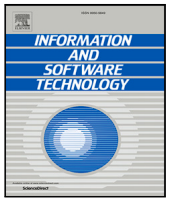
Information and Software Technology 160 (2023) 107226

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



On the effectiveness of automated tracing from model changes to project issues

Wouter van Oosten [a], Randell Rasiman [a], Fabiano Dalpiaz [a,*], Toine Hurkmans [b]

[a] Utrecht University, The Netherlands
[b] Mendix, The Netherlands

ARTICLE INFO

Keywords:
Requirement Traceability
Trace link recovery
Model-driven development
Low-code development
Machine learning

ABSTRACT

Context: Requirements Traceability (RT) is concerned with monitoring and documenting the lifecycle of requirements. Although researchers have proposed several automated tracing tools, trace link establishment and maintenance are still prevalently manual activities.
Objective: In order to foster the adoption of automated tracing tools, we study their empirical effectiveness in the context of model-driven development (MDD). We focus on trace link recovery (TLR) from (i) SVN revisions of MDD models to (ii) JIRA issues that represent requirements and bugs.
Method: Based on the state-of-the-art in automated TLR, we propose the LCDTrace tool that uses 131 features to train a machine learning classifier. Some of these features use specific information for MDD contexts. We conduct three experiments on ten datasets from seven MDD projects. First, we evaluate the effectiveness of three ML algorithms and four rebalancing strategies using all 131 features, and we derive two optimal combinations for trace link recommendation and for trace maintenance. Second, we investigate whether the MDD-specific features convey higher performance than a version of LCDTrace that excludes those features. Third, we employ automated feature selection and study whether we can reduce the number of features while keeping similar performance, thereby boosting time and energy efficiency.
Results: In our experiments, the gradient boosting models outperform those based on random forests. The best combinations for trace recommendation and maintenance achieve an F_2-score of 61% and $F_{0.5}$-score of 67%, respectively. While MDD-specific features do not provide additional value, automated feature selection succeeds at reducing feature numerosity without compromising performance.
Conclusion: We provide insights on the effectiveness of state-of-the-art TLR techniques in MDD. Our findings are a baseline for devising and experimenting with alternative TLR approaches.

1. Introduction

Requirements Trace Link Recovery (RTR) is the process of establishing trace links between requirements and other artifacts [1], when such links are not existing. Many of the automated approaches for trace link recovery (TLR) use information retrieval (IR) techniques [2]; roughly, if two artifacts are textually similar, they should be traced [3]. Common IR algorithms include Vector Space Models, Latent Semantic Indexing, Jenson–Shannon Models, and Latent Dirichlet Allocation [2,4,5].

The ever-increasing adoption of machine learning (ML) has affected automated TLR too [2]. ML approaches treat TLR as a classification problem: the Cartesian product of the two sets of trace artifacts defines the space of candidate trace links [6,7]. A subset of these links, manually defined by domain experts, are valid links. An ML classifier is tasked to build a model that predicts whether unseen trace links are valid or invalid. This is achieved by representing the trace links as a vector of features. Most ML approaches for TLR use similarity scores of IR-based methods as features [6–8] and they have been shown to outperform IR-based TLR approaches [7].

We investigate the effectiveness of ML-based TLR in the context of model-driven development, specifically, in low-code development. In this development paradigm, which has shown potential for RTR [9,10], applications are built by creating models that describe the domain entities and their relationships, the application logic, the algorithms, and the mapping between data and user interfaces. We study how to automatically recover missing trace links from model changes to project management issues.

We respond to the call of the research community [2,5] by experimenting with industrial datasets in an emerging application domain. Thanks to the collaboration with Mendix, an MDD platform producer

* Corresponding author.
E-mail address: f.dalpiaz@uu.nl (F. Dalpiaz).

https://doi.org/10.1016/j.infsof.2023.107226
Received 13 August 2022; Received in revised form 24 March 2023; Accepted 3 April 2023
Available online 10 April 2023

that specializes in low-code development, we investigate ten datasets from seven real-world projects, with the research goal of *studying the effectiveness of ML-based TLR algorithms in the context of MDD*.

Following the Design Science research methodology [11], we conduct an empirical study at Mendix with a focus on vertical traceability [12]: the recovery of trace links between artifacts at different abstraction levels. Our main contributions are as follows:

1. We build the LCDTrace tool, an ML approach that recovers trace links from model revisions to JIRA issues using 131 features from the literature [7,8]. LCDTrace is available as an open-source project: https://github.com/RELabUU/LCDTrace.
2. We use ten datasets from seven projects at Mendix to evaluate the effectiveness of three ML algorithms (random forests, XGBoost, and LightGBM) with four rebalancing strategies. We identify two combinations that are best suited for the scenarios of trace link recommendation and of trace link maintenance.
3. We empirically evaluate whether the MDD-specific features of LCDTrace have an effect on the performance. To do so, we compare the performance of all the features against LCDTrace trained only with those features that do not use MDD-specific information.
4. We study whether we can reduce feature numerosity while maintaining comparable effectiveness, so to decrease energy consumption and processing time. In our experiments, we use the automated feature selection algorithm mRMR [13,14] that discards highly correlated features.

In addition to reporting quantitative results, we attempt to interpret the performance, with the overall aim of understanding the reasons behind the results. To do so, we investigate which features and feature families offer the highest predictive power.

Paper organization. Section 2 presents the background on requirements traceability and automated TLR. Section 3 describes our case: MDD at Mendix. Section 4 presents our research method and the research questions. Section 5 elaborates on the construction of LCDTrace. Sections 6–8 show the results to the research questions. Section 9 discusses the threats to validity, while Section 10 concludes and outlines future work.

2. Related work

RT practices are mandated by standards such as the Capability Maturity Model, ISO 9000/9001, and IEEE 830-1998 [15,16]. Thus, organizations who aim or need to comply with such standards embrace RT practices. These are expected to deliver benefits for project management and visibility, project maintenance, and verification & validation.

Despite these benefits, RT activities are found to be "time-consuming, tedious and fallible" [17]. Even when conducted, manual tracing is favored, leading to traces which are error-prone, vulnerable to changes, and hard to maintain. These difficulties gave birth to a vast research landscape, with many automated techniques that were proposed for recovering trace links.

After providing the terminological background in Section 2.1, we review the major types of approaches to automated TL recovery (without aiming at a comprehensive literature review) in Section 2.2.

2.1. Basics of software and requirements traceability

Gotel et al. [18] define (software) traceability as the *potential for traces to be established (i.e., created and maintained) and used*. A trace is denoted as a triple: (source artifact, target artifact, trace link).

Trace artifacts can be organized into *types* [18] that have a similar structure and/or purpose. For instance, requirements may be a distinct artifact type. In our research, we have two trace artifact types: (i) *JIRA issues*, which indicate requirements or bugs; and (ii) *revisions*, which summarize the changes in the models that the MDD developer commits to a repository like SVN.

We use revisions as our source artifact, and JIRA issues as our target artifact. Since JIRA issues contain also requirements, this setting is an instance of *requirements traceability* [19]. In particular, we study *vertical* traceability: relating artifacts at differing levels of abstraction [18]. This contrasts with horizontal tracing, which focuses on the establishment and maintenance of trace links between artifacts at the same level of abstraction.

Traceability can also be characterized in terms of the direction of the trace links recovery, either forward or backward [18]. We investigate backward traceability: *tracing that follows antecedent steps in a developmental path*. In our case, this antecedence also corresponds with a chronological path: we assume that changes cannot be committed for an issue that does not yet exist.

A final distinction relates to the direction with respect to a requirements specification (RS) [19]. Requirements in the RS can be traced either (1) to information prior to the requirements' inclusion in the RS (pre-RS traceability), or (2) to information after their inclusion in the RS (post-RS traceability). We focus on post-RS traceability. Our RS is expressed in a lightweight format: a collection of JIRA issues.

2.2. Automated trace link recovery

Gotel and colleagues [18] distinguish between three terms that pertain to tracing activities: *manual* tracing (e.g., drag and drop methods), *automated* tracing where a tool identifies the links, and *semi-automated* tracing where the tool suggests and a human vets [20] the results.

We investigate automated and semi-automated tracing, a rich area of research that is characterized by numerous approaches. We review the major types of approaches in the following paragraphs.

2.2.1. Information retrieval (IR)

These techniques assume that when two artifacts are textually similar, they should probably be traced to each other [3]. Thus, to recover trace links, an IR-based algorithm computes a score of the textual similarity between software artifacts. Trace links scoring above a threshold are considered valid.

Many IR-based methods use Vector Space Models (VSM): each document is transformed into a vector space, and the similarity between documents is calculated via cosine distance [1]. More advanced techniques are the Jenson–Shannon Models, which take documents as a probabilistic distribution [21,22], and the Jenson–Shannon Divergence as a measure of semantic difference.

There are two fundamental challenges with IR methods. *Synonymy*–the use of different terms for the same concept (e.g., 'drawing' and 'illustration')–threatens recall, unless the tracing algorithm knows which terms are synonyms. *Polysemy*–terms that have multiple meanings (e.g., 'fall')–impacts precision, as IR algorithms are not that strong at word sense disambiguation [23].

Latent Semantic Indexing (LSI) can mitigate these challenges by predicting a word's meaning based on its co-occurrence with other words in a document. Existing approaches that apply LSI to TLR show mixed results [3,24].

Many other approaches have further improved performance. However, according to De Lucia et al. [25], there does not seem to be a clear winner: the best performing IR-based approach depends on the case.

2.2.2. Machine learning

Most state-of-the-art techniques for RTR employ ML nowadays. ML approaches are used for their ability to recognize implicit patterns in the data that are then used to distinguish between valid and invalid traces.

ML approaches treat the TLR process as a classification problem: the Cartesian product of the two trace artifact sets is calculated, and the resulting elements represent *candidate trace links* [6,7]. An ML classifier

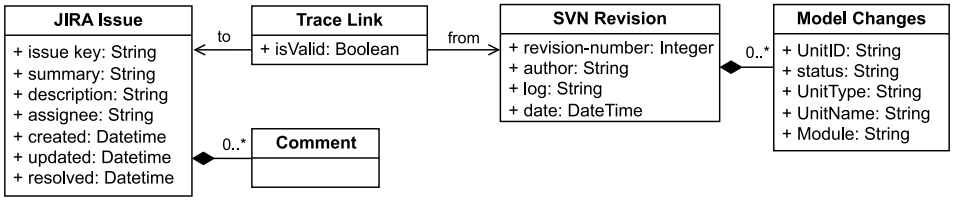


Fig. 1. Meta-model showing the relationships between JIRA issues and SVN revisions.

learns from the sample data, which is manually traced, and the classifier is then used to *predict* whether unseen couples of artifacts should be traced to one another.

In order to learn the patterns that distinguish valid trace links from invalid ones, the classifier uses a characterization of couples of artifacts in terms of features: attributes that can either refer to a single artifact (e.g., the number of words in a requirement), or to the relationship between them (e.g., their textual similarity). Most ML TLR approaches use IR similarity scores as features [6–8], although other features, such as query quality [7], source code location distance [26], process-related features [8], Requirements to Requirements Set (R2RS) metrics [27], have been proposed.

Besides feature representation, researchers have also analyzed which ML classification algorithms perform best. Falessi et al. [27] have compared decision trees, random forests, Naïve Bayes, logistic regression, and bagging, with random forests yielding the best results.

2.2.3. Deep learning

Advances in neural networks can be employed in automated TLR. Guo et al. [28] proposed a combination of word embeddings and recurrent neural networks for establishing forward traces from subsystem requirements to subsystem design descriptions. In their experiments, deep learning outperforms IR approaches based on VSM and LSI.

Another comparison between IR and deep learning is conducted by Lin et al. [29]. They study datasets that include multiple (intermingled) languages and, like us, they link revisions to issues. In their experiments, neural networks outperform IR-based approaches.

Although deep learning approaches have the potential of achieving good results without the costs of feature engineering, neural networks are best suitable with large datasets. Unsurprisingly, researchers have started assembling open datasets, such as SEOSS 33 [30], a collection of data regarding open-source projects. The availability of large data collections is not applicable to all industrial cases, like the one described in this paper, where the number of issues and commits is too low to enable deep learning.

3. Case description: MDD at Mendix

We conduct research in collaboration with Mendix, the producer of the Mendix Studio Low-Code Platform (MLCP). The MLCP employs MDD principles and allows creating software by defining graphical models for the domain, the business logic, and the user interface [31].

We focus on the practices of MLCP developers employed by Mendix, who are building MLCP applications for internal use. These developers follow the SCRUM development process. Product Owners are responsible for managing and refining requirements, which are documented as JIRA issues and added to the product backlog, alongside the bugs. The issues for the Sprint Backlog are chosen by the development team, and each item is assigned to one developer who is responsible for its implementation in the sprint.

After becoming acquainted with the JIRA issue, the MLCP developer opens the latest MLCP model, navigates to the relevant modules, and makes the required changes. These changes determine a revision that is committed to the SVN repository once they fulfill the acceptance criteria. Each revision includes a log message, in which the MLCP developer outlines the changes they made, as well as the JIRA issue ID for traceability purposes.

3.1. Studied artifacts

We consider the artifacts associated with MLCP-based development. We focus on tracing committed SVN revisions to *JIRA*, using manual trace information obtained from development teams who followed traceability practices. Fig. 1 shows the relationships among the trace artifacts.

Jira issues. Within the widespread project management tool Atlassian JIRA, project members define work items called issues, which Mendix uses to document requirements. The following attributes are shared by all JIRA issues: (I1) a unique *issue key* serving as identifier, (I2) a *summary* that represents a user story, (I3) a *description*, which further explains the requirements alongside the acceptance criteria, (I4) an *assignee*: the person who is responsible for implementing the issue. Finally, each issue has three date/time indicating when the issue was (I5) created, (I6) last updated, and (I7) resolved.

SVN revisions. The MLCP, like any modern development environment, employs version control. An adapted version of Apache Subversion is integrated into the MLCP, which the developer can access through a GUI. Each SVN revision[1] contains: (R1) *revision-number*, a unique integer, (R2) *author*, the email of who committed the *revision*, (R3) *log*, an optional text field for the developer to summarize the revision, and (R4) *date*, the date/time when the revision was committed.

Finally, each SVN revision contains *MDD-specific information* that describes the changes made to so-called units: models or elements within a model. Each change is stored as an element of an array that contains (R5) *unitID*, (R6) the *status* (added, deleted, or modified), (R7) *unitName*: the name of that unit, (R8) *unitType*: the category of the unit (e.g., microflow or form), (R9) the *module* where the unit is located. For example, when the developer commits their work, one of the changes in the revision which may indicate that an Entity (*unitType*) called Customer (*unitName*), with identifier 345 (*unitId*) has been added (*status*) to CustomerDB (*module*).

3.2. Studied datasets

We acquired data from seven internal MLCP projects, produced by three development teams. We refer to them as (i) *Company*, (ii) *Control*, (iii) *Data*, (iv) *Learn*, (v) *Portfolio*, (vi) *Service*, and (vii) *Store*. For projects Control and Learn, we obtained multiple datasets, each referring to a different time period, leading to Control_1, Control_2, Control_3, Learn_1, and Learn_2. For each project, we used a data export of one JIRA project and one MLCP repository.

Table 1 summarizes the availability of manually established traces, distinguishing between revisions that trace to no issues, to a single issue, and to two or more issues. Several revisions are untraced. This could be because the revision is too generic (e.g., creation of a branch), or because the developer forgot about tracing. Also, the revisions were not always traced to issue keys of the JIRA projects we acquired. This happens because multiple teams, each with their own JIRA project, may operate on the same repository.

[1] Throughout the paper, we use *revision* to indicate an *SVN revision*

Table 1
Summary of the acquired project data. The average values in the last row, here and elsewhere, are reported as the macro-average across the datasets.

Dataset	Team ID	Issues	Revisions	Revisions traced to					
				0 issues		1 issue		≥2 issues	
				Count	%	Count	%	Count	%
Company	A	29	626	153	24.44	440	70.29	33	5.27
Control_1	A	111	1,629	485	29.77	1,118	68.63	26	1.60
Control_2	A	116	768	275	35.81	488	63.54	5	0.65
Control_3	A	99	498	145	29.12	335	67.27	18	3.61
Data	B	58	818	236	28.85	556	67.97	26	3.18
Learn_1	C	278	1,239	211	17.03	894	72.15	134	10.82
Learn_2	C	168	303	105	34.65	166	54.79	32	10.56
Portfolio	A	97	1,056	254	24.05	781	73.96	21	1.99
Service	B	173	2,930	1,435	48.98	1,462	49.90	33	1.13
Store	C	634	713	508	71.25	202	28.33	3	0.42
Average		176	1,058	381	34.40	644	61.68	33	3.92

3.3. Traceability practices at Mendix

We reviewed the traceability practices at Mendix to better assess the industrial applicability of our research. We conducted two focus groups with a total of eight (four per session) industry experts from Mendix: tech leads, architects, and engineers. We discussed every aspect of traceability within their development process, not limited to our JIRA issues and model revisions. Mendix has numerous teams across the company that develop applications for both internal use and external clients. These teams can choose how their team operates, to a large extent, making them in charge of their process.

Establishing trace links by including a JIRA issue key when making revisions is seen as 'good practice' by the development teams. However, how this is organized within the teams varies. While some teams follow simply instruct their members to include the identifier, others adopt more advanced processes. Some teams defined a repository rule that checks if the revision log of a committed change includes an issue key. Any revision that includes a key that is not part of the current sprint gets rejected.

Still, both cases allow for errors to be introduced, for instance by misspelled issue keys. As the experts said, fixing errors without having trace links to the model changes is more challenging. The experts showed great interest in our field of study. They highly value their traceability practices, but at the same time they would not want solutions that over-constrain their processes.

4. Research method

As stated in the introduction, our research goal is that of *studying the effectiveness of ML-based TLR algorithms in the context of MDD*. We aim to establish whether and to what extent state-of-the-art TLR approaches can be applied to MDD, with a focus on those models used in low-code development.

After presenting the evaluation scenarios in Section 4.1, we describe and illustrate the research questions and the research approach in Section 4.2.

4.1. Evaluation scenarios and metrics

We adopt the two scenarios by Rath et al. [8]: *Trace Recommendation* and *Trace Maintenance*. We evaluate LCDTrace in both scenarios using the F-measure, the harmonic mean between precision and recall. We take the versions of the F-measure [32] proposed by Rath et al..[2]

Trace recommendation. At Mendix, MDD developers use a GUI to commit changes to the remote repository. When doing this, the developer outlines the changes made and can specify (*log* field of *SVN revision* in Fig. 1) the identifier of the relevant JIRA issue(s). Integrating a trace recommendation system can improve this scenario by showing a list of candidate issues that are most likely to be linked to the current revision. The developer manually vets the trace links. It is cognitively affordable and relatively fast since developers generally know which specific JIRA issue they have implemented. This scenario requires high recall: valid traces must be in the list for a developer to review them. Precision is less important because developers can ignore invalid traces. Therefore, Rath and colleagues suggest using the F_2-measure.

Trace maintenance. Not all the revisions are traced to a JIRA issue. As visible in the '0 issues' columns of Table 1, circa 34% of the revisions are not traced to issues. Thus, maintenance is needed to recover traces, which leads to the goal of the second scenario: an automated trace maintenance system. Such a system would periodically recover traces that were forgotten by the developers, ultimately leading to a higher level of RT. Since this system should consider the Cartesian product of all issues and all untraced revisions, precision is more important than recall, leading to Rath's suggestion of using the $F_{0.5}$-measure.

4.2. Research questions and approach

We describe the research questions and the research approach; see Fig. 2 for an illustration. We aim at a thorough empirical assessment, rather than contributing new techniques for automated TLR. Therefore, the first research question focuses on investigating the most effective ML approach for our context:

> RQ1: What is the most effective ML approach for trace maintenance and trace recommendation between revisions and JIRA issues in MDD contexts?

To answer RQ1, we need to define what is meant by ML approach, which we characterize as (i) a classifier algorithm, (ii) a rebalancing strategy, and (iii) a set of features. Details on these aspects are provided in Section 5. For now, note that we take 131 features from the literature, and that we investigate a combination of three classifier algorithms and four rebalancing strategies.

The answer to RQ1 leads to two combinations of classifier algorithm and rebalancing strategy, one per each of the two evaluation scenarios introduced in Section 4.1. A first answer to RQ1 was given in our previous work [33]; here, we rely on ten datasets rather than three, and we use a slightly improved version of LCDTrace that fixes some bugs. RQ1 is analyzed in Section 6.

The selected ML approaches then feed into two research questions that we did not explore at all in previous work. First, we examine whether the features that make use of MDD-specific information lead to improved performance:

[2] A rigorous estimation of F_β [32] is beyond the scope of this paper. To draw more solid conclusions, we also discuss precision and recall when interpreting the results.

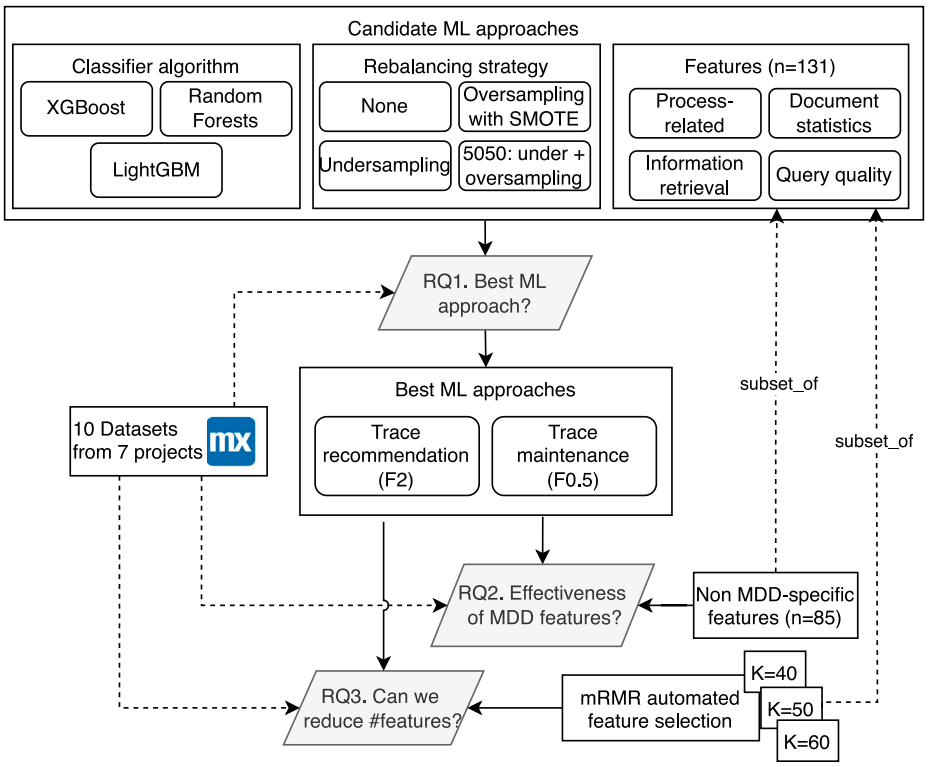


Fig. 2. Illustration of the research approach to address RQ1–RQ3.

RQ2: What is the added value delivered by the features that use MDD-specific information?

To address RQ2, we follow the same experimental procedure that we use for RQ1, but using a subset of the 131 features: those 85 features that do not make use of MDD-specific information (features that make use of *unit names*). By comparing the F-measures of RQ2 with those of RQ1, we assess the added value of the MDD-specific features. RQ2 is analyzed in Section 7.

The best ML approaches from RQ1 also feed into a third research question, which aims at reducing the number of the features, as the higher the number of features, the slower the algorithm and the higher the energy footprint:

RQ3: Can we increase efficiency while keeping similar performance by reducing the number of features?

To answer RQ3, we use a highly efficient automated feature selection algorithm called mRMR (minimum redundancy-maximum relevance) [13,14], which identifies those features that are expected to deliver the highest gain by removing correlated features that would be redundant. We make experiments on our ten datasets trying to select the 40, 50, and 60 most relevant features according to mRMR. We discuss RQ3 in Section 8.

5. Requirement trace link classifier for MDD (LCDTrace)

To support the trace recommendation and trace maintenance scenarios, we construct an ML classifier to categorize the validity of traces, based on recent research [7,8]. Our LCDTrace classifier is publicly available as open source (https://github.com/RELabUU/LCDTrace) and in our online appendix [34].

We first describe how we pre-process the datasets from Mendix and we construct the trace links in Section 5.1. Then, we describe the feature engineering process (Section 5.2), we discuss the classification algorithms (Section 5.3), and we elaborate on the data rebalancing strategies (Section 5.4).

5.1. Data description and trace construction

To train the LCDTrace classifier, we used the data from the ten datasets from Mendix that are summarized in Table 1. For each dataset, we obtained a file including the JIRA issues, and another one with a list of SVN revisions.

Revisions. The input data is a textual file in the Subversion Dump format. LCDTrace uses Regular Expressions to transform the data and to extract the issue key(s) from the log message and to store it as a distinct issue key column. The issue key is then removed from the log message, so that the data can be used for training and testing the classifier.

The log message is further pre-processed using common techniques: (1) words are lowercased, (2) interpunction is removed, (3) numeric characters are removed, (4) sentences are tokenized with NLTK, (5) stop words are removed using the corpus from NLTK, and (6) all remaining terms are stemmed using the Porter Stemming Algorithm [35].

These activities lead to a pre-processed dataset that consists of (using the labels we defined in Section 3.1): R1 (Revision Number), R2 (Author), R3 (Log), R4 (Date), R7 (Unit Names), R8 (merge of Log and Unit Names), and associated JIRA key (a reference to I1).

JIRA issues. The JIRA datasets are comma-separated text files that are exported from JIRA. Pre-processing is carried out similarly to the revisions, leading to a dataset that consists of I1 (Issue key), I2 (Summary), I3 (Description), I4 (Assignee), I5 (Created date), I6 (Last updated date), I7 (Resolved date), plus one additional feature: I9 (JIRA All Natural Text): the union of I2 and I3.

Trace link construction. We discarded revisions that are not traced to issues, because we could not tell whether (i) the revision was related to no issues, or (ii) the trace link had been forgotten. We calculated the Cartesian product between the JIRA issues and the retained revisions. Each element of the Cartesian product represents a *candidate trace link* (revision, issue), whose validity was determined by comparing the issue key to the revision's related issue key. If the issue key is present, the trace link is classified as valid; else, it is invalid.

Table 2
Potential number of trace links ($|I| \times |R|$), candidate trace links by excluding revisions that are not linked to any issue, and valid/invalid traces after applying causality filtering.

Dataset	$\|I\| \times \|R\|$	Causality filtering	Candidate trace links	Invalid traces		Valid traces	
Company	28,154	Before	13,717	13,355	97.36%	362	2.64%
		After	8,067	7,705	95.51%	362	4.49%
Control_1	180,819	Before	126,984	126,363	99.51%	621	0.49%
		After	93,451	92,830	99.34%	621	0.66%
Control_2	89,088	Before	57,188	56,988	99.65%	200	0.35%
		After	21,567	21,367	99.07%	200	0.93%
Control_3	49,302	Before	34,947	34,620	99.06%	327	0.94%
		After	22,646	22,319	98.56%	327	1.44%
Data	47,444	Before	33,756	33,305	98.66%	451	1.34%
		After	27,815	27,364	98.38%	451	1.62%
Learn_1	344,442	Before	285,784	284,965	99.71%	819	0.29%
		After	144,591	143,772	99.43%	819	0.57%
Learn_2	50,904	Before	33,264	33,119	99.56%	145	0.44%
		After	12,195	12,050	98.81%	145	1.19%
Portfolio	102,432	Before	77,794	77,023	99.01%	771	0.99%
		After	51,415	50,644	98.50%	771	1.50%
Service	506,890	Before	258,635	258,215	99.84%	420	0.16%
		After	89,233	88,813	99.53%	420	0.47%
Store	452,042	Before	129,970	129,884	99.93%	86	0.07%
		After	33,627	33,541	99.74%	86	0.26%
Average	185,152	Before	105,204	104,784	99.23%	420	0.77%
		After	50,461	50,041	98.68%	420	1.31%

Then, we applied causality filtering to the trace links [8]: given a candidate trace link, when the revision date is antecedent to the creation date of an issue, the trace link is deemed invalid due to causality. This is a common situation in agile development: JIRA issues are created as the project unfolds, when some or many revisions have already been made. Table 2 summarizes the results about causality filtering. Column $|I| \times |R|$ indicates the potential number of trace links by using the cardinality of the issues and revisions sets in Table 1; column *candidate trace links* discards the revisions that are not linked to any issue. The table also shows the number of valid and invalid traces before and after causality filtering.

5.2. Feature engineering

The existing trace links are used for training an ML classifier. For this, the trace links have to be described as a set of features. Based on literature, we engineered a total of 131 features grouped into four categories: process-related, document statistics, information retrieval, and query quality.

Process-related. These four features build on the research by Rath et al. [8]. Feature F1 captures stakeholder information by indicating if the assignee of the JIRA issue is the author of the revision. The remaining three features capture temporal information. F2 is the difference between the revision date (R4) and the issue creation date (I5), F3 is the difference between R4 and the date when the issue was last updated (I6), and F4 is the difference between R4 and the date when the JIRA issue was resolved (I7).

Document statistics. These features rely on the work of Mills et al. [7]: they gauge document relevance and the information contained within the documents. In our case, a document consists of the text in a JIRA issue or in a revision. Within this category, seven metrics are included:

- *Total number of terms*, calculated for the JIRA issue (F5) and for the revision (F6).
- *Total number of distinct terms* for the JIRA issue (F7) and for the revision (F8).
- *Overlap of terms between JIRA issue and revision*. To calculate this metric, the number of distinct terms that appear both in the JIRA issue and in the revision are divided in three ways, each leading to a feature: by the number of distinct terms in (i) the JIRA issue (F9); the revision (F10); and (iii) either the JIRA issue or the revision (F11).

Table 3
TF-IDF combinations used for VSM.

ID	Revision artifact	Issue artifact	Features
1	Log Message	Summary	F12–F13
2	Log Message	Description	F14–F15
3	Log Message	JIRA All Natural Text	F16–F17
4	Unit Names	Summary	F18–F19
5	Unit Names	Description	F20–F21
6	Unit Names	JIRA All Natural Text	F22–F23
7	Revision All Natural Text	Summary	F24–F25
8	Revision All Natural Text	Description	F26–F27
9	Revision All Natural Text	JIRA All Natural Text	F28–F29

Information retrieval. This set of features captures the semantic similarity between two trace artifacts. We first apply VSM with TF-IDF weighting to transform the trace artifacts to a vector representation. Because we use TF-IDF weighting, the chosen corpus used for weighting impacts the resulting vector. For instance, the term 'want' occurs commonly in the JIRA summary, as Mendix developers write user stories in there. However, it might be rare when considering all the terms in a JIRA issue.

Since we could not determine which corpus best represents the trace artifact, we explore multiple representations: we construct three vector representations for the issues (I2: Summary, I3: Description, I9: Summary & Description) and three representations for the revisions (R3: log message, R7: unit names, and R8: log & unit names). This results in 9 distinct pairs for each trace link candidate, as per Table 3. The cosine similarity of each pair was computed and utilized as a feature. Mills and Haiduc [36] showed that the chosen trace direction (which artifact in the trace link is used as a query) affects traceability performance. Thus, we calculated the cosine distance in either direction, resulting in the 18 IR-features (F12–F29) in Table 3. We used Scikit-learn for TF-IDF weighting and SciPy for calculating the cosine distance.

Query quality. This determines a query's expected ability to retrieve relevant documents in a document collection. A high-quality query returns the relevant document(s) at the top of the results lists, whereas a low-quality query returns them at the bottom of the list or not at all. Query quality provides complementary information to IR-techniques: do two artifacts exhibit low cosine similarity because they actually refer to an invalid trace link, or is this due to a low-quality query?

Table 4
Query Quality Features from the work by Mills and Haiduc [36].

Sub-family	Measure	Metric	Features	
			Query: Revision	Query: JIRA
Specificity	TF-IDF	{Avg, Max, Std-Dev}	F30–F38	F39–F47
	TF-ICTF	{Avg, Max, Std-Dev}	F48–F56	F57–F65
	Entropy	{Avg, Med, Max, Std-Dev}	F66–F77	F78–F89
	Query Scope		F90–F92	F93–F95
	Kullback-Leiber divergence		F96–F98	F99–F101
Similarity	SCQ	{Avg, Max, Sum}	F102–F110	F111–F119
Relatedness	PMI	{Avg, Max}	F120–F125	F126–F131

Mills and Haiduc [36] devised several metrics for query quality (QQ), which we adopt. These QQ metrics are organized into pre-retrieval and post-retrieval. The former consider only the properties of the query, whereas the latter also consider the information captured by the query results. We focus on pre-retrieval QQ metrics, evaluating three different aspects:

- *Specificity:* the query's ability to distinguish relevant documents from irrelevant ones. Highly specific queries contain terms which are rare in the document collection, while lowly specific queries contain common terms. Highly specific queries are desirable.
- *Similarity:* the degree to which the query is similar to the document collection. Queries that are comparable to the collection suggest the existence of many relevant documents, increasing the possibility that a relevant document is returned.
- *Term relatedness:* how often terms in the query co-occur in the document collection. If the query terms co-occur in the document collection, the query is considered of high quality.

We calculated these metrics for each of the six corpora mentioned in the information retrieval paragraph (log message, unit names, revision All Natural text, summary, description, and JIRA All Natural text), resulting in a total of 102 QQ features: F30–F131, listed in Table 4.

5.3. Classification algorithms

We considered two state-of-the-art supervised ML algorithms for classifying the validity of trace links: Random Forests and Gradient Boosted Decision Trees. While the former are shown to be the best RTR classifier in earlier research [7,8], Gradient Boosted Decision Trees outperformed Random Forests in other domains like computational economics [37] and oceanography [38].

To implement Random Forests, we used the framework of Scikit-learn. For Gradient Boosted Decision Trees, we used two frameworks: XGBoost, and LightGBM. These frameworks differ in two major respects. First, the method of splitting. XGBoost splits the tree level-wise rather than leaf-wise, whereas LightGBM splits the tree leaf-wise. Second, how the best split value is determined. XGBoost uses a histogram-based algorithm, which splits a feature and its data points into discrete bins, which are used to find the best split value. LightGBM uses a subset of the training data rather than the entire training dataset. Its sampling technique uses gradients, resulting in training times that are up to 20 times faster than XGBoost [39].

5.4. Data rebalancing

The training data in traceability settings is generally highly imbalanced because only a few links are valid [8,28], causing challenges for classifier training [8]. Table 2 shows that this occurs in our datasets too: valid links are between 0.26% and 4.49% (mean: 1.31%). The positive samples that the classifier would encounter during learning are much lower than the negative ones.

Thus, we applied four rebalancing strategies [7] to the training data:

1. *None.* No rebalancing method is applied.
2. *Oversampling.* The minority class is oversampled until it reaches the size of the majority class, by applying SMOTE [40].
3. *Undersampling.* The majority class is randomly undersampled until it has the same size as the minority class.
4. *5050.* Oversampling via SMOTE is applied to the minority class with a sampling strategy of 0.5: this leads to increasing the numerosity of the minority class until 50% of the majority class. Then, undersampling is applied to the majority class until the sizes of both classes are equal.

6. What is the most effective ML approach (RQ1)?

To answer RQ1, we experimented with the different combinations of the rebalancing strategies in Section 5.4 and of the classification algorithms in Section 5.3. We analyzed the ten datasets independently by dividing each into a training (80%) and a testing (20%) set using stratified sampling, so that the two sets have a comparable proportion of positives (valid traces) and negatives (invalid traces).

To mitigate randomization effects, inspired by Mills and Haiduc [7], we repeated the evaluation (training–testing set random, stratified splitting, 10-fold stratified cross-validation on the training set to tune the model's hyper-parameters, classifier training on the 80%, testing on the 20%) ten times, then we averaged the outputs, leading to the results in Section 6.1. Furthermore, we discuss the relative importance of the features in Section 6.2.

6.1. Quantitative results

Table 5 shows the average precision, the recall, and the $F_{0.5}$- and F_2-measure across the 10 runs for each dataset. The table compares the three algorithms (Random Forests, XGBoost, LightGBM) that are visualized as macro-columns.

The results for each project are presented in a different set of rows: each row refers to one of the four rebalancing strategies (none, oversampling, undersampling, 5050). The last macro-row reports the macro-average over the projects.

Trace recommendation (F_2). In this setting, where recall is weighted more than precision, the optimal rebalancing strategy is clearly 5050 (fourth row in the macro-rows), which generally leads the highest results. When comparing the algorithms with the 5050 rebalancing, both XGBoost and LightGBM outperform Random Forests. Interestingly, Random Forests have consistently been found to be the best performing algorithm in prior RTR research [7,8]. Here, in line with research from other fields [37,38], we find that Gradient Boosted Decision Trees can outperform Random Forests in RTR-tasks too.

Concerning our two implementations of gradient boosted decision trees, each obtains the best F_2 score in five cases. The differences between these two algorithms are marginal, with the largest difference being 3.8% for the Control_2 dataset. The macro-average shows that XGBoost reaches 61.45%, with LightGBM following closely: 60.94%.

To confirm these findings, we employ statistical tests for testing classifiers [41]: we execute Friedman's omnibus test that assesses whether there is a statistically significant difference between all the tested clas-

Table 5
Mean precision, recall, and $F_{0.5}$- (trace maintenance scenario) and F_2-measure (trace recommendation) across the ten datasets. The green-colored cells indicate the best results per each dataset. For accuracy and readability, the table shows F-scores in percentage.

Proj.	Rebal.	Random Forests				XGBoost				LightGBM			
		P	R	$F_{0.5}$	F_2	P	R	$F_{0.5}$	F_2	P	R	$F_{0.5}$	F_2
Company	None	78.78	39.31	65.41	43.63	74.35	48.89	**67.20**	52.42	71.95	46.39	64.71	49.89
	Over	69.46	48.06	63.70	51.17	64.58	57.36	62.84	58.55	61.97	57.78	61.00	58.50
	Under	17.65	86.94	20.99	48.64	21.59	91.39	25.48	55.46	20.86	90.42	24.64	54.16
	5050	52.49	53.89	52.67	53.50	50.47	67.36	53.07	63.03	54.78	66.67	56.66	**63.69**
Control_1	None	83.86	29.11	60.71	33.45	79.53	41.29	**66.91**	45.63	62.86	38.39	55.70	41.61
	Over	65.68	44.35	59.88	47.42	63.18	49.11	59.66	51.34	45.45	51.13	46.41	49.79
	Under	6.55	94.03	8.05	25.62	6.78	95.32	8.33	26.38	7.01	95.40	8.61	27.08
	5050	47.90	52.74	48.76	51.66	46.12	54.44	47.55	**52.51**	37.10	55.48	39.69	50.41
Control_2	None	74.69	28.50	55.86	32.43	70.76	48.25	**64.43**	51.39	65.73	42.50	58.67	45.47
	Over	59.91	44.75	55.94	47.03	64.24	61.25	63.58	**61.79**	54.33	57.75	54.90	56.94
	Under	8.85	94.25	10.81	32.09	9.88	93.00	12.02	34.61	9.40	94.00	11.47	33.53
	5050	50.41	55.50	51.26	54.29	51.37	62.25	53.16	59.60	44.89	59.75	47.12	55.81
Control_3	None	80.32	50.92	**71.63**	54.78	75.99	56.00	70.83	59.05	71.45	52.77	66.50	55.56
	Over	69.84	54.92	66.18	57.34	69.18	62.62	67.65	63.74	64.38	66.46	64.72	65.96
	Under	13.46	96.00	16.25	43.06	15.44	97.85	18.57	47.29	15.93	98.46	19.13	48.23
	5050	58.15	62.92	58.98	61.84	54.68	71.08	57.28	**66.99**	51.77	71.54	54.76	66.39
Data	None	89.13	29.91	63.46	34.43	84.18	61.73	**78.39**	65.16	82.02	58.45	75.81	61.97
	Over	75.00	45.27	66.20	49.13	75.62	69.27	74.16	70.38	73.01	71.09	72.57	71.43
	Under	16.04	91.73	19.21	47.12	19.79	95.36	23.51	53.97	19.11	96.36	22.75	53.20
	5050	67.74	57.27	65.34	59.09	64.14	75.55	66.09	72.89	62.68	76.18	64.92	**72.94**
Learn_1	None	82.73	15.79	44.56	18.83	80.65	44.45	**69.19**	48.78	54.84	39.15	50.62	41.44
	Over	59.86	40.30	54.50	43.09	68.24	51.52	64.01	54.14	53.31	55.61	53.72	55.10
	Under	5.49	92.80	6.76	22.18	6.68	94.09	8.21	26.02	6.58	96.46	8.09	25.84
	5050	53.03	47.24	51.19	47.79	52.86	59.94	54.12	**58.36**	46.18	60.67	48.45	57.01
Learn_2	None	78.69	23.79	53.42	27.59	59.14	31.03	49.20	34.05	59.61	30.69	49.56	33.83
	Over	66.19	42.76	**59.33**	45.88	57.67	45.86	54.64	47.68	56.68	41.72	52.43	43.79
	Under	6.04	83.10	7.41	23.38	7.28	92.41	8.92	27.60	8.05	93.45	9.85	29.89
	5050	47.81	42.76	46.46	43.50	44.79	48.28	45.15	47.25	46.68	48.97	46.93	**48.27**
Portfolio	None	77.42	35.32	62.39	39.60	74.15	46.23	**66.07**	49.96	68.39	42.27	60.78	45.73
	Over	64.94	43.44	59.03	46.49	58.77	52.92	57.42	53.93	48.26	59.22	50.03	56.52
	Under	9.48	91.49	11.55	33.50	13.95	95.45	16.82	43.98	13.86	96.75	16.73	44.03
	5050	50.50	53.70	51.04	52.96	46.20	61.49	48.60	57.63	40.69	66.17	44.06	**58.74**
Service	None	98.82	17.50	50.90	20.93	78.77	46.67	**69.02**	50.71	64.12	49.76	60.42	51.97
	Over	72.15	42.02	62.96	45.80	69.87	63.10	68.16	64.15	61.20	66.67	62.18	65.45
	Under	6.69	93.21	8.21	25.93	6.79	97.38	8.34	26.51	6.54	97.02	8.04	25.71
	5050	58.91	54.40	57.89	55.20	58.98	70.24	60.86	67.56	54.79	73.57	57.64	**68.69**
Store	None	87.84	37.06	68.25	41.76	82.10	57.65	**74.62**	60.75	59.20	52.35	56.73	52.37
	Over	71.06	47.06	63.88	50.17	76.79	65.29	74.05	67.21	78.89	58.24	72.51	60.88
	Under	4.11	91.76	5.07	17.26	2.56	91.76	3.18	11.38	3.41	85.88	4.22	14.69
	5050	65.39	50.00	61.22	52.22	61.27	71.18	62.80	**68.66**	58.22	70.59	60.13	67.43
MacroAvg	None	83.23	30.72	59.66	34.74	75.96	48.22	**67.59**	51.79	66.02	45.27	59.95	47.98
	Over	67.41	45.29	61.16	48.35	66.81	57.83	64.62	59.29	59.75	58.57	59.05	58.44
	Under	9.44	91.53	11.43	31.88	11.08	94.40	13.34	35.32	11.08	94.42	13.35	35.64
	5050	55.23	53.04	54.48	53.20	53.09	64.18	54.87	**61.45**	49.78	64.96	52.04	60.94

sifiers, followed by the Nemenyi's post-hoc test that determines which couples of classifiers lead to significantly different results. Furthermore, we measure effect size using Hedges' g, which corrects the results of Cohen's d for small sample sizes like ours [42]. We compare ten data points per classifier: the average performance (precision, recall, ...) of that classifier on a specific dataset across the ten runs.

The results in Table 6 confirm that, for F_2, the two implementations of gradient boosting are not statistically different, while LightGBM+5050 outperforms RF+5050. For precision, RF+5050 has a significantly higher precision than LightGBM+5050 with medium effect size. This is, however, compensated by the lower recall: both LightGBM+5050 and XGBoost+5050 outperform RF+5050 in a statistically significant manner and a large effect size.

Given the comparable performance of LightGBM and XGBoost, and the substantially faster speed of LightGBM, we conclude (RQ1) that LightGBM with the 5050 rebalancing strategy is the best combination for trace recommendation.

Trace maintenance ($f_{0.5}$). The superiority of gradient boosted decision trees is confirmed also in the trace maintenance scenario, with a few nuances. Unlike our preliminary results [33], where gradient boosting always outperformed random forests, in two cases (Control_3 and Learn_2) random forests have the highest $F_{0.5}$-score. Nevertheless, the XGBoost implementation obtains a much better $F_{0.5}$-score in all the other eight cases, with an average $F_{0.5}$-score of over 67% (no rebalancing), compared to 61% of random forests (with oversampling) and almost 60% of LightGBM (no rebalancing).

Regarding rebalancing, the 'none' strategy delivers the best results for XGBoost, while oversampling seems to be more effective for random forests. For LightGBM, no rebalancing is slightly more effective than oversampling, but the difference is only 1%.

XGBoost's average $F_{0.5}$-score is over 6% higher than the competing algorithms. However, in order to properly answer RQ1 (means are susceptible to outliers [41]), we statistically analyze four interesting combinations: (i) XGBoost+none and (ii) LightGBM+none, which have the highest mean $F_{0.5}$; (iii) random forests with oversampling, the highest $F_{0.5}$ for random forests; and (iv) random forests with no rebalancing, which delivers high precision. Table 6 reports the results of this statistical comparison.

Table 6
Statistical tests for trace recommendation and trace maintenance, using Friedman's omnibus test followed by Nemenyi's post-hoc. The values in green indicate statistical significance with $p \leq 0.05$. Effect size is reported using Hedges' g, and interpreted according to Fritz and colleagues [42]: *small* if $g \geq 0.2$, *medium* if $g \geq 0.5$, *large* if $g \geq 0.8$.

Metric	Omnibus		Post-Hoc		Effect size	
	χ^2	p	Comparison	Nemenyi	g	interp.
Trace recommendation						
F_2	12.20	0.002	RF+5050 vs. LightGBM+5050	0.010	-1.082	large
			RF+5050 vs. XGBoost+5050	0.005	-1.181	large
			LightGBM+5050 vs. XGBoost+5050	0.900	-0.061	none
Precision	9.80	0.007	RF+5050 vs. LightGBM+5050	0.005	0.691	medium
			RF+5050 vs. XGBoost+5050	0.261	0.299	small
			LightGBM+5050 vs. XGBoost+5050	0.261	-0.429	small
Recall	15.80	0.0004	RF+5050 vs. LightGBM+5050	0.001	-1.576	large
			RF+5050 vs. XGBoost+5050	0.010	-1.490	large
			LightGBM+5050 vs. XGBoost+5050	0.632	0.087	none
Trace maintenance						
$F_{0.5}$	9.96	0.019	RF+none vs. RF+over	0.900	-0.219	small
			RF+none vs. LightGBM+none	0.900	-0.035	none
			RF+none vs. XGBoost+none	0.110	-0.944	large
			RF+over vs. LightGBM+none	0.900	0.187	none
			RF+over vs. XGBoost+none	0.046	1.000	large
			LightGBM+none vs. XGBoost+none	0.029	-0.946	large
Precision	23.76	2.80×10^{-5}	RF+none vs. RF+over	0.001	2.456	large
			RF+none vs. LightGBM+none	0.001	2.203	large
			RF+none vs. XGBoost+none	0.160	0.974	large
			RF+over vs. LightGBM+none	0.900	0.203	small
			RF+over vs. XGBoost+none	0.160	-1.322	large
			LightGBM+none vs. XGBoost+none	0.160	-1.268	large
Recall	23.16	3.74×10^{-5}	RF+none vs. RF+over	0.005	-1.748	large
			RF+none vs. LightGBM+none	0.073	-1.472	large
			RF+none vs. XGBoost+none	0.001	-1.728	large
			RF+over vs. LightGBM+none	0.799	0.003	none
			RF+over vs. XGBoost+none	0.507	-0.408	small
			LightGBM+none vs. XGBoost+none	0.110	-0.331	small

The results indicate that the four classifiers are statistically different (with $p < 0.05$). The results for $F_{0.5}$ confirm that XGBoost+none statistically outperforms both LightGBM+none and RF+oversampling with a large effect size, while it does not statistically outperform RF+none, although this is probably due to the low number of samples; indeed, the effect size is large indicating a visible difference of almost one standard deviation. An interesting finding concerns precision: RF+none, with a mean precision of 83.23%, statistically outperforms RF+over and LightGBM+none (large effect size). Therefore, RF+none could be a suitable option when precision is much more important than recall. Indeed, the increased precision comes at the expense of recall: both RF+over and XGBoost+none are significantly better than RF+none for recall with large effect size. Since we use $F_{0.5}$ for trace maintenance, we recommend XGBoost+none as the best combination for this scenario.

On rebalancing. No rebalancing seems to be the winning option when optimizing for $F_{0.5}$: in 20 of the 30 cases of Table 5, this leads to the best results. On the other hand, the 5050 strategy, which combines under- and over-sampling, is the most effective one for F_2: in 26/30 cases.

The 'pure' strategies, i.e., under- and over-sampling, are less effective for the metrics that we used. Nevertheless, they may be useful in certain contexts. For example, in the trace recommendation scenario, recall is much more important than precision [32] (perhaps more than just F_2): in that case, undersampling could be an option. The last macro-row of Table 5 shows that undersampling leads to a recall of 91%, 94%, and 94% for random forests, XGBoost, and LightGBM, respectively. This has a huge impact on precision, which is around 10%. This would mean that, on average, the developer would be presented ten possible target artifacts, only one of them being valid, and that in circa 5%–10% of the cases, the valid target artifact would not be listed.

6.2. Feature importance

We analyze the feature importance with the goal of understanding which features deliver the highest information gain [43] to distinguish the positives from the negatives.

We consider the average gain for each of the feature families defined in Section 5.2, with QQ broken down into its three subcategories due to the many features. The cumulative (total), max, and average gain is shown in Table 7,[3] while Fig. 3 presents them visually.

Trace recommendation. The total column in Table 7 and the plots of Fig. 3(a) show that the process-related features contribute the most to information gain: 64.55% on average, from 53.95% (Company) to 72.55% (Service). The second best contributing family is QQ-specificity (average 20.32%), from 10.21% (Service) to 29.17% (Store). Nevertheless, this gain arises from the high number of features: the average gain per feature (0.28%) is lower than that of document statistics (0.38%) and it is comparable to term relatedness and similarity.

When looking at the ten features with the highest information gain (data in the online appendix), we see that feature F4: the difference between the issue resolution date and the revision date is always the best, with an average gain (Max column in Table 7, process-related, trace recommendation) of 51.27%, reaching 68.73% for Control_3. This is therefore a very important predictor for the trace recommendation scenario. Feature F3 (the difference between the issue's last update date and the revision date) also occurs in all ten datasets, but with a significantly lower gain: 6.86%. Other features with relatively high

[3] Errata: we presented similar results for three datasets in Table 5 of [33]. However, both in the table and in the interpretation, we had erroneously inverted over- and under-sampling.

Table 7
The total, average, and maximum gain (in percentage over the total gain) per feature family for trace recommendation (LightGBM+5050) and maintenance (XGBoost+none).

		Recommend.			Maintenance		
		Total	Avg	Max	Total	Avg	Max
Process-related	Company	53.95	13.49	48.83	6.57	1.64	2.71
	Control_1	63.90	15.97	46.83	5.79	1.45	2.90
	Control_2	63.59	15.90	56.79	4.88	1.22	1.94
	Control_3	71.92	17.98	68.73	5.36	1.34	3.56
	Data	69.04	17.26	29.57	9.98	2.50	3.71
	Learn_1	67.88	16.97	59.82	7.91	1.98	4.36
	Learn_2	54.56	13.64	41.79	4.32	1.08	1.88
	Portfolio	66.03	16.51	63.47	4.91	1.23	3.27
	Service	72.55	18.14	42.42	10.66	2.67	3.87
	Store	62.10	15.53	54.49	5.37	1.34	2.43
	Average	*64.55*	*16.14*	*51.27*	*6.58*	*1.65*	*3.06*
Document statistics	Company	0.79	0.11	0.27	5.08	0.73	1.15
	Control_1	1.35	0.19	0.47	6.94	0.99	1.90
	Control_2	1.12	0.16	0.59	10.84	1.55	3.32
	Control_3	2.19	0.31	1.41	8.43	1.20	1.91
	Data	0.74	0.11	0.24	4.43	0.63	0.82
	Learn_1	1.46	0.21	1.00	6.21	0.89	1.99
	Learn_2	1.22	0.17	0.30	7.12	1.02	2.09
	Portfolio	1.00	0.14	0.43	4.86	0.69	1.14
	Service	1.05	0.15	0.49	10.08	1.44	3.04
	Store	0.24	0.03	0.10	3.98	0.57	1.24
	Average	*1.12*	*0.16*	*0.53*	*6.80*	*0.97*	*1.86*
Information retrieval	Company	6.26	0.35	1.64	26.83	1.49	8.14
	Control_1	5.48	0.30	0.76	23.49	1.31	2.35
	Control_2	3.48	0.19	0.62	19.69	1.09	4.52
	Control_3	1.04	0.06	0.18	14.69	0.82	2.15
	Data	11.45	0.64	5.36	20.43	1.13	2.01
	Learn_1	10.18	0.57	3.18	24.01	1.33	3.28
	Learn_2	12.77	0.71	5.41	26.05	1.45	5.21
	Portfolio	3.42	0.19	1.21	24.24	1.35	6.87
	Service	12.90	0.72	5.71	15.55	0.86	1.72
	Store	1.91	0.11	0.79	16.04	0.89	3.80
	Average	*6.89*	*0.38*	*2.49*	*21.10*	*1.17*	*4.01*
QQ-Specificity	Company	27.85	0.39	3.60	47.70	0.66	2.22
	Control_1	20.85	0.29	3.43	50.10	0.70	2.71
	Control_2	25.02	0.35	5.78	46.43	0.64	3.06
	Control_3	15.37	0.21	1.71	53.65	0.75	4.11
	Data	14.43	0.20	1.17	47.92	0.67	3.14
	Learn_1	15.60	0.22	1.50	46.50	0.65	1.73
	Learn_2	20.92	0.29	2.91	45.88	0.64	1.72
	Portfolio	23.73	0.33	5.95	51.56	0.72	4.00
	Service	10.21	0.14	1.16	47.14	0.65	1.40
	Store	29.17	0.41	7.82	60.46	0.84	7.49
	Average	*20.32*	*0.28*	*3.50*	*49.73*	*0.69*	*3.16*
QQ-Similarity	Company	6.70	0.37	2.39	6.51	0.36	1.24
	Control_1	5.94	0.33	2.21	6.96	0.39	0.81
	Control_2	5.21	0.29	1.67	8.04	0.45	1.28
	Control_3	5.68	0.32	2.21	5.23	0.29	0.82
	Data	3.07	0.17	1.46	9.24	0.51	2.04
	Learn_1	2.18	0.12	0.91	7.36	0.41	0.91
	Learn_2	5.01	0.28	1.53	10.21	0.57	2.96
	Portfolio	3.51	0.20	1.35	7.36	0.41	1.28
	Service	1.28	0.07	0.19	9.04	0.50	1.60
	Store	3.37	0.19	1.78	6.76	0.38	1.26
	Average	*4.20*	*0.23*	*1.57*	*7.67*	*0.43*	*1.42*
QQ-Term-Relatedness	Company	4.45	0.37	1.19	7.31	0.61	1.50
	Control_1	2.49	0.21	0.71	6.72	0.56	1.03
	Control_2	1.59	0.13	0.56	10.12	0.84	3.54
	Control_3	3.79	0.32	2.51	12.65	1.05	7.28
	Data	1.26	0.10	0.29	8.00	0.67	1.30
	Learn_1	2.69	0.22	0.63	8.01	0.67	0.83
	Learn_2	5.52	0.46	1.58	6.42	0.54	1.42
	Portfolio	2.31	0.19	0.46	7.06	0.59	1.16
	Service	2.00	0.17	0.72	7.53	0.63	1.76
	Store	3.22	0.27	1.67	7.40	0.62	1.63
	Average	*2.93*	*0.24*	*1.03*	*8.12*	*0.68*	*2.15*

importance are F1: whether the issue assignee is the committer (4.83%) and F73: the standard deviation of the entropy of the SVN log as a query (2.54%). All other features have an information gain below 2%.

Trace maintenance. While process-related features, and especially F4, are predominant for trace recommendation, the situation is less clear for trace maintenance. First, when visually inspecting the plots in Fig. 3(b), the process-related features stand out much less prominently. Although they still have the highest average gain (1.65% in Table 7 in the Avg column of Process-related, Maintenance), no feature stands out the same way as in trace recommendation. This may be because XGBoost creates more complex decision trees than LightGBM.

Analyzing feature family gain, QQ-Specificity leads with 49.73%, ranging from 45.88% (Learn_2) to 60.46% (Store). The second family is information retrieval: total gain of 21.10%, average gain per feature of 1.17%. Although the total gain of these two families is good, the average gain is low. The individual feature with the highest gain is still F4, but with a limited contribution (2.97%). Only other two features are above 2%: F29 (2.28%): similarity based on VSM of all the text in the revision and in the issue (see last row of Table 3), and F23 (2.17%): similarity based on VSM based on unit names and JIRA description.

Given the limited information gain of each feature (1%–2%), it is difficult to draw solid conclusions on which features are the most important for trace maintenance. In this case, we see how a highly sophisticated classifier like XGBoost achieves better performance than other models at the expense of interpretability.

7. What is the value of MDD-specific features (RQ2)?

RQ2 examines the value of features that use MDD-specific information. One unique aspect of revisions in MDD is the granularity at which changes are stored. Other version control system store changes such as which lines of which files were modified. MDD provides readily-available, finer-grained information, as it describes changes occurring at the unit level (e.g., entity A in the domain model was deleted and entity B was added) rather than at the file level. We examine if making use of this granularity level affects the classifier's performance.

To address RQ2, we prune from the 131 features those that are specific to the MDD domain, leading to a subset that do not use MDD-specific information.

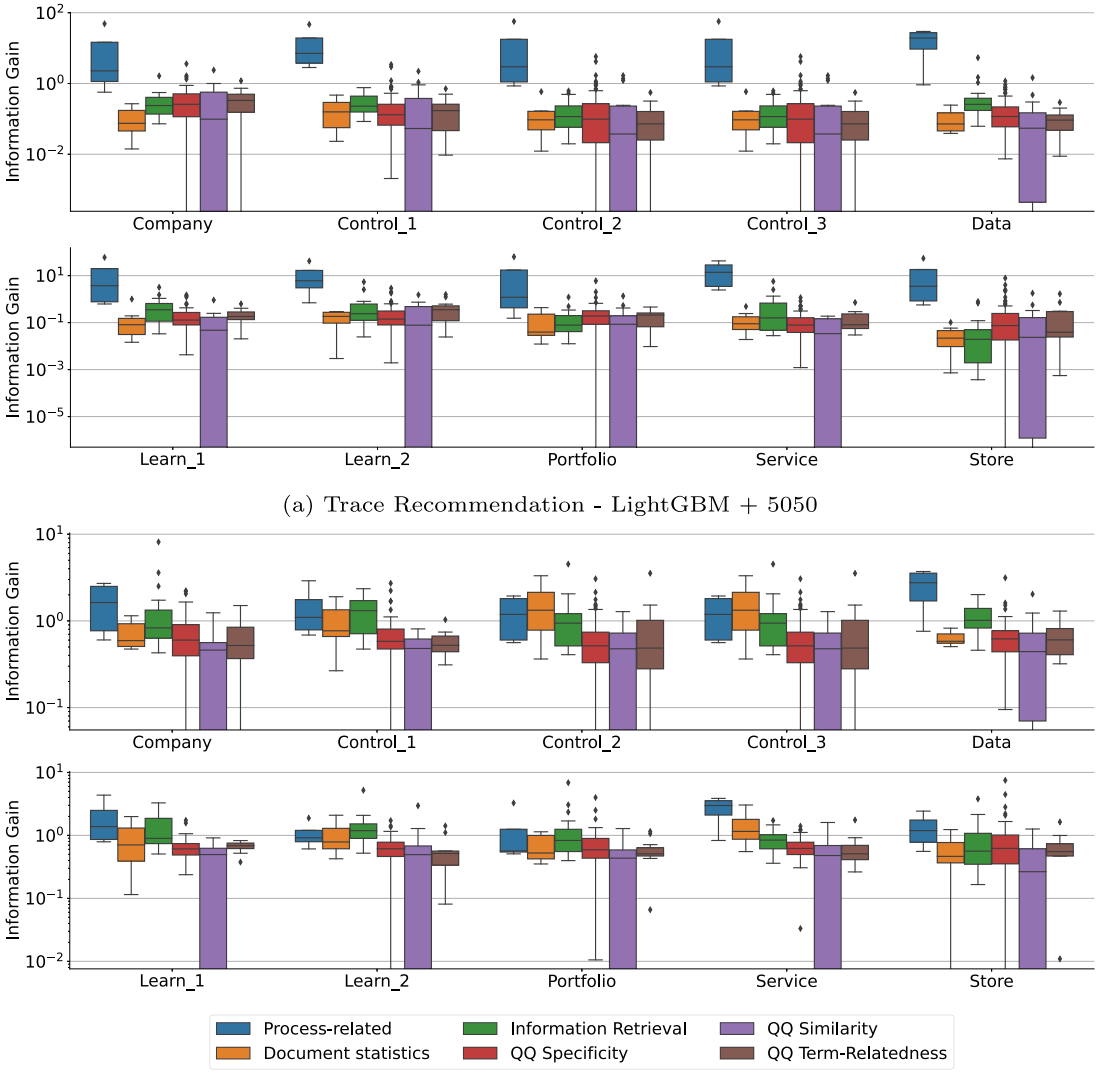
By MDD-specific information, based on our feature engineering, we refer to the use of *unit names* in the calculation of the value for a feature. Note that our strategy for deriving MDD-agnostic features is dependent on how we use MDD information in LCDTrace.

Table 8 summarizes the features that we altered or deleted:

- *Modified*: two features of the document statistics family (F6, F8) use the unit names (R7) to calculate the number of terms in a revision. F6 and F8 are also used by F9–F11 to determine the overlap between the terms in the revision and those in the JIRA issue. These features are modified by using only the log message (R3).
- *Deleted*: we remove all those features that use unit names. For example, as visible in Table 3, F18–F29 all make use of unit names, either uniquely (F18–F23) or in combination with the log message (All Natural text features F24–F29).

7.1. Quantitative results

In line with Fig. 2, we compare the two sets of features (131 vs. 85) using only the best algorithms from RQ1: for trace recommendation, LightGBM+5050; for trace maintenance, XGBoost+none. The quantitative results are summarized in Table 9. To facilitate the reader in comparing the results, Table 9 presents the specific results with the



(a) Trace Recommendation - LightGBM + 5050

(b) Trace Maintenance - XGBoost + none

Fig. 3. Average gain per feature family for trace recommendation (a) and for trace maintenance (b). The y-axis uses an exponential scale to improve readability.

Table 8
Overview of the MDD-specific features that we deleted or modified to study RQ2.

Feature family	Count		Features
	Total	Non-MDD	
Process Related	4	4	–
Document Statistics	7	7	Modified: F6, F8–F11
Information Retrieval	18	6	Deleted: F18–F29
QQ-Specificity	72	48	Deleted: F30–F32, F36–F38, F48–F50, F54–F56, F66–F69, F74–F77, F90, F92, F96, F98
QQ-Similarity	18	12	Deleted: F102-F104, F108-F110
QQ-Term Relatedness	12	8	Deleted: F120-F121, F124-F125
Total	*131*	*85*	*Deleted: 46* *Modified: 5*

85 non-MDD features, while Table 9 presents the results for the 131 features that were shown in Table 5, plus the standard deviation.

The results do not reveal remarkable differences. When comparing the macro-average across the ten datasets, the F_2-score of LightGBM+5050 decreases only slightly: from 60.94% to 60.19%, with a comparable standard deviation; and the $F_{0.5}$-score with XGBoost+none decreases even less: from 67.59% to 67.18%. We observe similar small differences also for precision and recall.

To confirm our preliminary conclusion that the MDD-specific features do not improve performance, we statistically compare the two classifiers using the Wilcoxon Signed-Rank test, the non-parametric alternative to the paired samples T-Test that is recommended when the assumption of normality does not hold [41]. The results confirm that no statistically significant difference (with $p < 0.05$) exists when comparing the results for RQ1 with those for RQ2. For trace recommendation: precision $W = 18$, $p = 0.375$; recall $W = 22$, $p = 1.000$; F_2 $W = 21$,

Table 9
Mean and standard deviation for precision, recall, and F_2-measure (trace recommendation) and $F_{0.5}$- (trace maintenance) across the ten datasets, (a) for the non-MDD features (RQ2), and (b) for all 131 features (RQ1).

(a) Results for the 85 non-MDD features

Project	Recommendation: LightGBM+5050						Maintenance: XGBoost+None					
	Precision		Recall		F_2		Precision		Recall		$F_{0.5}$	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
Company	47.90	3.21	67.08	5.03	61.99	3.60	71.17	5.78	51.25	5.95	65.83	4.60
Control_1	38.80	2.73	58.47	5.67	53.05	4.60	74.81	4.46	42.82	3.26	64.97	3.17
Control_2	46.76	2.89	61.50	12.32	57.56	8.89	70.06	5.40	48.50	6.89	64.25	5.88
Control_3	51.73	3.60	77.54	3.34	70.41	2.29	76.32	5.61	58.46	7.18	71.76	5.32
Data	64.06	15.92	71.76	14.08	69.15	11.74	85.47	16.54	55.29	13.64	76.35	14.14
Learn_1	45.12	2.61	62.13	3.55	57.75	3.10	79.84	3.44	44.02	2.89	68.61	2.81
Learn_2	39.20	7.98	46.90	15.69	44.93	13.47	57.27	10.36	30.34	6.66	48.51	9.07
Portfolio	39.54	2.43	66.75	4.70	58.64	3.69	74.09	3.84	45.97	4.09	65.88	2.92
Service	48.10	4.53	68.81	3.58	63.24	3.08	78.41	3.38	50.36	6.76	70.38	4.07
Store	57.45	4.64	67.65	10.83	65.14	8.76	82.24	12.31	57.65	10.30	75.24	10.03
Avg	*47.87*	*5.05*	*64.86*	*7.88*	*60.19*	*6.32*	*74.97*	*7.11*	*48.47*	*6.76*	*67.18*	*6.20*

(b) Results for the 131 features (re-shown from Table 5 for comparison)

Project	Recommendation: LightGBM+5050						Maintenance: XGBoost+None					
	Precision		Recall		F_2		Precision		Recall		$F_{0.5}$	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
Company	54.78	4.50	66.67	6.64	63.69	4.93	74.35	7.35	48.89	2.43	67.20	4.87
Control_1	37.10	2.31	55.48	4.41	50.41	3.22	79.53	6.27	41.29	4.06	66.91	4.42
Control_2	44.89	6.18	59.75	11.27	55.81	9.06	70.76	7.45	48.25	7.64	64.43	6.58
Control_3	51.77	5.61	71.54	5.54	66.39	5.25	75.99	7.02	56.00	6.03	70.83	6.30
Data	62.68	3.63	76.18	5.03	72.94	3.85	84.18	3.74	61.73	5.34	78.39	3.58
Learn_1	46.18	3.42	60.67	3.31	57.01	2.53	80.65	3.95	44.45	3.62	69.19	2.28
Learn_2	46.68	8.83	48.97	8.57	48.27	7.86	59.14	11.36	31.03	11.26	49.20	11.42
Portfolio	40.69	2.41	66.17	5.60	58.74	4.03	74.15	6.94	46.23	3.12	66.07	5.05
Service	54.79	4.72	73.57	5.44	68.69	3.89	78.77	6.43	46.67	5.49	69.02	4.82
Store	58.22	7.95	70.59	9.20	67.43	7.96	82.10	10.35	57.65	12.34	74.62	6.27
Avg	*49.78*	*4.95*	*64.96*	*6.50*	*60.94*	*5.26*	*75.96*	*7.09*	*48.22*	*6.13*	*67.59*	*5.56*

Table 10
Feature importance, reported in percentage and divided by feature family, for the non-MDD specific features (RQ2).

Feature family	Count	Trace recommendation				Trace maintenance			
		Sum-Avg	Mean-Avg	Std.-Avg	Max	Sum-Avg	Mean-Avg	Std.-Avg	Max
Process related	4	63.89	15.97	2.20	63.43	8.95	2.24	0.54	5.40
Document Statistics	7	1.99	0.28	0.14	1.88	8.19	1.17	0.33	4.23
Information Retrieval	6	7.46	1.24	0.87	19.90	10.12	1.69	0.27	4.15
QQ-Specificity	48	19.55	0.41	0.11	5.56	55.61	1.16	0.06	9.78
QQ-Similarity	12	4.27	0.36	0.15	4.99	7.80	0.65	0.25	2.00
QQ-Term Relatedness	8	2.84	0.35	0.23	2.85	9.34	1.17	0.43	14.71

$p = 0.5557$. For trace maintenance: precision $W = 12$, $p = 0.131$; recall $W = 22$, $p = 0.610$; $F_{0.5}$ $W = 21$, $p = 0.322$.

7.2. Feature importance

We also analyze whether the subset of features has an effect on the feature importance. The results are summarized in Table 10, organized by feature family. Similar to the previous results, feature importance seems to be generally aligned with the results for RQ1 in Table 7.

Trace recommendation (F_2). There is no visible difference: the process-related family remains the most important (63.89% for RQ2, 64.55% for RQ1), followed by QQ-specificity (19.55% for RQ2, 20.32% for RQ1). The importance of individual features confirms F4 as the most informative (51.10% gain, on average, reaching 64.43% on Control_3). F3 is still the second most informative (6.47%), followed by F12 (3.73%, reaching 19.90% on Store), F1 (3.17%), F2 (3.15%), and F14 (2.07%). All the other features have gains of less than 2%. The information gain calculations and values can be found in the online appendix.

Trace maintenance ($f_{0.5}$). We see some differences when considering the feature families. QQ-Specificity is still the leading family, going up from 49.73% to 55.61%. On average, however, each feature in QQ-Specificity has a low information gain (1.16%). Information retrieval decreases from 21.10% to 10.12% as a result of the 12 dropped features, with a slightly higher average gain per feature (1.69%), which, however, can be due to the lower total number of features. Small increases also occur for process-related and document statistics features, but, again, the total gain is split across 85 features rather than 131. By analyzing the gain of the individual features, only few are above 2%: two features from the process-related family (F4: 3.73%, F1: 2.18%), the total number of terms in JIRA (F5: 2.30%), two information-retrieval features (F14: 2.13%, F17: 2.26%), F61 and F101 from QQ-Specificity (2.67% and 2.16%), F129 from QQ-Term relatedness (2.52%, reaching 14.71% on Control_3). The limited gain, however, does not allow us to label any feature as highly informative.

8. Can we reduce feature numerosity (RQ3)?

RQ3 focuses on reducing the number of features while maintaining comparable performance as with all the 131 features of LCDTrace (RQ1). To do so, we conduct an experiment (see Fig. 2) using the state-of-the-art mRMR automated feature selection algorithm. Like for RQ2, we consider our ten datasets and we use the best combinations of classifier and rebalancing strategy from RQ1.

Using mRMR [13], we assess the effect of smaller features sets on our ten datasets, employing feature sets of sizes 40, 50, and 60.

Preliminary experimentation showed promising results in terms of keeping performance when using only half our original feature set. This made a set size of 60 our starting point, while opting for even smaller sets as well. mRMR comes with various functions to determine feature relevance. We use the FCQ scheme which combines the F-statistic with (Pearson) correlation, one of the more computational efficient variants under the mRMR framework, that showed excellent results at Uber [14]. mRMR selects an optimal feature set by retaining features with maximum relevance and by removing false positives (redundant) ones. A reduction in the size of the feature set directly impacts the computational complexity of our model. The aim is to improve the effectiveness and efficiency of the model, while also increasing the interpretability [14], as the analyst can better understand the model predictions given the lower number and non-redundancy of the features.

8.1. Quantitative results

The results of both trace scenarios are presented in Table 11, using a similar visualization to Tables 5 and 9. Unlike those tables, the results are shown for each value of K: 40, 50, and 60. Then, Table 12 presents the results of statistical analysis using Wilcoxon's Signed-Ranks tests, comparing the results from automated feature selection with those of the 131 features of RQ1.

Trace recommendation (F_2). With the three feature set sizes, we see small differences in precision, recall, and F_2. The last row in the macro-rows of Table 11 shows fluctuations in the ranges of 2.93%, 0.38%, and 1.31% for the three metrics. When inspecting the average precision of the individual projects, we observe how Control_2, Control_3, Service and Store are the main contributors to the lower macro-average compared to RQ1. This difference in *precision* is statistically significant, as shown in Table 12, with medium to large effect sizes.

The macro-average of recall shows a similar deviation between the three set sizes, with a small delta of 0.38% between the best and the worst value. When considering the individual projects, there are larger fluctuations depending on K; the largest difference pertains to the Store dataset, ranging from 62.35% (K=40) to 72.35% (K=60). Interestingly, the lower precision compared to RQ1 leads to a statistically significant increment in *recall*, with medium effect size (see Table 12), with the classifiers from RQ3 outperforming LightGBM+5050 from RQ1.

The opposite results for precision (decreased) and recall (increased) seem to balance out each other, leading to the absence of significant differences in F_2, indicating that the reduced feature sets do not affect our reference metric.

Moreover, the distribution of green cells (best values) in Table 11, and the significance results in Table 12, indicate that no best value for K can be determined for trace recommendation.

Trace maintenance ($F_{0.5}$). The results are even more stable than those for trace recommendation. While comparing the different set sizes, the macro-averages of precision, recall and $F_{0.5}$ range between 0.10%, 0.40%, and 0.36%, respectively. This suggests that the choice of K has a smaller effect on the classifier algorithm, or that a smaller subset of the features present in the subsets (K=40, K=50, K=60) contribute the most to the performance.

When we compare the results to those of RQ1, we see that for all projects, there is always a feature subset that outperforms the complete set. This is visible through the Hedges' g score for $F_{0.5}$ in Table 12, which is negative although it does not reach a visible effect size. For both scenarios, although there is no statistically significant gain in the relevant metrics, the results confirm that it is possible to reduce feature numerosity without compromising performance, thereby answering RQ3 positively.

8.2. Feature importance

We analyze feature importance to assess whether there is an effect on the interpretability of the ML models.

Table 13 shows the average number of features (count), average sum, mean, standard deviation and maximum gain, with the aggregate results of the ten datasets.

Trace recommendation. The Sum-Avg column in Table 13 shows that process-related features are the largest contributor to information gain, ranging between 59.68% and 66.77% across the values of K. This feature family is followed by information retrieval-related (IR) and QQ-Specificity-related features, which score similarly (13.09%–16.03% for IR, 13.65%–17.75% for QQ-Specificity). Note that IR-related features show a higher maximum score than QQ-Specificity, indicating their effectiveness for some projects.

The online appendix includes details on the individual feature gain. With each value of K, F4 is the dominant feature (45.34% with K=40, 49.14% with K=50, 48.91% with K=60), followed by F1 (7.37%, 7.06%, 6,24%), F3 (6.22%, 6.21%, 8.10%), F2 (5.22%, 4.08%, 3.54%), and F12 (5.22%, 3.42%, 2.40%). All other features are below 2%.

When comparing these results to the results obtained in Section 6.2, we see similar results. For both the 131 feature set and the feature subsets, feature F4 is dominant, followed by feature F1 (indicating if the assignee is equal to the committer). Looking at the results from Tables 7 and 13, we can see that for both, process-related features contribute the most to information gain. The results of all feature subsets does show improvements over the complete feature set in terms of the aggregate average (mean-Avg) and max (Max) gain: 16.14%, 51.27% for RQ1; and 17.69%, 65.57% for RQ3.

Trace maintenance. For trace recommendation, the obvious contributor to information gain (process-related) was clear from Table 13. For trace maintenance, the differences are less evident (see Sum-Avg in Table 13, Trace Maintenance). IR-related features (29.12% to 32.19%) and QQ-Specificity (34.92% to 41.09%) contribute the most. While the Sum-Avg indicates that the latter is the largest contributor, columns Mean-Avg and Max provide a more specific insight: on average, the IR features (2.09%, 10.78%) contribute more than the QQ-Specificity features (1.99%, 8.58%).

It is harder to determine which individual feature contributes the most to the gain. Again, F4 has the highest value, but the gain is ten times lower than for trace recommendation: 4.46%, 4.06%, and 3.61% for K=40, K=50, and K=60, respectively. The other features are scattered, with a different dominant feature (F1, F10, F19, F25, F29, F62, F129) for various datasets. F1, F10 (overlap of terms between JIRA issue and revision), F25 (revisions All Natural text × JIRA summary) and F29 (revisions All Natural text × JIRA All Natural text) recur more often (≥ 4 projects) in the top-10 features. However, the individual gain is just above 2%.

Comparing the results for trace maintenance with those from RQ1, we can see a different order of the feature families. Though process related features still contribute with the highest information gain, IR becomes the second contributor for RQ3, and therefore switches places with document statistics compared to the results in Section 6.2.

There are no strong effects depending on the choice of K. Still, Table 13 shows large differences between the different sizes in terms of maximum values (e.g., 13.42% and 40.79% for IR in the trace recommendation scenario). Also, compared to RQ1, there are no big shifts between feature families. However, the efficiency of the model still benefits from the feature numerosity reduction.

9. Threats to validity

We discuss the major threats to validity organized according to the taxonomy by Wohlin and colleagues [44].

Table 11
Mean and standard deviation for precision, recall, and F_2-measure (trace recommendation) and $F_{0.5}$- (trace maintenance) across the ten datasets, feature subset sizes K=40, K=50, K=60 (RQ3). The colored cells highlight which value of K leads to the highest metric for each of the projects in either scenario. We use green for trace recommendation, orange for trace maintenance. The last sub-table repeats the results from Table 9.

Project K = 40	Recommendation: LightGBM+5050						Maintenance: XGBoost+None					
	Precision		Recall		F_2		Precision		Recall		$F_{0.5}$	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
Company	42.63	3.72	70.56	6.20	62.28	4.75	72.04	5.16	51.94	2.94	66.85	4.41
Control_1	31.71	2.15	67.10	2.27	54.81	2.13	77.66	5.86	45.32	4.04	67.79	4.06
Control_2	41.58	4.21	63.50	8.60	57.24	6.56	72.47	9.57	48.75	7.00	65.90	8.24
Control_3	41.81	3.07	80.15	4.38	67.70	3.85	70.31	7.14	56.92	5.08	66.91	4.82
Data	54.26	3.50	82.27	2.91	74.52	2.74	85.96	5.18	63.27	3.16	80.10	3.43
Learn_1	35.04	2.95	67.32	3.48	56.76	2.92	81.76	2.91	48.29	2.63	71.77	2.59
Learn_2	43.35	4.44	55.17	10.53	52.11	8.23	57.97	9.32	42.07	7.23	53.71	8.24
Portfolio	27.26	0.97	80.65	3.72	57.91	1.93	77.09	4.14	47.99	4.49	68.66	3.87
Service	40.23	3.52	72.62	5.83	62.43	4.25	81.96	7.18	48.33	3.60	71.75	4.55
Store	50.74	8.46	62.35	12.15	59.34	10.17	82.64	11.66	54.71	9.63	74.47	9.33
Average	*40.86*	*3.70*	*70.17*	*6.01*	*60.51*	*4.75*	*75.99*	*6.81*	*50.76*	*4.98*	*68.79*	*5.35*

Project K = 50	Recommendation: LightGBM+5050						Maintenance: XGBoost+None					
	Precision		Recall		F_2		Precision		Recall		$F_{0.5}$	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
Company	42.84	2.52	72.50	6.03	63.63	4.51	73.07	2.09	55.28	4.93	68.55	2.17
Control_1	32.61	1.85	65.24	2.91	54.29	1.45	77.21	4.22	43.23	5.55	66.46	3.83
Control_2	44.06	5.78	64.25	7.91	58.73	6.70	72.89	5.20	51.75	7.82	67.17	5.41
Control_3	44.51	4.54	80.92	4.30	69.41	3.81	71.01	2.75	55.54	6.30	67.11	2.67
Data	55.11	1.69	79.82	5.17	73.19	3.65	82.30	2.46	65.45	2.57	78.23	1.54
Learn_1	36.22	1.77	65.00	3.73	56.04	2.56	81.42	4.10	47.68	2.68	71.24	2.57
Learn_2	42.99	5.73	56.21	8.14	52.88	7.19	56.45	9.15	38.97	5.87	51.55	7.25
Portfolio	27.50	0.98	76.69	2.66	56.47	1.66	75.18	3.48	47.21	3.24	67.12	2.43
Service	47.03	3.19	72.14	3.18	65.14	2.91	84.43	4.56	48.21	3.47	73.35	3.91
Store	58.83	9.66	71.76	7.74	68.43	7.31	84.27	10.93	58.24	14.79	76.83	11.37
Average	*43.17*	*3.77*	*70.45*	*5.18*	*61.82*	*4.17*	*75.82*	*4.89*	*51.16*	*5.72*	*68.76*	*4.31*

Project K = 60	Recommendation: LightGBM+5050						Maintenance: XGBoost+None					
	Precision		Recall		F_2		Precision		Recall		$F_{0.5}$	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
Company	49.02	4.98	66.81	4.32	62.20	4.00	73.71	4.56	54.31	6.79	68.53	3.64
Control_1	31.96	1.67	63.87	2.40	53.21	1.81	77.43	4.73	45.32	3.70	67.74	3.94
Control_2	41.37	3.69	62.75	8.93	56.70	6.49	72.36	7.29	48.75	7.00	65.78	6.58
Control_3	46.27	4.37	77.85	5.09	68.39	4.32	68.33	3.98	55.23	8.38	65.03	4.53
Data	55.22	2.68	80.64	2.88	73.81	2.40	84.39	3.13	64.18	6.03	79.28	3.22
Learn_1	35.90	1.93	67.01	4.00	57.04	2.57	79.82	4.39	48.41	3.07	70.60	3.55
Learn_2	43.02	6.56	56.21	5.40	52.80	5.07	60.32	8.96	39.66	7.84	54.41	8.15
Portfolio	28.11	2.18	78.18	3.37	57.59	2.81	77.54	3.19	47.92	3.21	68.94	2.62
Service	49.84	4.27	75.00	3.17	68.06	3.18	80.77	5.78	52.38	4.99	72.63	3.70
Store	57.15	8.85	72.35	10.02	68.15	7.46	81.60	10.42	54.71	7.36	73.99	8.02
Average	*43.79*	*4.12*	*70.07*	*4.96*	*61.79*	*4.01*	*75.63*	*5.64*	*51.09*	*5.84*	*68.69*	*4.79*

All features (see Table 9)	Recommendation: LightGBM+5050						Maintenance: XGBoost+None					
	Precision		Recall		F_2		Precision		Recall		$F_{0.5}$	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
Average	*49.78*	*4.95*	*64.96*	*6.50*	*60.94*	*5.26*	*75.96*	*7.09*	*48.22*	*6.13*	*67.59*	*5.56*

Table 12
Statistical tests for RQ3: the Wilcoxon's Signed-Rank W test is used to compare the metrics of relevance between the automated feature selection and all the 131 features. Green-colored cells highlight statistically significant differences with $p < 0.05$. Effect size is reported via Hedges' g.

	K	Precision				Recall				F_2			
		W	p	Hedges' g		W	p	Hedges' g		W	p	Hedges' g	
Recommendation: LightGBM	40	0	0.002	1.058	large	8	0.049	-0.570	med.	26	0.922	0.055	none
	50	1	0.004	0.720	med.	2	0.006	-0.641	med.	17	0.322	-0.110	none
	60	0	0.002	0.647	med.	0	0.002	-0.589	med.	15	0.232	-0.105	none
	K	Precision				Recall				$F_{0.5}$			
		W	p	Hedges' g		W	p	Hedges' g		W	p	Hedges' g	
Maintenance: XGBoost	40	26	0.922	-0.003	none	6	0.027	-0.321	small	12	0.131	-0.158	none
	50	24	0.770	0.017	none	1	0.004	-0.339	small	12	0.131	-0.149	none
	60	27	1.000	0.045	none	7	0.037	-0.352	small	11	0.105	-0.148	none

Table 13
Feature importance, shown in percentage and divided by feature family, for feature subset sizes K=40, K=50, K=60 (RQ3).

Feature family	K	Count	Trace recommendation				Trace maintenance			
			Sum-Avg	Mean-Avg	Std.-Avg	Max	Sum-Avg	Mean-Avg	Std.-Avg	Max
Process Related	40	4	64.16	18.30	4.49	66.66	11.32	3.17	1.12	8.98
	50	4	59.68	16.51	5.84	64.97	9.88	2.73	0.90	7.91
	60	4	66.77	18.27	2.77	65.07	9.22	2.44	0.88	6.45
	Avg	*4*	*63.53*	*17.69*	*4.37*	*65.57*	*10.14*	*2.78*	*0.97*	*7.78*
Document Statistics	40	4	2.56	0.62	0.45	2.43	11.19	2.66	0.43	5.24
	50	5	3.29	0.72	0.80	4.59	9.95	2.22	0.48	5.77
	60	5	1.82	0.38	0.34	2.35	8.61	1.82	0.57	5.41
	Avg	*4*	*2.56*	*0.57*	*0.53*	*3.12*	*9.92*	*2.23*	*0.49*	*5.47*
Information Retrieval	40	13	16.03	1.25	0.45	40.79	32.19	2.40	0.52	11.97
	50	14	14.40	1.06	0.40	22.60	30.95	2.08	0.43	10.41
	60	16	13.09	0.84	0.31	13.42	29.12	1.79	0.41	9.96
	Avg	*14*	*14.51*	*1.05*	*0.39*	*25.60*	*30.75*	*2.09*	*0.45*	*10.78*
Query Quality (Specificity)	40	14	13.56	1.00	0.51	6.83	34.92	2.46	0.34	8.66
	50	20	17.75	0.82	0.47	7.65	38.80	1.93	0.25	8.72
	60	26	13.65	0.53	0.20	5.17	41.09	1.60	0.19	8.36
	Avg	*20*	*14.98*	*0.79*	*0.39*	*6.55*	*38.27*	*1.99*	*0.26*	*8.58*
Query Quality (Similarity)	40	3	2.08	0.87	0.78	6.23	4.57	1.68	0.78	3.78
	50	5	2.15	0.50	0.32	3.19	5.15	1.16	0.55	3.79
	60	7	2.44	0.39	0.27	3.77	6.06	0.91	0.45	3.44
	Avg	*5*	*2.22*	*0.59*	*0.46*	*4.40*	*5.26*	*1.25*	*0.59*	*3.67*
Query Quality (Term-relatedness)	40	2	1.61	0.54	0.55	2.25	5.81	2.35	2.86	16.99
	50	3	2.74	0.71	0.99	10.34	5.28	1.45	0.96	10.34
	60	4	2.24	0.58	0.68	6.67	5.90	1.41	0.96	15.47
	Avg	*3*	*2.20*	*0.61*	*0.74*	*6.42*	*5.66*	*1.74*	*1.59*	*14.27*

Conclusion validity. It refers to drawing correct conclusions about relations between treatment and outcome. We extended previous work [33] with seven additional datasets, leading to more credible conclusions about the effectiveness. Yet, these originate from a single company and were developed by three teams. Obviously, a more heterogeneous sample would be beneficial. To mitigate this threat, we conduct statistical analysis that minimizes reliance on assumptions, and we run each experiment ten times.

Internal validity. It regards influences that may affect the independent variable with respect to causality, without the researchers' knowledge. The datasets are created by teams who follow the development method outlined in Section 3. While we compared the common attributes, we excluded those that were used only by certain datasets, e.g., JIRA comments. Furthermore, it is possible that certain trace links were incorrect and some links were missing. However, we used the original datasets without performing any attempts to repair the datasets, which could have increased the bias. Consequently, we discarded those revisions that were linked to no JIRA issues.

Construct validity. It concerns generalizing the result of the experiment to the underlying concept or theory. The main threat concerns the research design: we approximate performance in the two scenarios via the $F_{0.5}$ and F_2-scores. Although our method is aligned with the state-of-the-art [8], in-vivo studies should be conducted for a more truthful assessment of the effectiveness, e.g., by deploying a system based on the algorithms and measuring the performance in use. Part of these studies could use the technique by Berry [32] to identify the most suitable F-score for the scenarios. Moreover, for RQ2 and RQ3, we continued the study only with the best combinations. Although these configurations were identified as the best-performing in RQ1, the results should be carefully analyzed and interpreted. In particular, the choice of LightGBM over XGBoost for trace recommendation, justified by its efficiency and the comparable F_2-score, did not allow us to examine the information gain per feature (family) for XGBoost in trace recommendation. Furthermore, we have not explored the performance of many other families of algorithms that could have lower performance but higher interpretability.

External validity. It regards the generalizability of the results to industrial practice. Our claims mostly hold for low-code development at Mendix. Although we collected projects from three teams, using more data (from Mendix, its clients, or the providers of similar MDD platforms) would be beneficial. Also, to minimize overfitting and enhance generalizability, we followed the standard practice of separating training and test set. Yet, we cannot argue that the performance scores would be comparable in other settings, although the better performance of gradient boosting and the effectiveness of automated feature selection are likely to hold with other industrial datasets, as these techniques have shown to be highly effective in various application domains of ML.

10. Conclusions and future work

In this study, we have investigated the effectiveness of ML techniques for the recovery of trace links in MDD, specifically, in low-code development. We focused on backward, vertical trace links from committed revisions (indicating model changes) to JIRA issues (representing requirements or bugs).

We empirically assess the effectiveness of state-of-the-art TLR approaches on ten datasets from seven projects provided to us by a low-code development platform provider: Mendix. Upon analyzing the MDD process at Mendix, we confirm the two scenarios for automated trace link recovery by Rath et al. [8]: trace recommendation and trace maintenance. Each requires a different performance score: F_2 for the former, $F_{0.5}$ for the latter.

RQ1: What is the most effective ML approach for trace maintenance and trace recommendation between revisions and JIRA issues in MDD contexts? To answer this question, we have constructed the LCDTrace classifier, which combines recent research [7,8] and uses 131 features from the literature organized in four families. In addition to random forests, which performed the best in earlier studies [7,45], we experimented with gradient boosted decision trees, a family of algorithms that performs excellently in other domains [37,38]. We explored twelve combinations of ML algorithm and rebalancing strategy (Section 6), leading to two choices: (i) for trace recommendation, the LightGBM implementation of gradient boosting with a combination of under- and over-sampling for rebalancing the data (F_2-score ~61%); and (ii) for

trace maintenance, the XGBoost implementation of gradient boosting with no rebalancing ($F_{0.5}$-score ~67%). For trace recommendation, we could not find statistical differences in the performance of LightGBM and XGBoost, so we suggest using LightGBM because of its speed. We also investigated feature importance, in order to understand which feature families or individual features have the highest information gain. For trace recommendation, process-related features are the most informative (64% in total, with feature F4 alone providing 51% of the gain). For trace maintenance, instead, we could not identify a feature with high predictive power: while query quality features have the highest total gain, this is better explained by their number than by their informativeness.

RQ2: What is the added value delivered by the features that use MDD-specific information? We repeated the experimental procedure of RQ1 on a subset of features: we removed 46 features that used MDD-specific information (making use of unit names in the delta between the committed models), and we modified 5 features by excluding that information while retaining the feature (see Table 8). We tested the best algorithms from RQ1, and the results did not show major differences between the classifiers trained with the RQ2 features and those from RQ1. The relative importance of the gain provided by the feature families changed (see Table 10 in Section 7.2), but this does not correspond to individual features gaining prominence compared to RQ1. Therefore, we conclude that the MDD-specific features do not provide additional performance when considering unit names in the models as MDD-specific information.

RQ3: Can we increase efficiency while keeping similar performance by reducing the number of features? Despite their effectiveness, thanks to their ability to combine many features in non-trivial ways, high-dimensional classifiers require longer training time, making them less suitable for practical use. We attempted to improve on this aspect via automated feature selection: we experimented with the efficient mRMR algorithm [13,14] that removes highly correlated features, and we trained the best classifiers from RQ1 with 40, 50, and 60 features. This study reveals two major results: (i) the performance with the features subsets is comparable to that of all 131 features; and (ii) there are very marginal differences in the three settings with a different number of features. Since the statistical comparison of RQ3 vs. RQ1 does not reveal significant differences in the scores of interest (Table 12), we answer RQ3 positively: it is possible to reduce feature numerosity without affecting performance.

Future work. More research is needed about which features to include in production environments and those that characterize well the MDD setting. Our experimentation with unit names (models or the elements therein) did not lead to any gain, as highlighted by our answer to RQ2. It will be important to analyze whether this is due to the limited informativeness of the features that make use of unit names, or by the specific ML algorithms that favor other features such as project-related ones. Furthermore, the ML-based approach needs to be compared to its deep learning counterpart, and also to non-learning approaches that rely on capturing the semantics of a change in the models. Studying additional datasets outside Mendix is a priority: although ten datasets allowed us to reach more solid conclusions than in earlier work [33], we still need to determine how well LCDTrace would work with other development teams and with different low-code platforms. Moreover, analyzing the performance of the tool in use is essential: while we based our analysis and discussion on F-measures, only a user study can reveal the actual quality of the recommended and recovered traces, that is, whether the developers who have to vet and use the traces find them useful and efficient. Finally, studying horizontal traceability, i.e., the existence of links between artifacts at the same abstraction level (e.g., between JIRA issues) is a relevant direction.

In the broad sense, this paper contributes to the literature on software traceability through an empirical study on industrial data, in a not-widely-explored domain, that determines the effectiveness of existing TLR techniques. We encourage other researchers to build on our research and to conduct additional studies where they compare alternative techniques to LCDTrace.

CRediT authorship contribution statement

Wouter van Oosten: Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing. **Randell Rasiman:** Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing. **Fabiano Dalpiaz:** Conceptualization, Methodology, Validation, Investigation, Writing, Supervision, Project administration. **Toine Hurkmans:** Conceptulization, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

A snapshot of the LCDTrace source code repository and a replication package are available in a Zenodo repository [34].

References

[1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, E. Merlo, Recovering traceability links between code and documentation, IEEE Trans. Softw. Eng. 28 (10) (2002) 970–983.
[2] T.W.W. Aung, H. Huo, Y. Sui, A literature review of automatic traceability links recovery for software change impact analysis, in: Proc. of ICPC, 2020, pp. 14–24.
[3] R. Oliveto, M. Gethers, D. Poshyvanyk, A. De Lucia, On the equivalence of information retrieval methods for automated traceability link recovery, in: Proc. of ICPC, 2010, pp. 68–71.
[4] A. Marcus, J.I. Maletic, Recovering documentation-to-source-code traceability links using latent semantic indexing, in: Proc. of ICSE, 2003, pp. 125–135.
[5] M. Borg, P. Runeson, A. Ardö, Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability, Empir. Softw. Eng. 19 (6) (2014) 1565–1616.
[6] D. Falessi, M. Di Penta, G. Canfora, G. Cantone, Estimating the number of remaining links in traceability recovery, Empir. Softw. Eng. 22 (3) (2017) 996–1027.
[7] C. Mills, J. Escobar-Avila, S. Haiduc, Automatic traceability maintenance via machine learning classification, in: Proc. of ICSME, 2018, pp. 369–380.
[8] M. Rath, J. Rendall, J.L.C. Guo, J. Cleland-Huang, P. Maeder, Traceability in the wild: Automatically augmenting incomplete trace links, in: Proc. of ICSE, 2018, pp. 834–845.
[9] S. Winkler, J. von Pilgrim, A survey of traceability in requirements engineering and model-driven development, Softw. Syst. Model. 9 (4) (2010) 529–565.
[10] J. Di Rocco, D. Di Ruscio, L. Iovino, A. Pierantonio, Collaborative repositories in model-driven engineering, IEEE Softw. 32 (3) (2015) 28–34.
[11] R.J. Wieringa, Design Science Methodology for Information Systems and Software Engineering, Springer, 2014.
[12] B. Ramesh, M. Edwards, Issues in the development of a requirements traceability model, in: Proc. of ISRE, 1993, pp. 256–259.
[13] C. Ding, H. Peng, Minimum redundancy feature selection from microarray gene expression data, J. Bioinform. Comput. Biol. 3 (02) (2005) 185–205.
[14] Z. Zhao, R. Anand, M. Wang, Maximum relevance and minimum redundancy feature selection methods for a marketing machine learning platform, in: Proc. of DSAA, 2019, pp. 442–452.
[15] F. Blaauboer, K. Sikkel, M.N. Aydin, Deciding to adopt requirements traceability in practice, in: Proc. of CAISE, 2007, pp. 294–308.
[16] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, E. Romanova, Best practices for automated traceability, Computer 40 (6) (2007) 27–35.
[17] B. Wang, R. Peng, Y. Li, H. Lai, Z. Wang, Requirements traceability technologies and technology transfer decision support: A systematic review, J. Syst. Softw. 146 (2018) 59–79.
[18] O. Gotel, J. Cleland-Huang, J.H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. Maletic, P. Mäder, Traceability fundamentals, in: Software and Systems Traceability, Springer, 2012, pp. 3–22.
[19] O.C. Gotel, C. Finkelstein, An analysis of the requirements traceability problem, in: Proc. of RE, 1994, pp. 94–101.
[20] D. Cuddeback, A. Dekhtyar, J. Hayes, Automated requirements traceability: The study of human analysts, in: Proc. of RE, 2010, pp. 231–240.

[21] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, S. Panichella, On the role of the nouns in IR-based traceability recovery, in: Proc. of ICPC, 2009, pp. 148–157.
[22] A. Abadi, M. Nisenson, Y. Simionovici, A traceability technique for specifications, in: Proc. of ICPC, 2008, pp. 103–112.
[23] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, R. Harshman, Indexing by latent semantic analysis, J. Am. Soc. Inf. Sci. 41 (6) (1990) 391–407.
[24] P. Heck, A. Zaidman, Horizontal traceability for just-in-time requirements: The case for open source feature requests, J. Softw. Evol. Process 26 (12) (2014) 1280–1296.
[25] A.D. Lucia, A. Marcus, R. Oliveto, D. Poshyvanyk, Information retrieval methods for automated traceability recovery, in: Software and Systems Traceability, Springer, 2012, pp. 71–98.
[26] H. Abukwaik, A. Burger, B.K. Andam, T. Berger, Semi-automated feature traceability with embedded annotations, in: Proc. of ICSME, 2018, pp. 529–533.
[27] D. Falessi, J. Roll, J.L.C. Guo, J. Cleland-Huang, Leveraging historical associations between requirements and source code to identify impacted classes, IEEE Trans. Softw. Eng. 46 (4) (2018) 420–441.
[28] J. Guo, J. Cheng, J. Cleland-Huang, Semantically enhanced software traceability using deep learning techniques, in: Proc. of ICSE, 2017, pp. 3–14.
[29] J. Lin, Y. Liu, J. Cleland-Huang, Information retrieval versus deep learning approaches for generating traceability links in bilingual projects, Empir. Softw. Eng. 27 (1) (2022) 1–33.
[30] M. Rath, P. Mäder, The SEOSS 33 dataset: Requirements, bug reports, code history, and trace links for entire projects, Data Brief 25 (2019) 104005.
[31] E. Umuhoza, M. Brambilla, Model driven development approaches for mobile applications: A survey, in: Proc. of MobiWIS, 2016, pp. 93–107.
[32] D.M. Berry, Empirical evaluation of tools for hairy requirements engineering tasks, Empir. Softw. Eng. 26 (6) (2021) 1–77.
[33] R. Rasiman, F. Dalpiaz, S. España, How effective is automated trace link recovery in model-driven development? in: Proc. of REFSQ, in: LNCS, 13216, Springer, 2022, pp. 35–51.
[34] W. van Oosten, R. Rasiman, F. Dalpiaz, T. Hurkmans, Online appendix of "On the effectiveness of automated tracing from model changes to project issues", 2023, http://dx.doi.org/10.5281/zenodo.7757275, Zenodo.
[35] M.F. Porter, An algorithm for suffix stripping, Program: Electron. Libr. Inf. Syst. 14 (3) (1980) 130–137.
[36] C. Mills, S. Haiduc, The impact of retrieval direction on IR-based traceability link recovery, in: Proc. of ICSE NIER, 2017, pp. 51–54.
[37] J. Yoon, Forecasting of real GDP growth using machine learning models: Gradient boosting and random forest approach, Comput. Econ. 57 (1) (2021) 247–265.
[38] A. Callens, D. Morichon, S. Abadie, M. Delpey, B. Liquet, Using random forest and gradient boosting trees to improve wave forecast at a specific location, Appl. Ocean Res. 104 (2020).
[39] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, LightGBM: A highly efficient gradient boosting decision tree, in: Proc. of NIPS, Vol. 30, 2017.
[40] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique, J. Artificial Intelligence Res. 16 (2002) 321–357.
[41] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[42] C.O. Fritz, P.E. Morris, J.J. Richler, Effect size estimates: current use, calculations, and interpretation., J. Exp. Psychol. [Gen.] 141 (1) (2012) 2.
[43] J.T. Kent, Information gain and a general measure of correlation, Biometrika 70 (1) (1983) 163–173.
[44] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer-Verlag, 2012, pp. 1–236.
[45] C. Mills, J. Escobar-Avila, A. Bhattacharya, G. Kondyukov, S. Chakraborty, S. Haiduc, Tracing with less data: Active learning for classification-based traceability link recovery, in: Proc. of ICSME, 2019, pp. 103–113.