic literature review. A protocol is a predetermined plan that describes research questions and how the study will be conducted. In the next sub-sections, we present the decisions taken in each study design phase [19].

Research Objectives and Research Questions. The goal of this study can be described as follows: "*Analyze* existing software engineering literature *for the purpose of* understanding the application of machine learning technologies for solving software engineering practices, *with respect to*: (a) the targeted software engineering practices; (b) the proposed machine learning solutions; and (c) the mapping between them". To systematically explore the aforementioned goal, our study is built around three RQs:

RQ$_1$: Which SE problems are solved with machine learning technologies?

- **RQ$_{1.1}$**: Which SE practices are targeted by ML approaches?
- **RQ$_{1.2}$**: Which quality attributes are benefited by the ML technologies?

RQ$_2$: Which machine learning technologies have been used for approaching software engineering problems?

- **RQ$_{2.1}$**: Which are the most common learning styles (i.e., unsupervised, supervised, or semi-supervised) used in SE?
- **RQ$_{2.2}$**: Which are the most common ML algorithms used in SE?

RQ$_3$: What is the mapping between SE problems and ML solutions?

Software engineering is a mature science field, which, however, strives for new solutions to its well-known problems. With the rise of artificial intelligence and the increment of the volume of data produced during software development, many researchers have tried to investigate how artificial intelligence (specifically machine learning) can aid in improving analysis and predictions problems. On the one hand, RQ1 tries to catalogue the software engineering practices that are approached through machine learning, placing special emphasis on the practices that are attempted to be improved and the targeted quality attributes (QA) of interest. On the other hand, RQ2 investigates machine learning technologies that aim at satisfactorily solving software engineering problems, compared to more traditional approaches. Special emphasis is placed on machine learning algorithms, learning styles, challenges, and success indicators. Finally, RQ3 attempts to synthesize the findings of the previous research questions with the goal of mapping solutions to practices in which machine learning is used.

Search Process. The search procedure aims at the identification of candidate primary studies. The search plan involved automated search into five top-quality publication venues. Narrowing the search space of the primary studies to specific top-quality venues is acknowledged as a well-known practice [19] for broad studies, in the sense that it guarantees the quality and relevance of primary studies [1]. Venue selection was based on the process applied by Karanatsiou et al. [15], in the well-known series of bibliometric studies for top-scholars and institutes in software engineering, being published for more than two decades by JSS. The venue selection process is based on four criteria: (a) venues ***classified*** as "Computer Software" by the Australian Research Council; (b) ***evaluation*** higher than or equal to level "B" in the same schema; (c) on average ***more*** than 1 ***citation per month*** per published article; and (d) ***general-scope journals***, not restricted to phases or activities. Next, based on the above, we retained the top-5 journals (excluding magazines). In particular, we searched the articles identified in *Information and Software Technology*, *IEEE Transactions on Software Engineering*, *ACM Transactions on Software Engineering and Methodology*, *Journal of Systems and Software*, and *Empirical Software Engineering*. In particular, in Figure 1 we present an overview of the process along with the number of studies at each step. Finally, we retrieved 90 primary studies. The oldest publication is from 1995 and the newsiest from 2019: 82% of the publications are from 2010 and on.

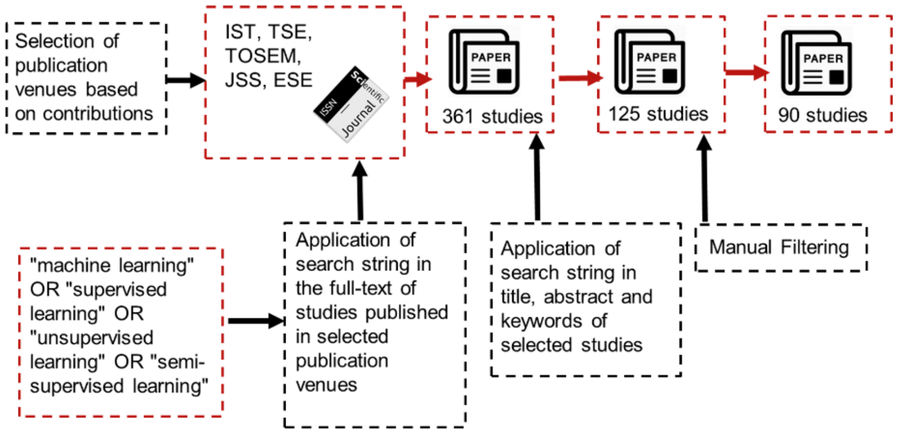
Selection of publication venues based on contributions
IST, TSE, TOSEM, JSS, ESE
ISSN Scientific Journal
PAPER
361 studies
PAPER
125 studies
PAPER
90 studies
"machine learning" OR "supervised learning" OR "unsupervised learning" OR "semi-supervised learning"
Application of search string in the full-text of studies published in selected publication venues
Application of search string in title, abstract and keywords of selected studies
Manual Filtering

Fig. 1. Overview of Search Process

Since all publication venues are strictly on the software engineering field, the search string needed to be focused only on ML technologies. As keywords for the search string we have chosen to use simple and generic terms, which may yield as many meaningful results as possible without any bias or preference to a certain machine learning method or technique. Thus, apart from the term "machine learning" per se, we used the most common learning styles, i.e., "supervised", "unsupervised", and "semi-supervised" learning [3].The search string has been applied to the abstract and title of the manuscripts of all selected venues, without any time constraints. The search has been conducted automatically through the DLs of each venue. The final search string was:

```
"machine learning" OR "supervised learning" OR "unsupervised learning" OR
"semi-supervised learning"
```

Articles Filtering Phase. The papers that were selected as candidate primary studies in the review should be relevant to applications of machine learning in software engineering. In line with Dybå and Dingsøyr [8], an important element of the systematic mapping planning is to define the Inclusion Criteria (IC) and Exclusion Criteria (EC). A primary study is included if it satisfies one or more ICs, and it is excluded if it satisfies one or more ECs. The inclusion criteria of our systematic mapping are: ***IC1***: The study applies one or more ML technologies to a SE practice; and ***IC2***: The study defines one or more ways to evaluate quality with ML. The exclusion criteria of our systematic mapping are: ***EC1***: Study is an editorial, keynote, opinion, tutorial, workshop summary report, poster, or panel; ***EC2***: Study's full text is not available; and ***EC3***: Study mentions ML only in introduction or related work section.

The identified articles went through these inclusion/exclusion criteria, by taking into account the full text of the articles. Article inclusion and exclusion was performed independently from the first and second author, and conflicts have been resolved through discussion among the first three authors. During this process 24 conflicts have been identified and resolved either through an unanimous inclusion or exclusion of the article under consideration.

Quality Assessment. We omitted the step of quality assessment for two reasons: (a) since all papers have been obtained from top-quality venues in software engineering, their quality is (to some extent) ensured by the rigorous review process of the selected venues; and (b) we have set no research questions on the quality of research in the domain under study.

Data Collection. During the data collection phase, we collected data on a set of variables that describe each primary study. Similarly to article inclusion/exclusion, the data collection process, has also been handled independently by the first author and the second author. If both reviewers assigned the same value to one variable, this value would be assigned to the variable without further discussion. Conflicts have been resolved at two levels, first the two authors discussed internally, if no consensus was reached, then the discussion was extended to the third author. First level conflicts have been found in 18 studies, whereas second level conflicts were resolved in 6 studies. For every study, we have extracted the following data: [***V1***] Year; [***V2***] Title; [***V3***] Publication Venue; [***V4***] SE practice (e.g., cost estimation, refactoring); [***V5***] Targeted QA (business [17] or product qualities [14]); [***V6***] Learning Styles (i.e., un-, semi-, or supervised); [***V7***] ML Algorithm; [***V8***] Challenges (challenges of applying ML to SE data); and [***V9***] Evaluation Metrics (for ML).

Data Analysis. From the aforementioned variables [V1], [V2] and [V3] have been used for documentation purposes only. The analysis strategy for the research questions is as follows: to answer RQ_1 and RQ_2, we provide frequencies on variables [V4]-[V5] and [V6]-[V9], respectively. To answer RQ_3, we perform crosstabulation of the same variables. We note that due to a lack of quantitative data, no hypothesis testing or statistical analysis has been conducted.

4. Results

In this section we present the results of data analysis, organized by RQ. We note that the synthesized view of the results (i.e., the transfer of the obtained results in TDM context) is provided in Section 5.

Software Engineering Applications. In Table 1 we present the frequency of software engineering practices that are approached with ML. Through the analysis, we have identified 9 high-level (HL) software engineering practices. For each HL practice, we present their frequency, and SE problems which are solved through ML. By acknowledging the inherent relationship of TDM to maintainability, in Table 2, we provide an overview of the QAs that are targeted in each application of ML. From the obtained results we can observe that: (a) maintainability and its sub-characteristics (namely: testability, reusability, modifiability and analyzability) are a common target for ML technologies—i.e., ML technologies are relevant to TDM; and (b) business quality attributes are also targeted by ML—rendering them relevant to TDM, in the sense that optimizing business QAs is a main root for the accumulation of TD [18].

Table 1. Software Engineering Problems Approached with ML

SE Practice	**#**	**SE Problems**
Defect Management	21	Fault Proneness Prediction and Prioritization, Defect Prediction, Fault Localization
Cost/Effort Estimation	17	Development Cost/Effort Estimation, Software Maintenance Effort Prediction, Maintenance Type Classification
Design-time QAs	14	Change Proneness Prediction, User Interface Design, Software Product and Process Quality Assessment, Code Smells, Patterns and Tactics Detection, API Instability Detection, Refactoring of Test Suites, Refactoring Recommendations
Project Management	12	Bug Report and Change Requests Assignment Recommendations and Prioritization, Classification of Software Bugs, Commit Log Recommendations, Code Review Prioritization, Configuration Management Recommendation, Development Activity Detection, Software Upgrades Recommendation
Security	11	Malware, Malicious Code and Intrusion Classification/Detection, Fault Injection Detection, Software Vulnerabilities Detection
Requirements Engineering	9	Functional Requirements Recommendations, Non-Functional Requirements Detection, Requirements Prioritization, Requirements Assessment, Software SPL Configurations Detection, Application Domain Classification
Run-time QAs	3	Performance Prediction, Energy Efficiency Recommendations
Reuse	2	API Usage Recommendation, Code Examples Prioritization for Reuse
Program Comprehension	2	Trace Recovery, Reverse Engineering

Machine Learning Technologies. To solve the aforementioned problems a variety of ML algorithms and learning styles have been used. The dominant learning style is supervised learning algorithms (89%), followed by unsupervised (6%) and semi-supervised learning (5%). In Table 3 we present the most frequently used algorithms (i.e., used in more than 10 studies). Apart from the algorithm name and the frequency of its appearance, we also provide the generic category in which it can be classified. We note in cases when the authors have not specified a concrete algorithm (e.g., neural networks) the term Generic has been used as the ML algorithm. To evaluate an ML solution there are many performance measures. Performance measures are typically specialized to the class of the problem: e.g., classification, regression, clustering etc. For problems with discrete output such as classification / clustering, researchers use metrics that compare the actual with the predicted values such as precision, recall, etc. For problems with continuous output, such as regression they prefer metrics that capture error rate of predictions—e.g., MMRE, pred(0.25), etc.

Table 2. Targeted Quality Attributes

HL QA	Freq.	Low Level QA
Maintainability	29	Testability, Reusability, Modifiability, Analyzability
Functional Suitability	24	Functional Correctness
Security	12	-
Business Goals	10	Improve Market Position, Reduce Cost of Development
Performance Efficiency	5	Resource Utilization
Usability	1	-
Reliability	1	-

Table 3. Machine Learning Algorithms

ML Algorithm	Freq.	Generic Category
Bayesian Networks	35	Probabilistic Analysis
ID3, C4.5, CART	33	Decision Trees
SVM	31	Kernel Methods
Neural Networks	18	Biologically-inspired Computation
Random Forest	15	Ensemble Learner
Ripper	14	Rule System
Regression	13	Statistical Analysis
K-Means	13	Clustering
KNN	12	Nearest Neighbor

Mapping of SE practices to ML Approaches. As a next step, having presented the results originating from each discipline independently; we present a classification schema, in which we map the most common HL software engineering practices to the ML algorithms that have been used for solving them (see Fig. 2). To investigate if a relation between specific ML algorithms and software engineering practices exists, we have performed a chi-square test. The results suggested that the two variables are associated (alpha < 0.01). Therefore, according to the findings of the SLR, specific algorithms appear to be more appropriate for specific practices and vice-versa.

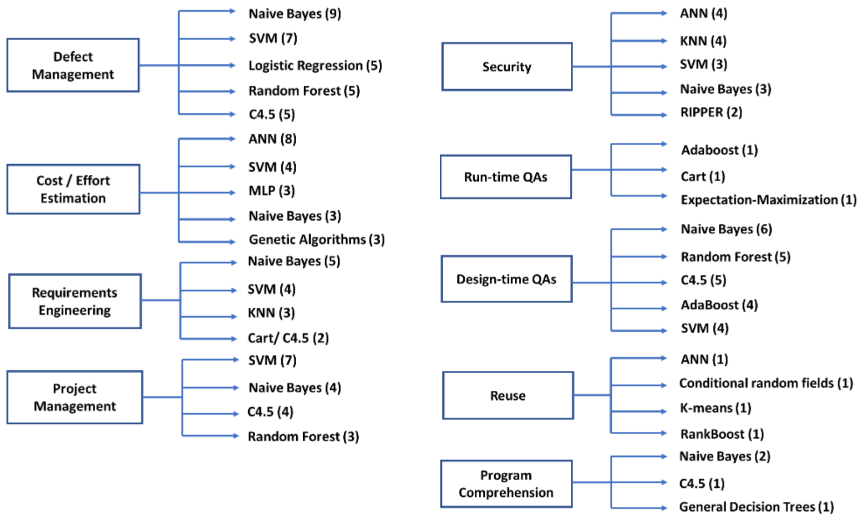

Fig. 2. Mapping of ML to Software Engineering Practices

5. TDM through Machine Learning

In this section we discuss the main findings of this work, i.e., the current status of research on using ML for SE problems, the identified opportunities for the TD community, and the challenges that might exist when applying ML in TDM research.

Current Status. We have observed that machine learning technologies have been applied to resolve multiple and quite diverse research problems; however, some of them appear to be prevalent. In particular, we observed that *defect management*, *cost/effort estimation*, *management of design-time quality attributes*, *recommendations for efficient project management*, and *detection of security threats* are the most common SE practices that have been investigated. We note that as management we refer to cases that we predict (future state), assess, classify, or detect a phenomenon of interest. In terms of quality attributes, the most relevant ones appeared to be the improvement of *maintainability* and *functional suitability* (i.e., correctness), followed by security and business quality attributes. In terms of ML algorithms, we suggest that Bayesian Networks, various Decision Trees, and SVM are the most frequently used ones. Finally, we identified that *Neural Network Analysis* appears to be fitting for *Cost / Effort Estimation* practices, *Bayesian Networks* for *Defect* and *Project Management* practices, and *Random Forrest* algorithms appear to be appropriate for *Managing Design-Time QAs*. On the other hand, *Clustering* and *Decision Trees* appear to be equally fitting for various SE practices.

TDM Opportunities. Based on the above results, it is evident that many of the studied practices and QAs of interest are related to TDM, and therefore can drive to interesting future research implications. On the one hand, regarding the results of Table 1 on the most frequently studied SE practices, we can observe that the vast majority can be mapped to TDM activities, as presented by Li et al. [22]. The only exceptions are Security and Management of Run-time quality attributes, whose inefficiencies, by definition are not categorized as TD. In particular, the following practices can be mapped to TD activities. For each TDM activity, we present the SE practices to which they map, and next how the SE practice can be used in the context of TDM research and practice.

- *TD Identification* deals with recognizing the software artifacts that suffer from TD and the particular problems that they contain. Therefore, studies that focus on ***Code Smells, Patterns and Tactics Detection*** (e.g., [9]) through ML approaches for *Improving Design-Time Quality Attributes* are considered as fitting for elaborate TD Identification. Based on the above, researchers should try to improve the detection accuracy of such approaches, whereas practitioners can use accompanying tools to identify design hot-spots, i.e., parts of the system that yield quality improvements.
- *TD Quantification*: Monetization is a key concept in the TD metaphor: to perform TDM, both principal and interest need to be quantified in some currency form. To this end, ***Cost/Effort Estimation*** methods (e.g., [10][23]) are highly relevant. However, in these studies, the authors do not discuss the findings in the context of TD quantification. On the one hand, researchers are encouraged to introduce cost or effort estimation approaches (e.g., based on past data) to predict the cost of applying refactoring (i.e., related to *TD principal quantification*) or to predict the cost of future maintenance effort (i.e., related to *TD interest quantification*). On the other hand, practitioners can use existing (or novel) such approaches, for getting monetary estimations of their TD, to improve the communication of poor software quality cost to higher non-technical management.
- *TD Prioritization*: In the literature, three ways of TD prioritization have been proposed, i.e., based on principal, interest, and interest probability. In that sense, studies that focus on ***Change***- [16] and ***Fault-proneness*** [29] ***assessment*** are relevant to TD prioritization, since these concepts are closely related to *interest probability*: changes and faults lead to maintenance activities that can accumulate interest. Based on this, researchers can introduce algorithms that predict which software modules are more prone to changes and faults; providing practitioners with tool support for identifying modules that need special attention in their TDM. Finally, regarding cases in which a monetization of TD interest is not of primary importance for prioritization, ranking in terms of maintainability (i.e., a proxy of interest) is a satisfactory compromise of accuracy and ease of use. Therefore, any method that is used for assessing or characterizing the levels of QAs (e.g., maintainability [12]) can be useful for prioritization based on interest.
- *TD Repayment / Prevention*: Regarding TD repayment, currently there are various approaches that propose the ***identification of refactoring opportunities***, or the ***ordering*** with which such ***refactorings*** shall be performed. Additionally, the adoption of ***reuse strategies***, as well as the ***creation*** of ***traces*** along artifacts are expected to be beneficial for preventing the accumulation of new TD principal. Based on the

above, on the one hand, researchers are expected to propose ML-based refactoring identification strategies by optimizing TD principal and interest minimization; allowing practitioners to perform more informed TD repayment. On the other hand, researchers are encouraged to first explore the relation between specific practices (e.g., traceability and reuse) to TD prevention, and if the relation is positive to provide mechanisms to practitioners for applying them into their system.

On the other hand, by considering the targeted quality attributes (see Table 2), we can also identify some connection to TDM. First, since the most frequently targeted quality attribute is maintainability, we can easily assume that all technologies used to improve maintainability are relevant to TD (see Section 2). Additionally, in many studies ML approaches are used to apply practices that aid in terms of the improvement of the market position of the product, or to reduce the development costs (e.g., by shrinking product time-to-market). In general, the satisfaction of business goals is roots of accumulating TD principal, e.g., bring the product to the market faster. Additionally, the improvement of the market position of a product can be considered as a by-product of TDM, especially in cases when combined through TD prioritization.

Challenges in Applying ML to TDM. As part of the analysis, we have identified specific challenges in applying ML to TDM practices. Among the most important ones we acknowledge the following. First, there is a need of a substantial pre-processing in the used datasets, so as to eliminate cases of imbalanced datasets, handling of duplicate values, multicollinearity of predictor variables, etc. Additionally, specifically in TDM it is expected to face many difficulties in creating a solid dataset, since the methods for quantifying TD are highly diverse and no state-of-practice techniques exist. Furthermore, for supervised learning algorithms labelling of training data (e.g. software modules) can be challenging as no universal approach for measuring TD exists. In contrast to other fields (e.g., cost estimation) there is a lack of benchmarks that can be used for training and testing of algorithms (e.g., COCOMO or ISBSG). Furthermore, a common challenge in applying ML in software engineering is the curse of dimensionality, in which the researcher shall limit the variables that shall be fed into the model. This challenge is also highly relevant to TDM, in the sense that TD is a multi-dimensional concept, whose assessment requires the consideration of multiple aspects (e.g., code smell, improper architectural decisions, etc.) but also people's habits and employed processes. Therefore, since the application of ML approaches requires a small subset of input variables to obtain a time-efficient, accurate, and noiseless model, it is of paramount importance to effectively perform data reduction.

6. Threats to Validity

In this section, we present the threats to validity that have been identified and mitigated as part of the study design. The threats are organized based on the guidelines for identifying, mitigating, and reporting threats to validity for secondary studies in software engineering proposed by Ampatzoglou et al. [1].

Study Selection Validity. To guarantee that all studies relevant to the topic have been identified, we systematically developed a search string, based on the types of existing machine learning approaches. However, it is possible that we have missed studies that mention in the title specific ML methods, such as deep learning, neutral networks, etc. To guarantee the relevance to software engineering, we have selected five journals that publish only SE articles. The full-texts of all articles were available through the used Digital Library, and were all written in English. Since our goal was to target high quality research only, we have excluded grey literature. To adequately filter articles, we have predefined a list of inclusion / exclusion criteria, which were discussed among others and piloted, with random screening, and authors voting.

Data Validity. Although we have limited our search to five publication venues, we have retrieved 90 papers for inclusion in the study and data collection, which constitutes our sample size as large enough for analysis. The selection of variables has been based on the set of research questions, and therefore is adequate for answering them. Although our results come from only five venues, we believe that there is no publication bias, since the articles in the top journals come from various communities. The quality of the primary studies is guaranteed by the quality of selected venues. To avoid data extraction bias, more than one author has been involved in the process: one has double-checked the results of the other, and agreement rates have been captured. In case of disagreement, open discussions have been performed.

Research Validity. To increase the reliability and replicability of the study, we involved more than one researcher to all steps of the process, and all data have been made available. Finally, we ensured that the correct research method has been used, i.e., an SLR since a synthesis was required to achieve the high-level goal. However, we acknowledge that the lack of direct related work has not allowed comparison of results; however, the experience of the authors on TDM research allowed interpretation of results, increasing generalisability.

7. Discussion / Conclusions

This study investigates how machine learning (ML) technologies can be applied in Technical Debt Management (TDM): to the best of our knowledge, there is no Systematic Literature Review study that focuses on how ML is applied to TDM. To achieve this goal, we have performed a broad literature review, i.e., on how ML technologies have been applied to solve SE practices in general. The results of the analysis suggest that: (a) the most common SE practices that have been approached through ML technologies are defect management and cost/effort estimation; (b) the target of these technologies is to improve both product (e.g., maintainability) and business (e.g., reduce development time) qualities; and (c) that some ML technologies better map to specific SE practices; however, others are so widespread that can be applicable to various cases.

The results of the study can provide multiple implications to researchers and software development industries. Regarding software development industries, the relevance of ML in resolving software engineering practices can highlight the potential benefits of hiring personnel (e.g., data scientists) that are dedicated in data analysis and interpretation. The outputs of the provided analysis can be proved useful in many aspects of the development, as presented in Table II. Additionally, software practitioners are encouraged to incorporate into their daily processes tools (or research prototypes) that are based on ML, and make use of the provided recommendations, or assessments (e.g., predictions, detections, etc.). On the other hand, we suggest TDM researchers to start exploring the possibility of applying machine learning technologies in their research endeavours. More specifically, we prompt them to migrate solutions from traditional SE practices (e.g., cost estimation, smell detection, etc.) to the context of technical debt management, since they are considered as very relevant. Additionally, the existence of various and non-trivial challenges in the adoption of ML in TDM research, strengthens the aforementioned argumentation, in the sense that high-quality research outcomes shall be produced to resolve them.

ACKNOWLEDGEMENTS

Work reported in this paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871177 (project: SmartCLIDE).

References

1. Ampatzoglou, A., Bibi, S., Avgeriou, P. Verbeek, M., Chatzigeorgiou, A.: Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. Information and Software Technology, 106(2), pp. 201-230 (2019).
2. Ampatzoglou, Ar., Ampatzoglou, Ap., Chatzigeorgiou, A., Avgeriou, P.: The financial aspect of managing technical debt: A systematic literature review. Information and Software Technology, 64(8), pp. 52-73 (2015).
3. Aroussi, S., Mellouk, A.: Survey on machine learning-based QoE-QoS correlation models. International Conference on Computing, Management and Telecommunications (ComManTel'), Da Nang, Vietnam, 27-29 April 2014.
4. Arvanitou, E. M., Ampatzoglou, A., Chatzigeorgiou, A., and Avgeriou, P.: Introducing a ripple effect measure: A theoretical and empirical validation. International Symposium on Empirical Software Engineering and Measurement (ESEM'15), IEEE, China, Oct. 2015.
5. Azeem, M. I., Palomba, F., Shi, L., Wang, Q.: Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. Information and Software Technology, 108(4), pp. 115-138 (2019).
6. Codabux, Z., Williams, B. J.: Technical debt prioritization using predictive analytics. 38th International Conference on Software Engineering Companion (ICSE '16), ACM, 2016.
7. Chatzigeorgiou, A., Ampatzoglou, Ap., Ampatzoglou, Ar., Amanatidis, T.: Estimating the breaking point for technical debt. 7th International Workshop on Managing Technical Debt (MTD' 15), IEEE, Germany, pp.53-56, 2 Oct. 2015.
8. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: a systematic review. Information and Software Technology, 50(9–10), pp. 833–859 (2008).
9. Fontana, F. A., Mantyla, M. V., Zanoni, M., Marino, A.: Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering, 21(3), pp. 1143-1191, June (2016).
10. Hamill, M., Goseva-Popstojanova, K.: Analyzing and predicting effort associated with finding and fixing software faults. Information and Software Technology, 87(7), pp. 1-18, (2017).

11. Heckman, S., Williams, L.: A systematic literature review of actionable alert identification techniques for automated static code analysis. Information and Software Technology, 53(4), pp. 363-387 (2011).
12. Herbold, S., Grabowski, J., Waack, S.: Calculation and optimisation of thresholds for sets of software metrics. Empirical Software Engineering, 16 (6), pp. 812-841 (2011).
13. Idri, A., Hosni, M., Abran, A.: Systematic literature review of ensemble effort estimation. Journal of Systems and Software, 118(8), pp. 151-175 (2016).
14. ISO/IEC 25010:2011, Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models, Geneva, Switzerland (2011).
15. Karanatsiou, D., Li, Y., Arvanitou, E. M., Misirlis, N., Wong, W. E.: A bibliometric assessment of software engineering scholars and institutions (2010–2017). Journal of Systems and Software, 147(1), pp. 246–261 (2019).
16. Kaur, L., Mishra, A.: Cognitive complexity as a quantifier of version to version Java-based source code change: An empirical probe. Information and Software Technology, 102, 2019.
17. Kazman, R., Bass, L.: Categorizing Business Goals for Software Architectures. CMU/SEI-2005-TR-021 (2005).
18. Kazman, R., Cai, Y., Mo, R., Feng, Q., Xiao, L., Haziyev, S., Fedak, V., Shapochka, A.: A Case Study in Locating the Architectural Roots of Technical Debt, 37th International Conference on Software Engineering, IEEE, Florence, pp. 179-188, 16-24 May 2015.
19. Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., Linkman, S.: Systematic literature reviews in software engineering – a systematic literature review. Information and Software Technology, 51(1), pp. 7–15 (2009).
20. Kitchenham, B., Pfleeger, S. L.: Software quality: the elusive target. IEEE Software, IEEE, 13(1), pp. 12 - 21 (1996).
21. Kruchten, P., Nord, R. L., Ozkaya, I.: Technical Debt: From Metaphor to Theory and Practice. IEEE Software, 29 (6), pp. 18-21 (2006).
22. Li, Z., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. Journal of Systems and Software, 101(3), pp. 193-220 (2015).
23. Mair, C., Kadoda, G., Lefley, M., Phalp, K., Schofied, C., Shepperd, M., Webster, S.: An investigation of machine learning based prediction systems. Journal of Systems and Software, 53(1), 23-29 (2000).
24. Myrtveit, I., Stensrud, E., Shepperd, M.: Reliability and validity in comparative studies of software prediction models. Transactions on Software Engineering, IEEE 31(5), 2005.
25. Sharma, T., Spinellis, D.: A survey on software smells. Journal of Systems and Software, 138(4), pp. 158-173 (2018).
26. Skourletopoulos, G., Mavromoustakis, C., Bahsoon, R., Masotrakis, G., Pallis, E.: Predicting and quantifying the technical debt in cloud software engineering. 19th International Workshop on Computer-Aided Modeling and Design of Communication Links and Networks (CAMAD), IEEE Computer Society, 2014.
27. Wen, J., Li, S., Lin, Z., Hu, Y., Huang, C.: Systematic literature review of machine learning based software development effort estimation models. Information and Software Technology, 54(1), pp. 41-59 (2012).
28. Zhang, D., Tsai, J. J. P.: Machine learning and software engineering. 14th IEEE International Conference on Tools with Artificial Intelligence, (ICTAI' 02), 4-6 November 2002.
29. Zhou, Y., Leung, H.: Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. Transactions on Software Engineering, 32 (10), 2006.