Evaluating classifiers in SE research
the ECSER pipeline and two replication studies

Dell'Anna, Davide; Aydemir, Fatma Başak; Dalpiaz, Fabiano

DOI
10.1007/s10664-022-10243-1

Publication date
2023

Document Version
Final published version

Published in
Empirical Software Engineering

Citation (APA)
Dell'Anna, D., Aydemir, F. B., & Dalpiaz, F. (2023). Evaluating classifiers in SE research: the ECSER pipeline and two replication studies. *Empirical Software Engineering*, *28*(1), Article 3. https://doi.org/10.1007/s10664-022-10243-1

Empirical Software Engineering (2023) 28:3
https://doi.org/10.1007/s10664-022-10243-1

Evaluating classifiers in SE research: the ECSER pipeline and two replication studies

Davide Dell'Anna[1] · **Fatma Başak Aydemir**[2] · **Fabiano Dalpiaz**[3]

Accepted: 20 September 2022

Abstract

Context Automated classifiers, often based on machine learning (ML), are increasingly used in software engineering (SE) for labelling previously unseen SE data. Researchers have proposed automated classifiers that predict if a code chunk is a clone, if a requirement is functional or non-functional, if the outcome of a test case is non-deterministic, etc.

Objective The lack of guidelines for applying and reporting classification techniques for SE research leads to studies in which important research steps may be skipped, key findings might not be identified and shared, and the readers may find reported results (e.g., precision or recall above 90%) that are not a credible representation of the performance in operational contexts. The goal of this paper is to advance ML4SE research by proposing rigorous ways of conducting and reporting research.

Results We introduce the *ECSER* (Evaluating Classifiers in Software Engineering Research) pipeline, which includes a series of steps for conducting and evaluating automated classification research in SE. Then, we conduct two replication studies where we apply *ECSER* to recent research in requirements engineering and in software testing.

Conclusions In addition to demonstrating the applicability of the pipeline, the replication studies demonstrate *ECSER*'s usefulness: not only do we confirm and strengthen some findings identified by the original authors, but we also discover additional ones. Some of these findings contradict the original ones.

Communicated by: Markus Borg

✉ Davide Dell'Anna
d.dellanna@tudelft.nl

Fatma Başak Aydemir
basak.aydemir@boun.edu.tr

Fabiano Dalpiaz
f.dalpiaz@uu.nl

[1] Delft University of Technology, Delft, The Netherlands

[2] Boğaziçi University, Istanbul, Turkey

[3] Utrecht University, Utrecht, The Netherlands

Published online: 08 November 2022

Keywords Automated classification · Machine learning · Software engineering · Replication study

1 Introduction

The increasing adoption of machine learning (ML) and deep learning (DL) techniques in software engineering (SE) research have brought in research methods that SE researchers are not yet fully familiar with. In particular, statistical results have become a prevalent component of many SE papers.

Books such as Darrell Huff's "How to lie with statistics" (Huff 1993) helped to make the intricacies and pitfalls of statistical results part of popular culture. Many lay people understand the drawbacks of reporting only the arithmetic mean without variance: for example, the statement 'On average, students read two books per year' is not necessarily informative, since it could be drawn from very different populations such as $\{2, 2, 2, 2, 2\}$ and $\{0, 0, 0, 0, 10\}$.

When we bring this intuition to the machine learning for software engineering field (ML4SE), are we (*SE researchers*) able to recognize and avoid possible pitfalls when conducting, reviewing, and reading ML4SE research? Do we understand which results we can confidently derive from our research (think of spurious correlation versus causation), and do we disseminate our results in a fair manner? Also, can *SE practitioners* have confidence that the results they read will translate to similar performance in an operational setting, i.e., in the software industry?

Motivating Example

A software engineer searching for a solution for automatically identifying non-functional requirements from a list of requirements encounters a published work that reports the F_1-scores of two classifier models as 0.89 and 0.87 on the test data set, without releasing further details on the study. After reading the report and checking the related work, the software engineer raises several questions:

1. Are the hyper-parameters of the models fine tuned?
2. What are the values of other performance metrics, which the software engineer cares about, such as sensitivity, specificity, and accuracy?
3. Would the F_1-scores be the same with a different training-testing partitioning of the data set?
4. Is the difference between the performance of two models significant? The software engineer has just read that LightGBM's implementation of gradient boosted decision trees is up to 20 times faster than XGBoost's [45].
5. Would the models perform similarly as good on her private data set?

Could these questions have been avoided if the SE researchers conducted and reported their work on automated classifiers in a different way?

The SE research community is increasingly aware of these challenges, and some researchers have started coping with them. For example, de Oliveira Neto et al. (2019) analyzed the predominant practices in empirical SE and proposed a conceptual model for a statistical analysis workflow. Kitchenham et al. (2017) discussed the importance of properly analyzing non-normally distributed data, which are common in SE data. Mahadi et al.

(2022) studied the effectiveness of classifiers when applied across projects, and they showed the instability of the conclusion validity results.

In a broader context, the SIGSOFT empirical standards (Ralph et al. 2020) are emerging as a response to the variety of research methods employed in SE, and to the difficulties of authors and reviewers when reporting and assessing research. We align with this perspective, and we focus on the provision of guidelines for *conducting and reporting ML4SE research*.

In this paper, we follow the research approach described in Section 2 to study how to rigorously conduct and report SE research that makes use of *automated classifiers*. A classifier is an algorithm that maps each element of a data set to one or more categories (classes) selected from a pre-defined list. Nowadays, the vast majority of classifiers employ ML and DL: they learn a classification model from a training set, then they use that model to predict the categories of a test set.

Our goal is to demonstrate the usefulness of following a simple pipeline that guides the researcher while conducting and reporting on the research results. In particular, we make the following contributions:

- We introduce the *ECSER* (Evaluating Classifiers in Software Engineering Research) pipeline for SE researchers to follow when conducting research with automated classifiers. *ECSER* adopts and consolidates recommendations from recent literature in ML and statistics, and customizes some of them—when necessary—to the context of SE research. *ECSER* is specifically aimed to assist SE researchers with limited ML background to avoid common mistakes and to present their results in a credible and correct manner.
- We conduct two replication studies—one in software testing, one in requirements engineering—by applying the *ECSER* pipeline. In doing so, we illustrate *ECSER* and we demonstrate its applicability and usefulness. The replications through *ECSER* allow us to strengthen some of the conclusions that the original papers had made, and also to identify additional findings, some of which contradict the original results.
- As part of *ECSER*, we include the metrics of *overfitting* and *degradation* for assessing the expected operational performance of the classifiers. The aim of these metrics is that of providing a credible assessment of ML4SE research results for other researchers and practitioners.
- We make available a replication package (Dell'Anna et al. 2021) that the interested reader can use to apply our pipeline in order to learn about it and as a starting point for comparing their classifiers and/or their data sets.

The rest of the paper is structured as follows. Section 2 describes our research method. Section 3 discusses related work. Section 4 presents the *ECSER* pipeline. Section 5 applies *ECSER* to the classification of functional and quality requirements, while Section 6 applies it to test case flakiness prediction. Section 7 discusses the threats to validity; Section 8 concludes the paper by listing the findings and the limitations of *ECSER* and by outlining future work.

2 Research Approach

Triggered by the increasing use of classifiers in ML4SE research, and by our personal observation on the varying styles and depth of reporting on the effectiveness of classifiers in SE research, we set our main research question:

MRQ. How can we enable SE researchers to accurately conduct and report on the evaluation of automated classifiers?

To answer this question, we follow the three steps of the design cycle of Wieringa's design science research methodology (Wieringa 2014). First, we conduct *problem investigation* to discover the problems with the current situation, leading to our first research sub-question:

RQ1. What are the challenges with the current research practices with classifiers in SE research?

To identify these challenges, we conduct a review of the existing literature and we rely on our own observations and experience as researchers in the ML4SE field. The answer to this research question can be found in the challenges that are listed in Section 3.

The second step of Wieringa's design cycle is to *design a treatment* to improve the current situation. This step is mapped to our second research sub-question:

RQ2. What is an easy-to-use, tangible artifact that can assist SE researchers when conducting and reporting research on classifiers?

We answer this question by proposing *ECSER*, a pipeline for SE researchers to use when constructing and evaluating classifiers. The pipeline is detailed in Section 4. The design process is guided by existing ML literature and by our experience. As evidenced in Table 1, the design of *ECSER* is informed both by general literature on machine learning (specifically: classifiers) and by specific SE literature that made use of classifiers. We decided to create an ML4SE-specific pipeline because we could not find an explicit step-by-step process in the literature. The main steps are mostly a consolidation of the steps suggested by major machine learning textbooks (Sheskin 2020; Flach 2012; Bishop 2006). We pay special attention to statistical techniques for a robust comparison among multiple classifiers (Demšar 2006; Benavoli et al. 2016a, 2017b), a topic that has also been discussed by prominent ML4SE researchers (Menzies and Shepperd 2019). Significant discussion of the metrics (S7) occurs both in the general ML (Japkowicz and Shah 2011; Lever 2016) and in the ML4SE (Yao and Shepperd 2020) literature. The interested reader may find detailed descriptions of the topics in the papers and books listed in Table 1.

Third, Wieringa's design cycle suggests to perform *treatment validation* for the design artifact, which in our case leads to the following sub-questions:

RQ3. How applicable is the pipeline to ML4SE research?
RQ4. How useful is the pipeline when applied to ML4SE research?

To answer RQ3 and RQ4, we conduct two replication studies in different sub-fields of SE: (i) requirements engineering, via the classification of functional and quality requirements (Hey et al. 2020a; Kurtanovic and Maalej 2017; Dalpiaz et al. 2019); and (ii) software testing, via the detection of test case flakiness (Alshammari et al. 2021a; Pinto et al. 2020). These studies, reported in Sections 5 and 6, respectively, were selected because the authors provided ready-to-use replication packages, their classifier comparison represents well the current practice regarding classification in SE research, and these studies did not assess the operational performance of the classifiers on unseen data. These replications allow us not only to derive a number of lessons learned regarding the pipeline (the object of our study), but also to identify findings concerning the replicated studies that were not present in the original publications.

Table 1 Overview of the main ML and ML4SE-specific literature that we consulted during the design of *ECSER*, referring to the steps presented in Section 4

Step	General ML	ML4SE-specific
S1. Select an evaluation method and split the data	Flach (2012), Read et al. (2011), Japkowicz and Shah (2011)	Dalpiaz et al. (2019), Herbold et al. (2020)
S2. Train the model		
S5. Test the model		
S3. Hyper-parameters tuning and validation	Tran et al. (2020)	Fu et al. (2016), Tantithamthavorn et al. (2019), Liu et al. (2021), Agrawal et al. (2021)
S4. Re-train with optimized params		
S6. Report the confusion matrix	Flach (2012), Lever (2016)	Hall et al. (2012)
S7. Report metrics	Lones (2021), Adams and Hand (2000), Stapor (2017), Flach (2012), Sorower (2010), Japkowicz and Shah (2011), Lever (2016)	Yao and Shepperd (2020), Berry (2021), Cleland-Huang et al. (2010)
S8. Analyze overfitting and degradation	Cawley and Talbot (2010), Bishop (2006)	
S9. Visualize ROC	Boyd and Eng (2013), Goadrich et al. (2006), Lever (2016), Fawcett (2006), Flach (2012)	
S10. Apply statistical significance tests	Sheskin (2020), Demšar (2006), Benavoli et al. (2017b, 2016a), Salzberg (1997), Stapor (2017), Japkowicz and Shah (2011), Good (2013)	Menzies and Shepperd (2019)

3 Related Work

Software engineering is one of the many fruitful domains for machine learning applications. In the early 2000s, Menzies' handbook provided practical examples of the use of machine learning for software engineering problems (Menzies 2001). Zhang and Tsai (2003) listed the software engineering tasks that are powered by machine learning as *i.* prediction and estimation, *ii.* property or model discovery, *iii.* transformation, *iv.* generation and synthesis, *v.* reuse library and construction, *vi.* requirements acquisition, and *vii.* capture development knowledge.

Over the past 20 years, the application of ML techniques to SE problems has become increasingly prevalent. Supervised ML techniques for classification can be easily applied by non-experts using libraries such as scikit-learn (Pedregosa et al. 2011) or Weka (Hall et al. 2009). Similarly, deep learning frameworks such as TensorFlow (Abadi et al. 2015) enable their users to quickly build large-scale neural networks for machine learning tasks such as classification, among others.

The many available tutorials and code snippets allow using classifiers without fully understanding the differences between the algorithms, the validation and testing options, or how to interpret the results. Therefore, ML techniques and tools are often used without proper knowledge, and the lack of understanding of the underlying complexities of the ML models may lead to poorly reported results.

In the broader ML field, researchers warned about the possible negative consequences of uninformed classification applications and provided guidelines to follow. Salzberg (1997) lists what to avoid when comparing classifiers and recommends using multiple algorithms, a benchmark, cross-validation with parameter optimization within each fold, and the binomial test to assess statistical validity. Adams and Hand (2000) discuss the use of suitable metrics for a reliable assessment of classifier performance. Demšar (2006) focuses on comparing classifiers over multiple data sets in such a way to obtain statistical significance. Benavoli et al. (2017b) adopt Bayesian Analysis to compare classifiers, and they also argue (Benavoli et al. 2016a) for the use and selection of post-hoc tests based on mean-ranks when comparing classifiers. Stapor (2017) provides the basic steps of classifier evaluation and lists alternative approaches for each step. Herbold (2020) presents Autorank: a software for non-experts that automatically ranks classifiers based on their performance. These are just a few examples of the complexity of conducting a solid evaluation of a classifier's effectiveness.

Despite the variety of guidelines provided by the ML research community and by the numerous textbooks on ML and statistics (e.g., Flach 2012; Sheskin 2020; Japkowicz and Shah 2011), researchers and practitioners in domains such as biomedical research (Luo et al. 2016; Tanwani et al. 2009), combinatorial science (Siebert et al. 2020) and medicine (Alonso-Betanzos et al. 2015) have raised concerns about the use and reporting of ML techniques by non-experts. Luo et al. (2016), for instance, highlight that ML is often considered a "black magic" by scholars in biomedical research and that this often leads to difficulties in interpreting the reported results and to spurious conclusions, which can compromise the credibility of other studies and discourage researchers from adopting ML techniques. SE is not dissimilar from these domains, for all are heavily affected by the emergence of ML.

Challenge 1

The easy access to ML techniques and code snippets allows researchers to apply these techniques as a black box without being fully aware of the intricacies of configuring and evaluating classifiers.

To assess the situation within SE research, we conducted an exploratory mapping study (raw data in our supplementary materials (Dell'Anna et al. 2021)) of the proceedings of the International Conference on Software Engineering (ICSE) from the year 2019 through 2021. We aimed to identify those papers that use classifiers and are, therefore, conducting and reporting research on classifiers in SE.

To conduct this analysis, the three authors of this paper have independently analyzed one year of the ICSE proceedings by checking relevance through the title, abstract, and full text. We first looked at the title and we read the abstract when possibly relevant. If the abstract still indicated potential relevance, we checked the paper. Since our analysis is meant to explore the problem, we decided to rely on a single annotator per paper for this preliminary task.

For each relevant paper, we collected information regarding the (i) used *evaluation metrics*: precision, recall, accuracy, F-Score, ROC-AUC; an (ii) *explicit justification for the metrics*: no, related to previous work, yes, implicit in the type of study (e.g., the RQ

mentions accuracy); (iii) inclusion of the *confusion matrix*, which enables the reader to determine all other metrics; (iv) evaluation over *multiple data sets*, which is important for the generalization of the results; (v) *type of baseline* being used: none, own, external; and (vi) *analysis of statistical significance* of the obtained results.

Table 2 summarizes the findings resulting from the exploratory mapping study. The study confirms the popularity of machine learning, and in particular automated classification via machine learning, for software engineering research. Out of the 376 papers accepted in the technical track of ICSE, we have marked 60 as related to classification tasks (circa 16% of the accepted papers).

Unfortunately, the analysis also confirms that, similarly to other research fields, also in SE research, the essential details are often omitted or poorly reported in the evaluation of ML-based solutions, leading to hard-to-reproduce and sometimes misleading results. The analyzed papers followed various steps for their machine learning pipelines and reported their results in different ways. Among the 60 marked papers, primarily based on ML or DL, precision and recall are the most reported performance metrics (38 times each), followed by F-score (27) and accuracy (24). Only one-fourth of the papers (14/60) do explicitly justify their selection of the metrics. Many (20/60) refer back to the custom metrics in the field (thus, if previous authors use inadequate metrics, the problem propagates through the community) or, in the ML literature. Twenty-three studies do not provide any explanation at all. For three papers, the justification is implicit in the kind of study, e.g., the title mentions a study on accuracy and the selected metric is therefore apparent. The confusion matrix, which provides a comprehensive analysis of the performance of a classifier and can be used to compute most performance metrics (Flach 2012), is reported only in six papers.

Table 2 Summary of the exploratory mapping study of the proceedings of the ICSE conference from the year 2019 through 2021

Year		2019		2020		2021		Total	
Accepted ICSE papers (Main track)		109		129		138		376	
Papers related to classification		19	(17.43%)	14	(10.85%)	27	(19.57%)	60	(15.96%)
Evaluation metrics	Precision	15	(78.95%)	7	(50%)	16	(59.26%)	38	(63.33%)
	Recall	17	(89.47%)	6	(42.86%)	15	(55.56%)	38	(63.33%)
	Accuracy	7	(36.84%)	2	(14.29%)	15	(55.56%)	24	(40%)
	F-Score	9	(47.37%)	6	(42.86%)	12	(44.44%)	27	(45%)
	AUC	2	(10.53%)	3	(21.43%)	4	(14.81%)	9	(15%)
	ROC plots	2	(10.53%)	0	(0%)	1	(3.7%)	3	(5%)
Metrics justification	No	8	(42.11%)	7	(50%)	8	(29.63%)	23	(38.33%)
	Implicit	1	(5.26%)	1	(7.14%)	1	(3.7%)	3	(5%)
	Previous work	5	(26.32%)	5	(35.71%)	10	(37.04%)	20	(33.33%)
	Yes	5	(26.32%)	1	(7.14%)	8	(29.63%)	14	(23.33%)
Confusion matrix		1	(5.26%)	3	(21.43%)	2	(7.41%)	6	(10%)
Evaluation over multiple dat sets		9	(47.37%)	7	(50%)	20	(74.07%)	36	(60%)
Type of baseline	None	1	(5.26%)	1	(7.14%)	3	(11.11%)	5	(8.33%)
	Own	8	(42.11%)	4	(28.57%)	7	(25.93%)	19	(31.67%)
	External	5	(26.32%)	9	(64.29%)	13	(48.15%)	27	(45%)
	External and Own	5	(26.32%)	0	(0%)	4	(14.81%)	9	(15%)
Analysis of statistical significance		5	(26.32%)	0	(0%)	1	(3.7%)	6	(10%)

Challenge 2

For most ML4SE papers, it is unclear why a performance metric was selected, and it is difficult to verify the metrics or to calculate other metrics due to the omission of the confusion matrix.

Visualization of the results with receiver operating characteristic (ROC) plots is also quite unpopular: 3/60. Interestingly, even though 55 papers compare their results with an external baseline or with the authors' previous work, just six papers report the statistical significance of the results, a recommended practice by the machine learning community (Demšar 2006) for drawing solid conclusions.

Challenge 3

Statistical significance of the results is hardly analyzed: from this perspective, ML4SE research is behind other disciplines, e.g., social sciences or medicine, where statistical significance is a must.

To mitigate these and other challenges, researchers and practitioners of different fields outlined ML guidelines tailored for the particular domains (Wang et al. 2020; Luo et al. 2016; Greener et al. 2022). The intent and benefit of such guidelines are not only to make accessible to non-experts the scattered, dense, and non-trivial ML knowledge that is essential for conducting adequate research but also to present such knowledge via examples and case studies that are relevant for the particular domain, so to facilitate the transfer of knowledge. We follow the same idea, and to promote better practices in SE research, we devise guidelines for the conduction and evaluation of classifier research in SE.

Guidelines have already proven helpful for many areas of software engineering research. Jedlitschka et al. (2008), for example, provide detailed guidelines on planning and reporting controlled experiments for SE in a level of detail, including the title, keywords, variables, and the discussion of the experiments. Kitchenham (2004) list the tasks and sub-tasks for planning, conducting, and reporting systematic reviews targeting the SE researchers as the audience. Similarly, Kuhrmann et al. (2017) present guidelines on designing literature studies for SE based on the authors' experience. Each process step is identified, starting from preparation, and continuing with the data collection, study selection, and conclusion steps. Garousi et al. (2019) focus on reporting grey literature and conducting multivocal literature reviews for SE. Garousi and Felderer (2017) also share their guidelines for data extraction in systematic reviews based on their experience. Petersen et al. (2015) update their previous guidelines for conducting systematic mapping studies (Petersen et al. 2008) discovering the existing guidelines were insufficient and therefore providing additional guidelines to support SE researchers. Fagerholm et al. (2017) propose guidelines for using empirical studies in SE education covering learning outcomes, planning, scheduling, and use of empirical studies for SE research.

Concerning ML4SE, however, only limited support exists for SE researchers. The work of Agrawal et al. (2021), for example, discusses good practices for hyper-parameter optimization. Rajbahadur et al. (2021), instead, focus on the impact of the noise of the dependent variable introduced by discretization on classifiers. In the context of software analytics, Menzies and Shepperd (2019) present a list of "bad smells", a term used in the agile software community to denote surface indicators of deeper problems. Examples include the focus on statistical significance rather than effect size, lack of data visualization, dangers of overfitting, and partial reporting of results. Yao and Shepperd (2020) discuss the importance

of metric selection and the issues in using common metrics such as F_1-score. Clear guidelines for reporting classification-related SE research are currently missing and scattered in the literature. We argue that the lack of a standard way to report classification results that we noted in our exploratory mapping study is partly due to the lack of, and could be mitigated with, guidelines for reporting classification-related SE research.

4 *ECSER*: A Pipeline for Evaluating Classifiers in SE Research

We present *ECSER* (Evaluating Classifiers in Software Engineering Research), our pipeline for SE researchers to use when conducting and reporting on SE research that *evaluates* one or more classifier models (algorithms). *ECSER* was designed following the research method described in Section 2, in order to answer RQ2, starting from extensive research into the literature in ML and our own experience.

Figure 1 illustrates the ten steps of *ECSER*, which are organized into two macro-activities: (i) the training, validation & testing of the classifier, and (ii) the analysis of the obtained results. The steps are presented sequentially for simplicity. Feedback loops are possible between the macro-activities (see the ⇔ arrows), either when one macro-activity finishes or at any time when an issue is identified. For example, if the performance metrics (S7) show high variance, the researcher may want to backtrack to treatment design

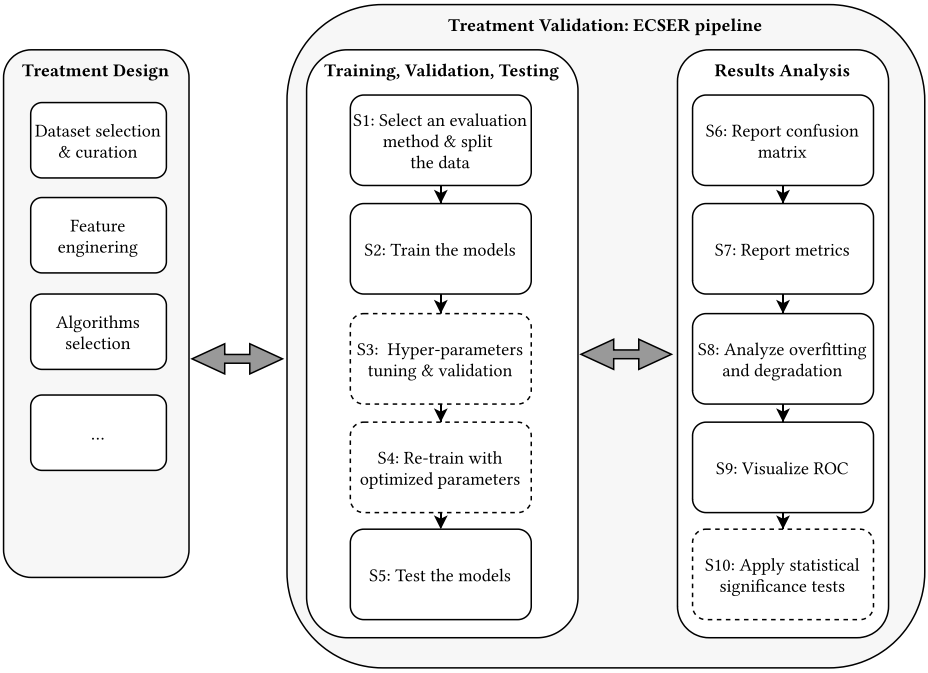


Fig. 1 The *ECSER* pipeline for evaluating classifiers in SE research, which can be seen as the *treatment validation* phase in Wieringa's design science research methodology (Wieringa 2014). Steps with a dashed border are optional. The ⇔ arrows indicate that feedback loops are always possible across the macro-activities

and conduct additional feature engineering, or to change the classifier algorithm, and then re-execute the pipeline from S1.

Looking at *ECSER* from the lens of Wieringa's design science research methodology (Wieringa 2014), it defines the *treatment validation* phase for SE researchers who are designing classifiers as their treatment. On the left of the figure, *treatment design*—outside the scope of this paper—includes important activities such as data set selection and curation, feature engineering, and algorithms selection.

Treatment Design This macro-activity corresponds to those steps that the researchers need to conduct to build their solution and develop their data set. The selection and curation of a data set focus on identifying real-world or synthetic data to assess the performance of the classifier(s). Explicit guidelines on how to transparently and credibly conduct this step are proposed, e.g., by Hutchinson et al. (2021). Algorithm selection involves the choice of the ML or DL algorithms such as Support Vector Machines, Gradient Boosting, Random Forests, and Neural Networks. Typically, studies in SE research opt for multiple algorithm so that a comparison can be made (Ghotra et al. 2015; Agrawal and Menzies 2018; Kurtanovic and Maalej 2017; Hey et al. 2020a). When ML algorithms are chosen, feature engineering is necessary to construct those features that the learning algorithm uses to predict a given data item's class(es). This is a broad topic about which entire books were written (Dong and Liu 2018; Duboue 2020). In ML4SE research, a multitude of feature types can be derived by using project management data (Montgomery et al. 2018), code metrics (Menzies et al. 2010), change metrics (Moser et al. 2008), textual artifacts (Kurtanovic and Maalej 2017), etc.

4.1 Training, Validation, Testing

S1. Select an Evaluation Method and Split the Data First, a researcher needs to decide on an evaluation method, i.e., which (and how) input data will be used to evaluate the classifiers, and split the data accordingly. Several alternatives exist. The simplest and one of the most popular methods is the *holdout method* where the data is split into training, validation, and test sets. In this setting, the model is trained using the training set (S2), the model's hyper-parameters are fine-tuned utilizing the validation set (S3), and the model is evaluated on the test set (S4–S5). One disadvantage of this method is that the results are possibly unstable, for the model is validated only once on the validation set. A more robust alternative is *k-fold cross-validation*: a test set is extracted from the data set, and the remaining data is shuffled and split into k groups (folds). k-fold cross-validation (S2–S3) consists of repeating k times training and validation. Every time one group is held as the validation set, the remaining $k - 1$ groups are used to train the model. The model's hyper-parameters are fine-tuned with respect to the average performance on the k folds. k-fold cross-validation can be stratified to keep the positives/negatives ratio roughly even across the groups. The resulting model is then evaluated on the test set (S4–S5).[1] Clearly, *k-fold cross-validation* requires more computational effort, because k models are trained and tested.

[1] To make results reproducible, it is good practice to specify a seed number for initializing the random-number generators used in classifiers, fold generation, sampling, etc. For example, the `KFold` class from the sklearn Python library can be created by indicating a `RandomState` instance and specifying the random seed to use.

Specific to the SE research, projects can be used to partition the data set when the data consists of multiple sub-sets from different SE projects. This is referred as the *p-fold* validation method, to emphasize the by-project splits (Herbold et al. 2020; Dalpiaz et al. 2019). Different projects can be used as validation and test sets for the holdout method. For p-fold cross-validation, some projects can be used as test sets and, rather than randomly shuffling the remaining data into sub-groups, the projects can be taken as a unit. For each of the p iterations, one project is held for validation and the remaining are used for training. One of the challenges is the existence of unevenly sized projects, e.g., too small projects or extremely unbalanced projects may lead to not-so-reliable results when used as the validation set. One special case of the p-fold method is the leave-one-project-out (LOPO) method; similarly to *k-fold cross-validation*, the data of a single project is reserved for testing while the rest is used for training. The p-fold method is more recent and hence less popular. On the other hand, dividing data per project is a realistic test setting for the software engineering domain to explore the generality of the results.

When the researchers have a data set that includes data from five distinct projects, they have several options. The simplest method is the holdout method, where the researchers would randomly set aside 20% of the data for testing and use the remaining 80% for training and validation. In this one-shot setting, the results are not so reliable since how the data is split directly impacts on the results. To reduce this effect, the researchers may opt for *k-fold cross-validation*, and if they set k to 10, they would train and test 10 different models and report the average results. The researchers may also get curious about the generality of the results and pose the question "*How would our classifier model perform on a new project?*". In this case, they may leave one project out for each fold, train the classifier model with the data from four projects and test the model on the remaining one. Then, they would report the average results.

S2–S5. Train, Validate, Test The decisions taken in S1 shape the following four steps: the traditional train (S2), validation (S3), and test (S5) activities in ML. S4 is introduced to emphasize the need to re-train the model after hyper-parameter tuning. After separating the *test set* in S1, S2 trains the classification model with the part of the data set that is not used as a test set. To train a classifier means to identify values for its *parameters* that allow the classifier to predict the desired output given different inputs correctly. The parameters that need to be identified depend on the selected algorithm. For example, training a Neural Network classifier means determining the weights associated with the connections between neurons. Similarly, training a Support Vector Machine means determining the coefficients of the variables of the polynomial function that characterizes the classifier.

A difference exists depending on the validation method. For the holdout method, the training step is executed by excluding the *validation set*. This set is used in S3 to run *hyper-parameter* tuning to identify those (hyper-)parameters that predict the validation set best. Instead, the optimal hyper-parameters are identified for cross-validation by iterating across the k folds. Hyper-parameters are different from the parameters of the models that are trained in S2. While model parameters characterize how the input data should be transformed into the desired output, hyper-parameters define the structure of the model that is being trained. For example, the hyper-parameters of a Random Forest include the number of decision trees to be considered in the forest or the maximum depth to allow for each decision tree. An example of hyper-parameter of Support Vector Machines is the degree of the polynomial features that characterize the model. For Neural Networks, hyper-parameters include the number of neurons in every layer and the number of layers. Several methods can be followed to perform hyper-parameter tuning. The most basic is *grid search*, where a

model is built for every possible combination of all of the hyper-parameter values that one intends to evaluate (e.g., for a Random Forest, one can consider the set {10, 20, 50, 100} for the number of decision trees, and the set {5, 10, 15, 20} for the maximum depth of each decision tree, and try all possible combinations of these values), and the architecture which produces the best results on the validation set is selected. Another common alternative is *random search*, where for each hyper-parameter, it is provided a statistical distribution from which values are randomly sampled instead of giving a discrete set of values to explore for each hyper-parameter.

In this step, the researchers should be careful not to overfit their model. For details on hyper-parameter optimization, see Agrawal et al. (2021).

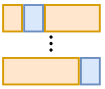
In S4, the model is re-trained using the best hyper-parameters identified in S3. A typical procedure to execute this is nested cross-validation, in which the model is trained while hyper-parameters are optimized. In terms of *ECSER*, this corresponds to executing S2 and S3 at the same time. The advantage of nested cross-validation is that it might reduce the model's bias toward the data set resulting from standard cross-validation. Still, the researchers should pay special attention to avoiding overfitting when using nested cross-validation (Cawley and Talbot 2010).

Finally, in S5, the optimized classification model is executed on the test set. S3 and S4 are optional in Fig. 1: although a good practice that may boost classifiers' performance and that can lead to simpler (e.g., to a Random Forest with a lower number of decision trees) tuned classifiers that perform better than more complex untuned ones (Fu et al. 2016; Tantithamthavorn et al. 2019), hyper-parameter tuning is not always effective and it may require extensive computational time (Tran et al. 2020).

A recommended practice, which helps to assess the generality of a classification model, is the inclusion of multiple data sets into the test set. When performing S4, this allows not only to measure the performance on unseen data but also to run statistical tests across these multiple data sets. We discuss this topic in S10.

Table 3 summarizes the Training, Validation, Testing phase of *ECSER* (i.e., steps S1-S5) by illustrating how the classification model evolves and how the data set is split in the holdout and cross-validation settings.

Table 3 Classification model and data set splitting in steps S1–S5 of *ECSER*, showing holdout and cross-validation

Step	Classification model	Holdout	X-Val
S1	None: the test set is extracted for use in S4		
S2	Fit non-test set with default hyper-parameters		
S3	Search hyper-parameters that predict the validation set best		
S4	Fit non-test set with optimal hyper-parameters from S3		
S5	Model from S4		

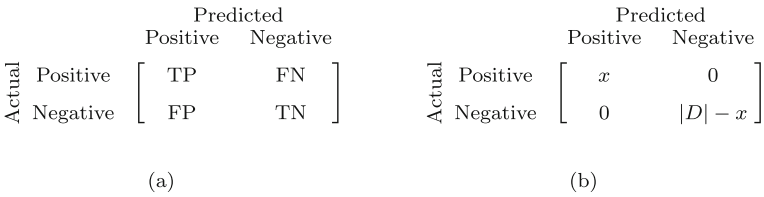


Fig. 2 Confusion Matrix (a) and an example of confusion matrix with no misclassifications of data points of a generic data set D of size $|D|$ (b)

4.2 Results Analysis

S6. Report the Confusion Matrix It is common to select a few metrics and to report only them. We recommend, instead, to present the confusion matrix so to maximize usefulness and information content of the reported data (Hall et al. 2012). A confusion matrix reports the number of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) results of a classifier. These four values comprehensively summarize the results for all classes. The readers can use them to calculate any other metrics of interest, in addition to those that the research authors find relevant for the domain. Figure 2 illustrates a generic confusion matrix alongside an example for a case where a classifier results in no misclassification, i.e., it accurately classifies every data point in a data set D.

S7. Report Metrics Depending on the domain, the researchers will report the relevant performance metrics; some examples of such metrics are shown in Table 4 (note how they all follow from the confusion matrix), and examples of their values given different confusion matrices are reported in Table 5. *Precision* and *recall* are complementary. The former evaluates the correctness of the predicted samples and the latter (also known as *sensitivity* or *true positive rate*) assesses the coverage of such predictions over the total number of positives. The F_1 score is the harmonic mean of precision and recall.

Depending on the relative human cost of correcting false positives and false negatives, the weights of precision and recall may change (Berry 2021), leading to adjusted versions of the F-Score. For example, in their ML-based approach to tracing regulatory codes to product requirements, Cleland-Huang et al. employ F_2, where higher weight is given to recall (Cleland-Huang et al. 2010). *Specificity (true negative rate)* is the ratio of the correctly identified as irrelevant samples to the overall irrelevant samples. *Accuracy* is the ratio of correct prediction to the overall cases. This metric is most suitable when the classes are balanced (Lones 2021). For instance, note that in the first unbalanced example in Table 5,

Table 4 Examples of metrics for classifier performance

Metric	Formula
Precision	$TP/(TP+FP)$
Recall (TPR)	$TP/(TP+FN)$
Specificity (TNR)	$TN/(TN+FP)$
Accuracy	$(TP+TN)/(TP+TN+FP+FN)$
F_1-score	$2\cdot(Precision\cdot Recall)/(Precision+Recall)$
F_β-score	$(1+\beta^2)(Precision\cdot Recall)/((\beta^2\cdot Precision)+Recall)$

Table 5 Examples of values of different metrics, given different confusion matrices

Data set	TP	FP	TN	FN	Precision	Recall	Specificity	Accuracy	F_1-score	F_2-score
Size: 12,	5	2	4	1	0.71	0.83	0.67	0.75	0.77	0.81
Balanced	4	4	2	2	0.50	0.67	0.33	0.50	0.57	0.63
	2	0	6	4	1.00	0.33	1.00	0.67	0.50	0.38
	6	0	6	0	1.00	1.00	1.00	1.00	1.00	1.00
	4	2	4	2	0.67	0.67	0.67	0.67	0.67	0.67
	3	3	3	3	0.50	0.50	0.50	0.50	0.50	0.50
Size: 100	99	1	0	0	0.99	1.00	0.00	0.99	0.99	1.00
Unbalanced	98	1	0	1	0.99	0.99	0.00	0.98	0.99	0.99
	0	0	99	1	–	0.00	1.00	0.99	–	–
	0	1	98	1	0.00	0.00	0.99	0.98	–	–

accuracy is almost perfect (0.99), despite the only negative data point has been misclassified, i.e., 100% of negative data points have been misclassified.

The researchers should choose the metrics based on their research goal and the problem (Yao and Shepperd 2020). Consider a system that deletes chunks of code if a classifier labels them as useless. For such a system, precision is crucial, as false positives would have catastrophic results. The recall metric might be more important for another system that marks chunks of code as smelly. Other cases may call for combining recall and precision with different weights, so F_β metric would be suitable. Berry (2021) discusses this issue specifically for requirements engineering problems.

In addition to reporting the performance of the optimized classifier model, the metrics are also helpful when comparing multiple classifier models. Although it is common to report the metrics for the performance of the classification model on the test set, the researchers should also share the metrics for the train and validation set to demonstrate the evolution of the classification model performance and to increase the replicability of their results. Primarily when the $k-$fold cross-validation method is adopted, presenting the mean and standard deviation of the metric values for the folds, as well as the cumulative confusion matrix across the folds (i.e., a confusion matrix where the values of TP, FP, TN, and FN are the sum of all the corresponding values obtained in every fold), provide a better insight for the performance of the classification model.

S8. Analyze Overfitting and Degradation This is one of the steps of *ECSER* that are less common in current research. Since SE research aims at principled ways of tackling practical problems, we propose that classification studies should report on the differences in performance between training, validation, and test sets.

Overfitting. ECSER quantifies overfitting by calculating the difference between the performance on the test set and the performance on the training set, using the metrics that were employed in S7. Thus, let M be the performance metric of relevance, then $overfitting = M_{test} - M_{tr}$. If multiple data sets are used for testing, we can compute the average overfitting as per (1), where *Test* is the set of data sets used for testing, and tr is the training set.

$$average\ overfitting = \frac{1}{\mid Test \mid} \sum_{t \in Test} (M_t - M_{tr}) \tag{1}$$

Based on the practical SE task that the classifier means to support, different metrics can be used to assess overfitting. Standard overfitting metrics include accuracy, mean square error (MSE) and zero-one loss (Bishop 2006). Accuracy and zero-one loss are suggested for binary classifiers whose output consists of the label assigned to the data points (e.g., an automated classifier extracts non-functional requirements from a specification document). Metrics such as MSE and other continuous loss functions (e.g., logistic or exponential loss), instead, provide a more fine-grained evaluation of the errors of the classifier, and they are more relevant for classifiers whose output is a score or a probability that a data point belongs to a particular class (e.g., a classifier that annotates non-functional requirements with the likelihood that they refer to one of the qualities from the ISO/IEC 25010 standard). Finally, when standard overfitting metrics are less relevant for the practical SE task (e.g., if it is essential that the classifiers have high recall), we recommend using the metrics in S7 also for overfitting.

As an example, consider the first six confusion matrices from (the first six rows of) Table 5 relating to six data sets of size 12. Suppose that the first matrix is obtained by a classifier c on the training set, the second matrix is obtained by c on the validation set, and the remaining four matrices are obtained on the test sets. Suppose to be interested in the accuracy metric. The average overfitting w.r.t. the accuracy is therefore $\frac{1}{4}((0.67-0.75)+(1.00-0.75)+(0.67-0.75)+(0.50-0.75))=-0.04$, indicating that the trained classifier present limited overfitting w.r.t. accuracy, since the average overfitting is close to 0.

Degradation. This metric compares the performance on the test set and that on the validation set, using the metrics of S7. Its calculation depends on the partitioning of the two sets. If both sets consist of a single project or consist of a data set that is not explicitly split into projects, degradation is the difference in the considered performance metric M: $degradation = M_{test} - M_{valid}$. In case the test set includes multiple projects or the validation is conducted via k-fold, we suggest calculating an average degradation, similarly to average overfitting in (1).

Continuing with the example introduced above, the average degradation w.r.t. the accuracy is $\frac{1}{4}((0.67-0.50)+(1.00-0.50)+(0.67-0.50)+(0.50-0.50))=0.21$, indicating no degradation of performance (actually an improvement) from the results obtained on the validation set to the results on the test set.

When the test set consists of multiple projects and the validation is conducted via k-fold, we recommend calculating degradation by statistically comparing the two distributions: the metric for the multiple samples of the validation set (e.g., for each of the k folds) and the metric for the various samples of the test set. If the data are normally distributed, the independent samples T-Test can be used. Else, the non-parametric alternative is suggested: Mann-Whitney's U test. These tests assess whether the degradation is statistically significant, i.e., if the p-value is below a given threshold. The researchers shall combine this result with the effect size, a statistical measure describing a phenomenon's strength. In line with Sullivan and Feinn (2012), we recommend reporting on the effect size also when the p-value is above the threshold in order to better interpret the results. This case may indicate that the population is too small to derive statistically significant results, and the researcher may want to increase the number of samples/projects. As an example of effect size, Cohen's d (Cohen 2008) computes the difference between two groups of measurements in terms of their common standard deviation, and a phenomenon has *no effect* if $|d| < 0.2$; *small effect*, if $0.2 \leq |d| < 0.5$; *intermediate effect*, if $0.5 \leq |d| < 0.8$; and *large*, if $|d| \geq 0.8$.

An example of analysis of the degradation via the Mann-Whitney's U test and the effect size is provided in Table 14 in Section 6.2 when discussing the degradation of three state-of-the-art flaky tests automated classifiers.

S9. Visualize ROC. Plotting the receiver operating characteristics (ROC) (Fawcett 2006) helps the reader to visually comprehend the performance of the model. A ROC plot (an example is reported in Fig. 3) has the true positive rate (TP/P) on the y axis and the false positive rate (FP/N) on the x axis.

Each point on a ROC plot summarizes graphically a confusion matrix. Indeed, a ROC plot is a coverage plot (i.e., a plot with the number of negatives in a data set on the x axis, and the number of positives on the y axis), with normalized axes. The normalized axes allow to deal with different class distributions, so that the plot always results squared, and classifiers can be compared with respect to different data sets on the same plot.

In Fig. 3, we plotted the results obtained with three classifiers on a given data set. We see that both Classifier 1 and 2 dominate Classifier 3, since Classifier 3 has lower TPR than both of them but does not have lower FPR than any of them. We also see that neither Classifier 1 nor 2 dominates the other: no clear winner can be established between them, and the selection of the classifier will depend on the relative importance of TPs and FPs in the specific problem that is considered. Since they are on the same diagonal, furthermore, Classifiers 1 and 2 have the same average recall.

A ROC plot is especially useful for evaluating the performance of multiple classifiers, as described in the following.

Testing generality across data sets. Every point in the ROC plot represents the performance of one classifier on a data set. This can be useful when the test set includes multiple

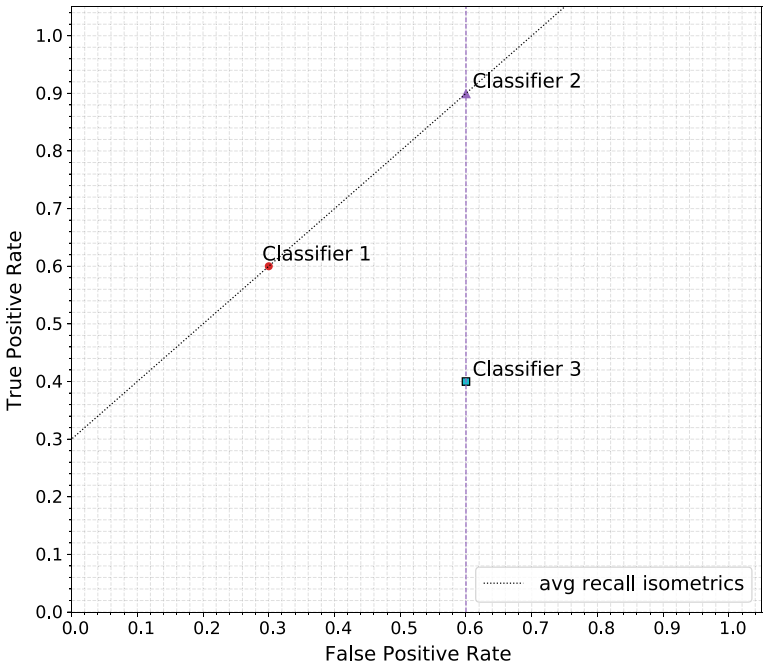


Fig. 3 An example of ROC plot. The three data points report the results of three classifiers on a given data set

SE projects. The researcher can identify, via visual analysis, the relative performance of the models on these data sets. Figure 4 reports, as an example, a ROC plot that shows the performance of four classifiers on 20 different data sets (note that for each classifier, the plot contains 20 data points). By visually inspecting the plot, we can see that the performance of Classifier 1 generalizes pretty well across all data sets: all data points are clustered together. Moreover, since all data points are close to the so-called ROC heaven (the top left corner of the plot), Classifier 1 performs almost perfectly on all data sets.

Conversely, the performance of Classifier 2 is generally poor, as for all data sets the TPR is low and the FPR is high. Despite the poor performance, however, all data points are generally clustered together, indicating that the (poor) results generalize across the data sets. Classifier 3 illustrates a different case: since the data points are not clustered together, the performance results do not generalize well across data sets and, while on some data sets the TPR is high, in other data sets it is low. Visually, however, the FPR appears to be relatively consistent across data sets, indicating that the lack of generality is mainly due to the TPR.

Exploring the sensitivity-specificity trade-off. In a ML classifier, a threshold t can be used to discriminate positives and negatives: the higher the threshold, the higher the probability that the classifier requires in order to associate an item with a class. A ROC plot can be used to visualize a so-called ROC curve, which shows the performance of the classifier with different thresholds. The area under the ROC curve (AUC) provides a summary measure that averages the accuracy across the spectrum of thresholds. It is worth noting that plotting the ROC curve could sometimes be misleading in case of data imbalance (Boyd and Eng 2013; Goadrich et al. 2006). In such cases, an alternative preferred visualization is the precision–recall (PR) curve, and the associated AUPRC (area under the precision-recall

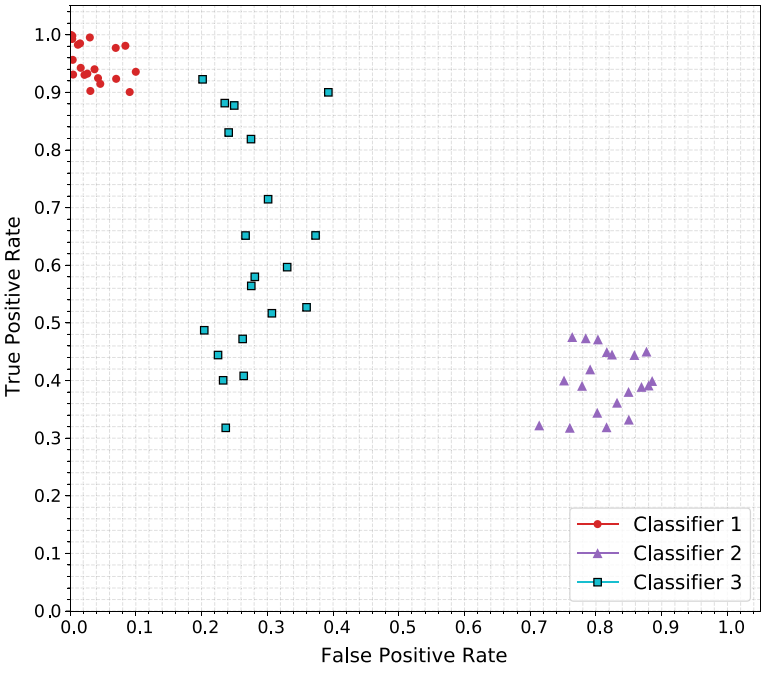


Fig. 4 An example of ROC plot reporting the performance of three classifiers on 20 different data sets composing the test set

curve) measure. The best classifier in a PR curve is as close to the top right as possible (for the ROC curve, the best classifier is as close to the top left as possible), where there is the best trade-off of precision and recall (Lever 2016).

S10. Apply Statistical Significance Tests Seeing a difference in the values of the metrics or on a plot does not entail a *statistically significant difference* in the performance of different classifiers. Proper statistical testing must be conducted to confirm that two classifiers indeed have meaningfully different performance.

Testing on a single data set. The randomization test (Good 2013) can be applied when only one data set is available for testing. It is a non-parametric method, so it can be applied even if the data are not normally distributed, or if the researchers do not know the distribution of the data. It tests the null hypothesis that the two classifiers yield to the same performance. A randomization test can be conducted using any performance metric. The idea is to verify if the results obtained with a classifiers are due to random chance by randomly shuffling the data and comparing the performance obtained on the randomized data with the actual one.

Testing on multiple data sets. Table 6 presents an overview of state-of-the-art methods for testing statistical significance across multiple data sets. This was assembled based on the recommendations on comparing multiple classifiers by Demšar (2006), the recent work on Bayesian statistical analysis by Benavoli et al. (2016a, 2017b), and the documentation of the Autorank Python package (Herbold 2020).

When comparing two classifiers, the simplest option is the paired samples T-Test. However, this requires each of the classifiers' results to be normally distributed (Herbold 2020), which is not a common situation when comparing classifiers (Demšar 2006); furthermore, this test is sensitive to outliers (Benavoli et al. 2016a). Non-parametric tests that make no assumptions on distribution and variance are an alternative. In particular, the most common options are Wilcoxon's Signed-Rank test and the Sign test. They both rely on ranks, rather than on the absolute difference in performance (as parametric tests do). Wilcoxon's Signed-Rank is preferable because of the stronger statistical power (Demšar 2006).

Table 6 A selection of state-of-the-art methods for testing statistical significance across multiple data sets

Test	Normal?	Same var?	Highlights	Suggested?
2+ Classifiers: Pairwise Comparisons				
Paired T	•		Sensitive to outliers (Demšar 2006), based on the absolute difference in performance	
Wilcoxon Signed-Rank			Based on ranks difference	•
Sign			Counts of wins, losses, ties. Weaker than Wilcoxon (Demšar 2006)	
Bayesian versions of Wilcoxon or Sign			Less affected by Type I Error. Requires definition of practical equivalence (Benavoli et al. 2017b)	
3+ Classifiers: Omnibus + Post-hoc test				
Repeated measures ANOVA	•	•	Post-hoc: Tukey's HSD	
Friedman			Post-hoc: Nemenyi	•

An alternative approach is Bayesian analysis, a paradigm shift in statistics. Benavoli et al. (2017b) explain the limitations of tests based on null-hypothesis testing and suggest Bayesian variants of the Wilcoxon and Signed tests. While powerful and less affected by Type I Error, these tests require the researcher to define a region for two classifiers to be considered as equivalent in practical settings. As the previous tests, Bayesian variants are run in a pairwise manner. The number of comparisons to make, therefore, grows exponentially with the number of classifiers.

When comparing a group of three or more classifiers, the most common strategy is to run an *omnibus* test that determines whether the group of classifiers differ in a statistically significant manner, and then a *post-hoc* test that reveals which pairs of classifiers are significantly different. We recommend two cases, in line with Autorank's documentation (Herbold 2020): (i) if the distributions are multivariate normal (Mardia 1970; Korkmaz et al. 2014), and they have approximately the same variance (*sphericity* assumption), the repeated measures ANOVA test can be executed as an omnibus, followed by the post-hoc test Tukey's HSD; (ii) the standard non-parametric alternative is Friedman's omnibus test, followed by Nemenyi's post-hoc (Demšar 2006).

Based on this discussion, SE researchers can safely use non-parametric tests as they do not make assumptions of normality (unbalanced data sets are likely to break this assumption). Wilcoxon's Signed-Rank and Friedman plus Nemenyi's post-hoc are the go-to options. The more ML-savvy researchers are invited to consider all the options in Table 6 based on the necessary analyses that need to be conducted prior to employing the tests. In Sections 5 and 6, we provide several examples of application of the statistical tests described above in two case studies comparing multiple classifiers on multiple data sets.

In some cases, the available data might be insufficient for providing meaningful statistical results. This is why step S10 is indicated as optional in *ECSER*.

4.3 Multi-class and Multi-label Classification

ECSER also applies to multi-class (2+ classes) and multi-label (1+ labels per sample) classification tasks. In multi-class problems (e.g., the problem of determining whether an app review is positive, neutral, or negative), the classes that can be attributed to data points are mutually exclusive. In multi-label problems, instead, each data point can be attributed multiple labels (e.g., a non-functional requirement can be related to both performance and security quality aspects). These problems can be reduced to several binary (2 classes, 1 label per sample) classification tasks by applying a *one-vs-rest* strategy, consisting in fitting a different binary classifier per each class (or label) against all other classes (labels). This strategy, applied in Section 5, is computationally efficient (it requires to train n binary classifiers, n being the number of classes or labels) and it is the most commonly used and advisable since it provides interpretable results about the specific classes/labels. With *one-vs-rest*, all steps of *ECSER* are only affected in that they need to be repeated for each class/label. *ECSER* also applies, with the exception of S9, to those less common multi-label classification problems where it is relevant to evaluate the classifiers w.r.t. all labels *at the same time*. In particular, S1-S5 are analogous, but need to be performed with a model that produces all labels for a given sample (e.g., classifier chains (Read et al. 2011)). In S6, a multi-label confusion matrix can be reported. For S7–S8, the literature offers several metrics such as the Jaccard index, the Hamming loss, or the multi-label generalizations of precision, recall, F_1, and accuracy (Sorower 2010). S9 cannot directly be applied for this particular type of task, unless *one-vs-rest* is applied. Finally, S10 is identical: the tests described in Table 6 can be applied w.r.t. the appropriate metrics selected for S7.

5 Case #1: Functional and Quality Requirements

As a first case study to answer RQ3 and RQ4, we take the well-known problem of classifying functional and non-functional requirements (Cleland-Huang et al. 2007), which is motivated by the importance of identifying quality aspects in a requirements specification starting from the early stages of SE. In line with recent literature (Kurtanovic and Maalej 2017; Dalpiaz et al. 2019; Hey et al. 2020a), we consider two independent classification tasks: that of identifying if a requirement contains functional (*isF*) and non-functional (*isQ*) aspects, respectively.

We apply *ECSER* to three of the most recent classifiers of requirements available in the field: *ling17* (Dalpiaz et al. 2019), *km500* (Kurtanovic and Maalej 2017) and *norbert* (Hey et al. 2020a). We compare these classifiers for two reasons. First, they adopt different strategies and NLP approaches for requirements classification. In particular, as we detail in Section 5.1, while *ling17* leverages 17 high-level linguistic features, *km500* is based on hundreds of low-level word features such as n-grams and POS n-grams, and *norbert* relies on a deep learning model. This aspect allows us to illustrate that the application of *ECSER* is independent of the type of classifiers and features being evaluated. Second, the three classifiers were recently compared on the same tasks that we consider by Hey et al. (2020a). Such comparison, however, is limited to the *validation* of the trained classifiers, i.e., step S3 of *ECSER*. We can therefore consider the work by Hey et al. (2020a) as our baseline, and use it to illustrate the usefulness of following the entire *ECSER* pipeline.

We make two contributions to the literature:

1. We annotate six additional data sets, four of which are released publicly (see Table 7).
2. We provide additional insights by *carrying out the missing steps of ECSER*, specifically, by testing the performance of the trained classifiers on unseen real-world projects.

5.1 Training, Validation, Testing

S1: Select an Evaluation Method and Split the Data Most of the recent literature in requirements classification (Kurtanovic and Maalej 2017; Li et al. 2014; Hey et al. 2020a) focused, for evaluation purposes, on *validating* their results on the PROMISE NFR data set (Cleland-Huang et al. 2006), a collection of 625 requirements from 15 projects, created and classified by graduate students. These works either split the PROMISE NFR data set into training and validation sets or apply cross-validation to PROMISE NFR. Almost

Table 7 Overview of the data sets of requirements

Data set	Public	New	Size	F	Q	Data set	Public	New	Size	F	Q
Dronology	✓		97	94	28	OAppT		✓	140	84	53
DUAP	✓	✓	148	138	110	PROMISE NFR	✓		625	310	382
ERec mgmt	✓	✓	228	163	149	RepReq		✓	99	40	47
ESA			236	91	211	ReqView	✓		87	75	32
Helpdesk			172	143	51	Streaming	✓	✓	291	135	233
Leeds Library	✓		85	44	61	User mgmt			138	126	25
NFR-Examples	✓	✓	130	15	117	WASP	✓		62	55	19
Totals									2538	1513	1518

no work in the literature completed *ECSER* after S3, i.e., by providing *testing* results after validation. The only exception we are aware of is our previous work (Dalpiaz et al. 2019); however, this study does not run statistical tests nor does it study overfitting and degradation in depth.

To study the generality of these classifiers in operational contexts, we consider 13 data sets from real-world projects other than PROMISE NFR: see Table 7 for an overview. According to existing terminology (Zimmermann et al. 2009), we are therefore investigating cross-project prediction. At the expense of some replicability, we decided to use private projects, protected by non-disclosure agreements with industrial partners, to ensure that our test set consists of *real, operational projects*.

In line with the literature, we use PROMISE NFR as the training set in S2, and we choose the *holdhout method* to *test* the trained classifiers on the 13 data sets in S5. Since we wish to explore the effectiveness of the classifiers on unseen data (S5. S5Long), we do not perform hyper-parameter tuning and validation. We take the classifiers as proposed in the literature, using the optimal hyper-parameters identified by the authors. Hence, we do not carry out S3–S4.

S2-S5: Train the Model and Test the Model We use the full PROMISE NFR data set to train both *ling17* and *km500*. We train each model separately for the two classification tasks *isF* and *isQ*. In the case of *norbert*, we use the pre-trained models that the authors made available in the replication package (Hey et al. 2020b), which also used PROMISE NFR as training set after hyper-parameter optimization.

We do not go into the details of the models and their features, which can be found in the corresponding papers and our online supplementary material (alongside the code and the public data sets) (Dell'Anna et al. 2021). Table 8 provides an overview of the major differences between the classifiers. We observe that both *ling17* and *km500* employ SVM. *ling17* uses a fixed set of 17 high-level linguistic features (e.g., dependency types), *km500* considers the top 500 low-level word features (e.g., n-grams, or POS n-grams) that characterize the training set. The *norbert* classifier, instead, adopts a transfer learning approach and is grounded on BERT (Devlin et al. 2018), the well-known deep learning model developed by Google. This step provides us with three different trained and optimized classifiers for each classification task.

We test the classifiers by studying the predictions made by the trained models for each of the 13 requirements data sets introduced in Table 7.

Table 8 Overview of the major differences between the considered classifiers of requirements

Classifier	Year	ML algorithm	Distinctive characteristics
km500 (Kurtanovic and Maalej 2017)	2017	SVM	500 lexical and syntactical features (Word-level)
ling17 (Dalpiaz et al. 2019)	2019	SVM	17 linguistic features (Sentence-level)
norbert (Hey et al. 2020a)	2020	Transfer learning	Word embedding (max seq. length 128), 10 epochs

Table 9 Confusion matrices for training and testing (S6)

Data set	Classifier	*isF*				*isQ*			
		TP	FP	TN	FN	TP	FP	TN	FN
Training (PROMISE NFR)	*ling17*	229	83	232	81	315	60	183	67
	km500	306	6	309	4	382	5	238	0
	norbert	301	10	305	9	382	27	216	0
Test (cumulative)	*ling17*	1009	321	365	194	673	258	495	463
	km500	655	185	501	548	806	377	376	330
	norbert	940	159	527	263	998	362	391	138

5.2 Results Analysis

S6-S7: Report the confusion matrix and Report metrics. Table 9 reports the confusion matrices obtained with the three classifiers on the training and testing data sets for the tasks *isF* and *isQ*. In addition to giving a comprehensive overview of the performance of the classifiers, the confusion matrix can be used to derive any metrics that the reader deems relevant, as a starting point to analyze the classifier's performance on specific data sets, and also (as shown later) to study the model's overfitting to the training data.

Table 10 reports the performance results of the three classifiers w.r.t. the three metrics that were considered in Hey et al. (2020a): precision, recall and F_1-Score. For the test sets, we report the mean value obtained on the 13 data sets and the standard deviation. The results reported for the training refer to the performance of the classifiers on the PROMISE NFR data set. By testing the classifiers on these data sets, we are the first to report on the performance of all the three state-of-the-art classifiers on previously unseen data.

By comparing the *average* performance of the classifiers on the test sets (*Test* in Table 10), we see that *norbert* consistently outperforms both *ling17* and *km500* for both tasks and all three metrics, with the exception of recall and *isF*, where *ling17* achieves the best results. These results confirm the findings reported in Hey et al. (2020a).

The performance of *ling17* and *km500* on the test sets, instead, does not seem to indicate a clear winner: the results are comparable for precision, *ling17* outperforms *km500* for recall in *isF*, while *km500* is better than *ling17* for recall in *isQ*. These results deviate from those reported in Hey et al. (2020a) when comparing *ling17* and *km500* on a 75-25 split of PROMISE NFR: there, the precision and recall of *km500* for both tasks were about 10% higher than those of *ling17*, and they were oscillating between 80% and 90%. Here, the average precision and recall of *km500* oscillate between 55% and 76% (about 20% lower than those reported in Hey et al. (2020a)), and the precision of *km500* and *ling17* is almost identical for both tasks, while the recall of *ling17* is higher for *isF*, and lower for *isQ*. This divergence of results can be attributed to the overfitting of *km500*, an issue which was also observed in Dalpiaz et al. (2019). Indeed, the performance of *km500* on the *training* data shows close-to-perfection results, indicating that the top-500 features selected by *km500* can almost perfectly characterize the whole set of requirements in PROMISE NFR. In S8, we will discuss this aspect more in detail.

The colored cells in Table 10 highlight the best-performing models. We note that it is difficult to identify an overall winner. While in some cases *norbert* is considerably better than the others (e.g., recall on the test sets for *isQ*), in other cases the difference appears

Table 10 S7–S8, performance and overfitting for the requirements classification case

Task	Classifier	Training	Test	Overfitting (Test - Training)
isF	*ling17*	0.73	0.74 ± 0.23	0.01 ± 0.23
	km500	0.98	0.76 ± 0.22	−0.22 ± 0.22
	norbert	0.97	0.83 ± 0.21	−0.14 ± 0.21
isQ	*ling17*	0.84	0.63 ± 0.19	−0.21 ± 0.19
	km500	0.99	0.61 ± 0.23	−0.38 ± 0.23
	norbert	0.93	0.67 ± 0.18	−0.27 ± 0.18
Recall				
isF	*ling17*	0.74	0.82 ± 0.11	0.08 ± 0.11
	km500	0.99	0.55 ± 0.13	−0.44 ± 0.13
	norbert	0.97	0.79 ± 0.16	−0.19 ± 0.16
isQ	*ling17*	0.82	0.57 ± 0.13	−0.26 ± 0.13
	km500	1.00	0.67 ± 0.15	−0.33 ± 0.15
	norbert	1.00	0.83 ± 0.17	−0.17 ± 0.17
F_1				
isF	*ling17*	0.74	0.75 ± 0.11	0.01 ± 0.11
	km500	0.98	0.61 ± 0.09	−0.38 ± 0.09
	norbert	0.97	0.79 ± 0.09	−0.18 ± 0.09
isQ	*ling17*	0.80	0.62 ± 0.09	−0.18 ± 0.09
	km500	0.99	0.60 ± 0.12	−0.39 ± 0.12
	norbert	0.96	0.71 ± 0.13	−0.25 ± 0.13

Green cells indicate the best results

to be negligible (e.g., precision on the test sets for *isF*). These observations motivate the execution of the following steps of *ECSER* in order to further analyze these mixed results.

S8: Analyze Overfitting and Degradation Table 10 reports also overfitting, obtained by comparing the results on the test and training sets with respect to the metrics from S7. Since we use pre-optimized classifiers, we do not analyze the performance degradation from validation to testing. Some overfitting is expected, as the testing is done on previously unseen data, but the degree of overfitting may reveal insights.

In Table 10, the average reduction of precision, recall and F_1 of the *ling17* classifier for *isF* is very close to 0. The trained model does not overfit the training data; in fact, the model tends to *under*fit the data. This is in line with the findings reported in Dalpiaz et al. (2019) and can be explained by the low number (n=17) of features. An overfitting value close to 0, however, may positively contribute to the generalizability of the classifier's performance to unseen requirements sets.

Differently, almost opposite results can be identified for *km500*, which relies on a large number (n = 500) of low-level features such as text n-grams and POS n-grams. These features, which were the most informative when fitting the training set (Kurtanovic and Maalej 2017), lead to substantial degradation when considering the test sets, reaching over 40% degradation in recall for *isF*. Similar results emerge for *isQ*; in that case, however, also *ling17* degrades of circa 20%.

Finally, although the results obtained with *norbert* on PROMISE NFR cannot be generalized to unseen data (differently from the result of *ling17* for *isF*), its performance on unseen data is generally better than the performance of *ling17*.

S9. Visualize ROC Figure 5 reports the ROC plots for the two tasks *isF* and *isQ*. Each point summarizes, graphically, the confusion matrix of a classifier on a certain data set reported in Table 9. A quick visual analysis confirms the earlier results: there is no clear-cut winner when considering the 13 test data sets. The results are mixed. Let us look at *isF*: *norbert* dominates (i.e, higher TPR, the vertical axis, and lower FPR, the horizontal axis) the others on the Helpdesk data set, while *ling17* dominates *norbert* on the Dronology data set. In other cases, the winner depends on the relative importance of TP and TN: for example, see the WASP or DUAP data sets.

The ROC plot also visually highlights the degree of overfitting to the training data. For *isQ*, for example, the performance on the PROMISE NFR training set for *km500* is almost perfect (close to the ROC heaven, where TPR is 1 and FPR is 0), while *ling17* has a lower performance on the training set. However, once tested on different data, we see that the performance of *km500* degrades to values that are considerably lower than those obtained on the training, and similar, and often even lower, than those of *ling17* (e.g., compare the results on ReqView, User mgmt, ERec mgmt). This suggests that the *km500* classifier faces some overfitting issues. Conversely, the results of *ling17* on the test sets are more clustered in the surroundings of the training data point (PROMISE) on the ROC plot, indicating less overfitting.

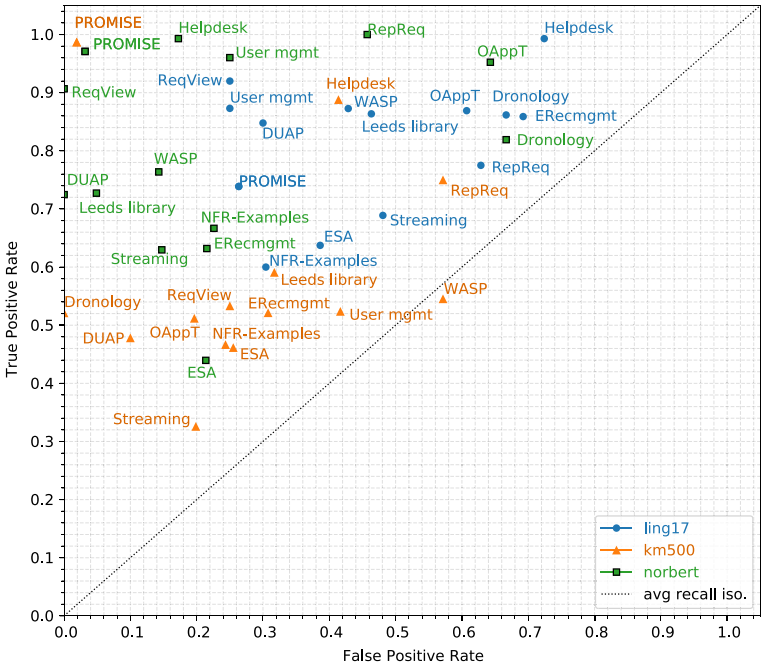
S10: Apply Statistical Significance Tests We check whether the observations made so far have statistical significance. Table 11 reports the results of the statistical tests when comparing the various classifiers. In line with Table 6, we use repeated measures ANOVA and Tukey's HSD when the assumptions of multivariate normality[2] and similar variance are met; otherwise, we employ Friedman followed by Nemenyi's post-hoc test. Please check our online appendix for all the details of the execution of the tests. Here, we only provide a summary. The yellow-highlighted cells denote statistically significant results with p-value ≤ 0.05.

Tables 10 and 11 highlight that the only case in which a classifier outperforms another on all metrics, with statistical significance, is *norbert* over *km500* for the *isF* task, with a large effect size for recall and F_1, and small for precision.

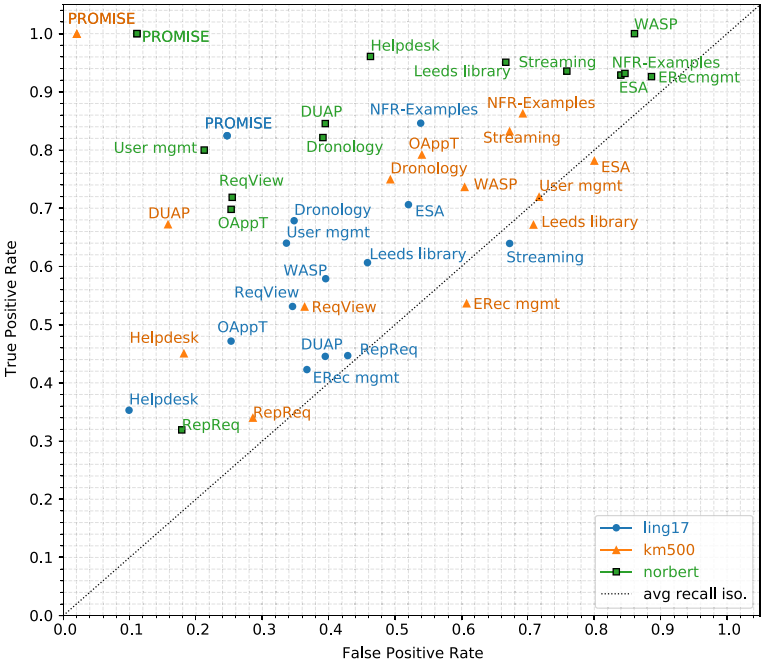
Also, by looking at the ROC plot of Fig. 5 and the statistical tests of Table 11, we can identify a finding concerning the *isQ* task, which is the one that originated this thread of research (Cleland-Huang et al. 2006). We can observe that there is no statistically significant difference between the three classifiers for precision and F_1 for the *isQ* task, although the omnibus test indicates a significant difference: this should be considered as a false positive and it shows the importance of running a post-hoc after the omnibus (Tian et al. 2018).

Furthermore, while the original paper showed that *km500* could outperform *ling17* using cross-validation (Hey et al. 2020a), this difference disappears when considering the test set, thereby highlighting the importance of extracting the test set and of analyzing the classifier models' performance against it, and providing evidence for the usefulness of applying *ECSER*.

[2]The `Autorank` package that we used approximates multivariate normality by checking the univariate normality of each distribution. For extra caution, we also executed Mardia's normality (Mardia 1970) test using R's package `MVN` (Korkmaz et al. 2014), which confirmed the results.



(a) *isF* task



(b) *isQ* task

Fig. 5 *ECSER*'s S9: ROC plot for *ling17*, *km500* and *norbert*

Table 11 S10: statistical tests for the performance metrics. p^f and p^a are the p-values obtained with the Friedman and repeated measures ANOVA omnibus tests, respectively

		Omnibus	Post-Hoc/Cohen's d (magnitude)		
			ling17 vs *km500*	*ling17* vs *norbert*	*km500* vs *norbert*
isF	Prec	$p^f = 0.002$**	0.059 (none)	0.37 (small)	0.314 (small)
	Rec	$p^a = 0.0$**	2.152 (large)	0.236 (small)	1.528 (large)
	F_1	$p^a = 0.0$**	1.39 (large)	0.43 (small)	1.989 (large)
isQ	Prec	$p^a = 0.066$			
	Rec	$p^f = 0.0$**	0.683 (medium)	1.659 (large)	0.977 (large)
	F_1	$p^a = 0.014$*	0.134 (none)	0.778 (medium)	0.807 (large)

* indicates $p \leq 0.05$, ** indicates $p \leq 0.01$. When the omnibus test identified no difference, the post-hoc tests (Nemenyi for Friedman, Tukey's HSD for ANOVA) were ignored. In the other cases, a yellow cell indicates significant difference. Values within brackets indicate the interpretation of Cohen's d for the measured difference between classifiers. *none* indicates $|d| < 0.2$, *small* indicates $0.2 \leq |d| < 0.5$, *medium* indicates $0.5 \leq |d| < 0.8$, and *large* indicates $|d| \geq 0.8$

5.2.1 Summary of Findings from Case #1

In the following, we summarize the key findings from the application of *ECSER* to the case of classification of functional and non-functional requirements.

1. S5 of *ECSER* confirms the findings reported in Hey et al. (2020a): *norbert* outperforms both *ling17* and *km500* for both *isF* and *isQ* tasks on unseen data in terms of precision, recall and F_1 (column *Test* in Table 10). The only exception is recall and *isF*, where *ling17* achieves the best results.
2. No clear winner, however, can be identified from the analysis of the performance in Table 10 and of the ROC plots in Fig. 5: *km500* fits best the training set, *norbert* performs best on the test set, while *ling17* has the smallest overfitting.
3. The only case in which a classifier outperforms another on all metrics with statistical significance (see Table 11) is *norbert* over *km500* for the *isF* task. The effect size is large on recall and F_1-score, small on precision.
4. While the original paper (Hey et al. 2020a) shows comparable performance for the *isQ* and the *isF* tasks, our application of *ECSER* to the test set reveals that the performance on *isQ* is lower than that of *isF* (Fig. 5) and shows few statistically significant differences between the classifiers (see Table 11).

6 Case #2: Test Case Flakiness

The second case concerns the classification of test cases as *flaky*, i.e., if their pass/fail status is non-deterministic. We apply *ECSER* to the three flaky test classifiers also compared by Alshammari et al. (2021a): FlakeFlagger (*FF*), Vocabulary-Based Approach (Pinto et al. 2020) (*Voc*), and a combination of the features from the previous two models (*VocFF*). The pipeline suggested by the original work is classifier agnostic. To demonstrate applicability, the original paper uses the following shallow machine learning techniques: Decision Trees (DT), Random Forests (RF), Support Vector Machines (SVM), Multilayer Perceptron (MLP),

Naive Bayes (NB), Adaboosting (Ada), and K-Nearest Neighbors (KNN). In this paper, we compare the three classifiers using the algorithm that performed the best according to the original authors, i.e., Random Forests. Similarly to the requirements classification case, our replication completes *ECSER* going beyond the validation step (S3). We rely on the feature engineering described in the original paper (Alshammari et al. 2021a): we start from the pre-processed data (which includes populated features) that we could find in *FF*'s online appendix (file `processed_data_with_vocabulary_per_test.csv` in Alshammari et al. (2021b)). In the repository for our online materials, we provide a summary of the steps of *ECSER* that we completed for both this and the previous case study (Dell'Anna et al. 2021).

6.1 Training, Validation, Testing

S1: Select an Evaluation Method and Split the Data The original paper (Alshammari et al. 2021a) uses stratified k-fold as a validation method. Their data set (from the replication package) was randomly shuffled and split in 10 folds. Every fold was used as validation set (S3) and the classifiers were trained using the remaining entries of the data set as training set (S2). The process was repeated 10 times until results were obtained for all folds. This method seems suitable, for the available data are unbalanced and stratification combined with the k-fold method contribute to the reliability of the performance results.

We adopt the same validation method, but we first extract a test set (as recommended by *ECSER*) in order to obtain a more credible evaluation. As shown in Table 12, we systematically select 7 projects for the test set: the one with the 2nd highest number of flaky tests, the 4th, ..., the 14th, leading to the projects {`hbase`, `okhttp`, `hector`, `java-websocket`, `httpcore`, `incubator-dubbo`, `wro4j`}. This test set has 5549 test cases, 358 of which are flaky. Our choice was done without setting additional selection criteria, with the main aim of keeping a sufficiently high number of samples in the training set. The other projects are used for training and validating the model via 10-fold cross-validation.

S2: Train the Model To cope with data imbalance in the training set, we use the SMOTE oversampling technique (as per Alshammari et al. 2021a) before training the classifiers. We train the three classifiers *FF*, *Voc* and *VocFF* from Alshammari et al. (2021a). They can be used with different classification models, e.g., Decisions Tree, Random Forest, Support Vector Machine, Naive Bayes, etc. The features of *FF* are factors identified in the literature to affect the flakiness of tests, such as the execution time, the number of covered lines of code, or the number of covered classes. *Voc*, instead, is a vocabulary-based approach. It differs from *FF* in that it uses lower-level features, such as keywords, APIs or tokens contained in the test case. *VocFF*, finally, combines the two approaches.

S3–S4: Hyper-parameters Tuning and Validation and Re-train with optimized Params To validate the model, the classifiers trained in each of the 10 folds are validated on the corresponding validation sets. In Alshammari et al. (2021a), the authors compared different sampling strategies and classification algorithms. We do not repeat hyper-parameters tuning; we use their results and apply *ECSER* with Random Forest as a learning algorithm and SMOTE oversampling. Therefore, we also do not execute S4, as it existentially depends on S3.

S5: Test the Model We conduct the following steps of *ECSER* up to S10, going beyond the validation step where the original paper (Alshammari et al. 2021a) stopped. In S5, we test

Table 12 Summary of the projects for the test case flakiness case. The second and third columns quantitatively describe the 24 projects based on the counts included in the pre-processed data provided by the authors of *FF* (Alshammari et al. 2021a), the last two columns indicate our partitioning into train set and test set

Project	Tests	Flaky	Data splitting	
			Train set	Test set
spring-boot	2,108	160	✓	
hbase	431	145		✓
alluxio	187	116	✓	
okhttp	810	100		✓
ambari	324	52	✓	
hector	142	33		✓
activiti	2,043	32	✓	
java-websocket	145	23		✓
wildfly	1,023	23	✓	
httpcore	712	22		✓
logback	805	22	✓	
incubator-dubbo	2,174	19		✓
http-request	163	18	✓	
wro4j	1,135	16		✓
orbit	86	7	✓	
undertow	183	7	✓	
achilles	1,317	4	✓	
elastic-job-lite	558	3	✓	
zxing	345	2	✓	
assertj-core	6,261	1	✓	
handlebars.java	420	1	✓	
ninja	307	1	✓	
commons-exec	55	0	✓	
jimfs	212	0	✓	
Train set total	16,397	449		
Test set total	5,549	358		

the classifiers on the test data we extracted in S1. In the following, we analyze the obtained results.

6.2 Results Analysis

S6-S8: Report the Confusion Matrix, Report metrics and Analyze Overfitting and Degradation Tables 13 and 14 report, respectively, the confusion matrices and the performance results obtained with the three classifiers on both the training set in S2, the validation set in S3, and the test sets in S5. For cross-validation and testing, the values reported in the confusion matrices are the *cumulative* values obtained in the 10 folds, and on the 7 projects in the test set, respectively. Table 14, additionally, reports an explicit analysis of overfitting and degradation of the classifiers w.r.t. the metrics from S7.

Table 13 S6: confusion matrix

Data set	Classifier	TP	FP	TN	FN
Training	*FF*	449	1	15947	0
	Voc	401	2655	13293	48
	VocFF	449	1	15947	0
Validation	*FF*	348	143	15805	101
	Voc	344	2515	13433	105
	VocFF	354	121	15827	95
Tests	*FF*	14	134	5057	344
	Voc	120	1669	3522	238
	VocFF	29	114	5077	329

The results for the *validation step* are in line with those reported in the original paper (Alshammari et al. 2021a). We see that *FF* and *VocFF* perform better than *Voc* w.r.t. all metrics. In particular, as noted by Alshammari and colleagues, the two versions of FlakyFlagger provide an increase of precision of circa 60% and of recall of about 2% compared to *Voc*. *VocFF*, furthermore, appears to perform better than *FF* in terms of precision and F_1. The two trained FlakeFlagger classifiers characterize well the training data (note the almost perfect results on the training data).

Table 14 S7–S8, performance results and analysis of overfitting (as per (1)) and degradation (Mann-Whitney U test and the effect size η^2)

Training	*FF*	1.00	1.00	1.00
	Voc	0.13	0.89	0.23
	VocFF	1.00	1.00	1.00
Validation	*FF*	0.71 ± 0.05	0.78 ± 0.07	0.74 ± 0.04
	Voc	0.12 ± 0.02	0.77 ± 0.08	0.21 ± 0.03
	VocFF	0.75 ± 0.04	0.79 ± 0.06	0.77 ± 0.03
Tests	*FF*	0.09 ± 0.19	0.05 ± 0.07	0.03 ± 0.04
	Voc	0.15 ± 0.17	0.34 ± 0.18	0.16 ± 0.15
	VocFF	0.12 ± 0.23	0.05 ± 0.06	0.06 ± 0.09
Overfitting	*FF*	-0.91 ± 0.19	-0.95 ± 0.07	-0.97 ± 0.04
(Test - Training)	*Voc*	0.01 ± 0.17	-0.55 ± 0.18	-0.07 ± 0.15
	VocFF	-0.87 ± 0.23	-0.95 ± 0.06	-0.94 ± 0.09
Degradation	*FF*	$p = 0.001$**	$p = 0.001$**	$p = 0.001$**
(Test vs Validation)		$\eta^2 = 0.686$ (*large*)	$\eta^2 = 0.686$ (*large*)	$\eta^2 = 0.686$ (*large*)
	Voc	$p = 0.193$	$p = 0.001$**	$p = 0.161$
		$\eta^2 = 0.11$ (*medium*)	$\eta^2 = 0.686$ (*large*)	$\eta^2 = 0.126$ (*medium*)
	VocFF	$p = 0.001$**	$p = 0.001$**	$p = 0.001$**
		$\eta^2 = 0.686$ (*large*)	$\eta^2 = 0.686$ (*large*)	$\eta^2 = 0.686$ (*large*)

The effect size is interpreted as *none* if $|\eta^2| < 0.01$, *small* if $0.01 \leq |\eta^2| < 0.06$, *medium* if $0.06 \leq |\eta^2| < 0.14$, and *large* if $|\eta^2| \geq 0.14$. Green indicates best results, yellow statistically significant ones

The test set results, however, portray a radically different picture. Row *Tests* in Tables 13 and 14 summarize the results of the 7 test data sets via, respectively, the cumulative confusion matrix and the macro-average and standard deviation. The mean precision of *FF* and *VocFF* dropped from ∼ 70% during validation to 9% and 12%, respectively, during testing. The average recall dropped from ∼80% to ∼5%, and the average F_1 from ∼75% to only 6% and 3% for *VocFF* and *FF*, respectively. Notably, *Voc*, which did not lose in precision on the test set, had also a more moderate degradation w.r.t. recall and F_1, and the recall of *Voc* (the baseline in Alshammari et al. 2021a) is about 30% *higher* than that of *FF* and *VocFF*.

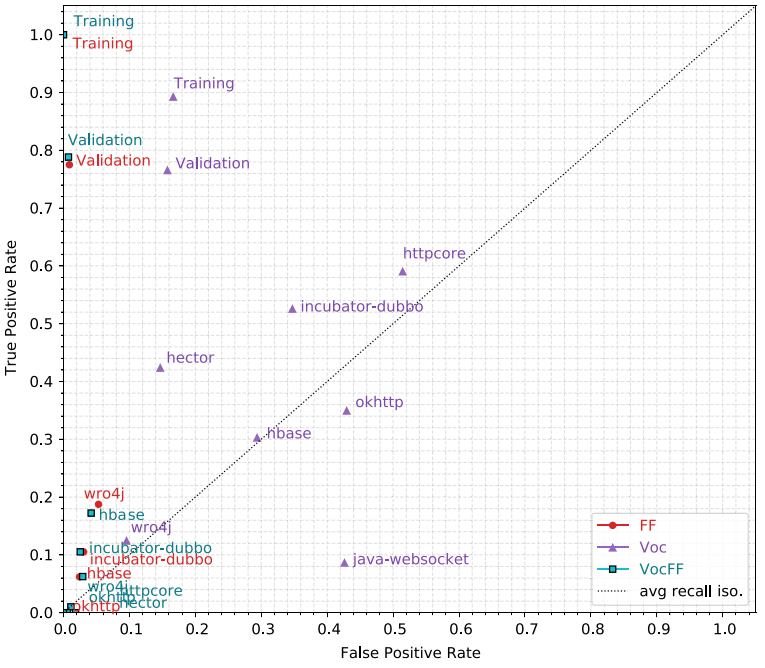
The analysis of overfitting and degradation reported in Table 14 reveals large values of overfitting (test-train); for example, the precision of *VocFF* decreased of 0.87 compared to the training set, and the recall decreased of 0.95. *Voc* is an exception: for precision, overfitting is almost zero, while for recall, the decrease is 0.55, much less than for *FF* and *VocFF*. For degradation (test-validation), unlike Section 5, we report the statistical tests since both the validation set and the test set consist of more than one sample: the validation set refers to 10 folds, the test set consists of 7 projects. These results are in line with those of overfitting. The effect size of the degradation is *large* for all metrics when considering both *FF* and *VocFF*, and they are significantly below the 0.01 threshold. The degradation of *Voc* is lesser: while the effect size is *large* for recall, with statistical significance $\leq$0.01, there is no statistically significant degradation for precision and F_1.

These results illustrate that while the features used by *FF* and *VocFF* characterize well the training set (i.e., they have a good *descriptive* power), the models trained with those features do not generalize well on unseen data (i.e., they do *not* have a good *predictive* power). For example, a decision tree that uses the feature *execution time* (used by *FF* and *VocFF*) could perfectly describe the training set by learning enough rules such as: IF 0.01 ms $\leq$ *execution time* $\leq$ 0.0199 ms THEN *not-flaky*, IF 0.02 ms $\leq$ *execution time* $\leq$ 0.0299 ms THEN *flaky*, IF 0.03 ms $\leq$ *execution time* $\leq$ 0.04 ms THEN *not-flaky*, etc. However, the learned rules will not perform well on unseen data unless the features are actual predictors of flakiness.

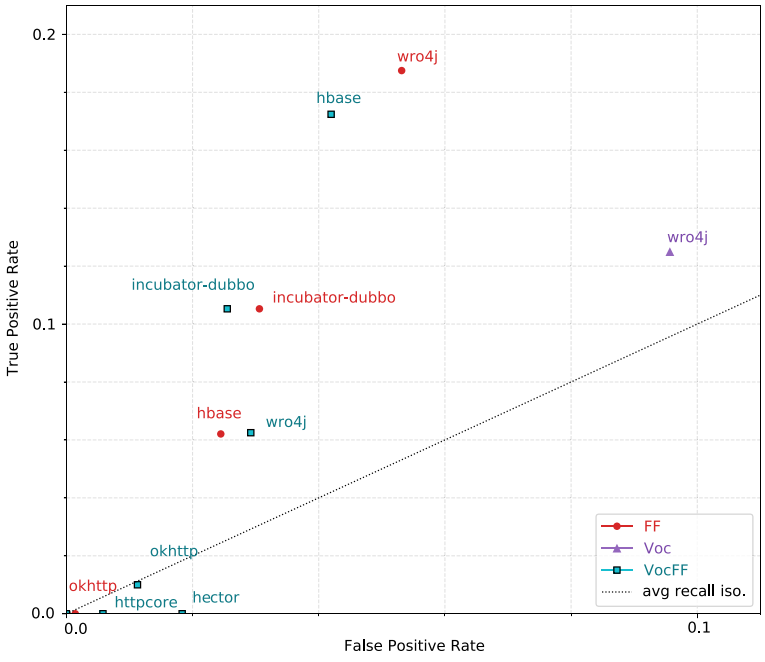
S9: Visualize ROC Figure 6 reports the ROC plot that compares the three classifiers. In addition to the results obtained on the training and on the test sets, we report also the validation results. For ease of visualization, we do not report in the figure the labels of the data sets for which the classifiers resulted in both TPR and FPR equal to 0 (i.e., for the data points lying at the origin of the axis). Details can be found in our online appendix (Dell'Anna et al. 2021). The ROC plot confirms the results about validation presented in Alshammari et al. (2021a): both *VocFF* and *FF* dominate *Voc*, with *VocFF* performing slightly better than *FF*. Thus, combining the features of *FF* and *Voc* allowed to improve the TPR without increasing too much the FPR, resulting in a slightly better classifier.

Moving to the test set, however, the plot confirms the analysis done in S6-S8: for almost all projects in the test set, the results obtained by the classifiers, and in particular *FF* and *VocFF*, are clustered around the origin of the axis. This indicates both a low FPR (i.e., they performed well on the negative class, which however was also the predominant one) and very low TPR, performing poorly on the positive class (the one of interest for the problem).

S10. Apply Statistical Significance Tests Table 15 reports the results of the Friedman omnibus test (the data are not normally distributed) and the Nemenyi post-hoc tests that compare the performance of the three test case flakiness classifiers w.r.t. the different metrics.



(a)



(b)

Fig. 6 ROC plots that compare *FF*, *Voc*, and *VocFF* (S9). Fig. 6b is a zoom-in of the bottom left area of Fig. 6a

Table 15 S10, statistical tests for the performance metrics of case #2

	Omnibus	Post-Hoc/Cohen's d (magnitude)		
		FF vs *Voc*	*FF* vs *VocFF*	*Voc* vs *VocFF*
Precision	$p^f = 0.698$			
Recall	$p^f = 0.011$**	2.037 (large)	0.009 (none)	2.062 (large)
F_1	$p^f = 0.289$			

p^f and p^a are the p-values obtained with the Friedman and repeated measures ANOVA omnibus tests, respectively. * indicates $p \leq 0.05$, ** indicates $p \leq 0.01$. When the omnibus test identified no difference, the post-hoc tests (Nemenyi for Friedman, Tukey's HSD for ANOVA) were ignored. In the other cases, a yellow cell indicates significant difference. Values within brackets indicate the interpretation of Cohen's *d* for the measured difference between classifiers. *none* indicates $|d| < 0.2$, and *large* indicates $|d| \geq 0.8$

The statistical analysis confirms the considerations made in the previous steps and, together with Table 13, allows us to derive a finding that deviates from the original paper's conclusions (in which *VocFF* was shown to be the best), providing further evidence of the usefulness of applying *ECSER*: there is no significant difference in the precision and F_1 of the three classifiers when predicting the flakiness of a test. Instead, the recall of *Voc* is significantly higher than those of *FF* and *VocFF*.

6.2.1 Summary of Findings from Case #2

In the following, we briefly summarize the key findings that resulted from the application of *ECSER* to the case of classification of test cases as flaky.

1. Validation (S3) is not always a predictor of operational performance. While *FF* and *VocFF* outperform *Voc* on the validation set (see Table 13), the predictive power of their features is low, as overfitting and degradation show a significant decrease (see Table 14).
2. On the test set, there is no significant difference in the precision and F_1-score of the three classifiers when predicting the flakiness of a test. Instead, the recall of *Voc* is significantly higher than those of *FF* and *VocFF*.

7 Threats to Validity

We discuss the major threats to validity following the taxonomy proposed by Wohlin et al. (2012).

Conclusion Validity is concerned with the relationship between the treatment (*ECSER*) and the outcome: its applicability (RQ3) and usefulness (RQ4). Some of the findings we obtained via the replications depend on the reliability of the statistical tests. Following the guidelines of Table 6, we chose robust tests and we derived conclusions only from results that had strong significance level and large effect size. Another threat concerns the reliability of the measures; in our case, the labelled data. We minimized this threat by using previously labelled data, and by involving multiple taggers (including a non-author) for the additional data sets in the first case. It is obviously possible that, should the data be tagged by other humans, the results could differ.

Internal Validity affects the independent variable with respect to causality. The selection of the data to be included in the training, validation, and test sets may affect the causality of

the findings. To mitigate these risks, we followed different strategies. In the requirements classification case, we selected PROMISE NFR as the training set, following state-of-the-art papers in the field, and we obtained data sets from external resources for the test set, like in Dalpiaz et al. (2019). We annotated the newly introduced data sets (Table 7) with the help of an external annotator. Each data sample has been tagged by two annotators, and conflicts have been resolved in discussion meetings. When there was difficulty for conflict resolution, all annotators discussed the annotation and resolved the conflict. For the flaky test case classification case, we allocated a subset of the original projects to the test set. We did this in a systematic way by choosing the 2nd, 4th, ... project with most positives. Although the selection carries some risks, we believe the effect is negligible, since the performance on the validation sets is similar to that reported in the original paper (Alshammari et al. 2021a).

Construct Validity concerns the degree to which an experiment can be used to draw general conclusions about the concept or theory behind it. *ECSER* itself aims to be an approach that can be used to derive more credible conclusions than the current state-of-the-art in SE research. To such extent, we include steps to report more detailed results (e.g., the confusion matrix in S6), to analyze overfitting and degradation (S8), and to measure the statistical significance (S10) of the results. The generality of the conclusions, however, also depends on the input data. *ECSER* minimizes the effect of the used data to train, validate, and test a classifier. However, it is possible that the results with different data sets may differ. Minor threats in this category may derive from our implemented code. Slight differences in the results for the flaky tests case study may stem from a different seed selection, which were not reported in Alshammari et al. (2021a). The code of the case studies is taken from the original studies where applicable, but additional code is written for the missing steps of the pipeline. All the code is available in Dell'Anna et al. (2021).

External Validity regards the ability to generalize the results beyond the studied context. First, *ECSER*'s applicability is assessed through two cases of classification in software testing and requirements engineering. We are aware of the limited generality that can be derived from the use of only two cases, which, however, indicate clear directions, and we invite researchers from different SE areas to apply *ECSER* to their cases. Furthermore, we could not study whether the statistically significant differences actually translate into a noticeable difference in practical settings, and whether statistical significance is necessary for practitioners to perceive that one classifier is better than another. This is beyond the scope of the present paper, but in-vivo studies are necessary to investigate the practical effectiveness (w.r.t. work practices) of SE classifiers.

8 Conclusions and Outlook

In this paper, we introduced the *ECSER* (Evaluating Classifiers in Software Engineering Research) pipeline to guide SE researchers in accurately evaluating and reporting their research on automated classifiers. *ECSER* aims to assist SE researchers in avoiding common mistakes and in presenting their results credibly and correctly. *ECSER* adopts and consolidates recommendations from recent literature in ML and statistics and presents them for SE researchers by illustrating the concepts with examples and two case studies from the SE community.

To demonstrate the applicability and usefulness of *ECSER*, we conducted two replication studies, one in software testing and one in requirements engineering, by applying the pipeline. Our replications allow us to strengthen some of the conclusions in the original

papers' and they reveal findings that are not present in the original studies, some of which contradict the original results.

Our overarching goal is to improve the quality of ML4SE research by proposing more rigorous ways of conducting and reporting research. To support this, we made available a replication package (Dell'Anna et al. 2021) for the interested reader to apply *ECSER* in order to compare their classifiers across their data sets. The pipeline is not set in stone. We call upon SE researchers to use *ECSER* for additional classification problems and we welcome improvements to it.

In the rest of this section, we discuss the main findings from this research. In Section 8.1, we answer the research questions that we addressed. In Section 8.2, we discuss the limitations of *ECSER* and we outline future work.

8.1 Answering the Research Questions

Our main research question *MRQ. How can we enable SE researchers to accurately conduct and report on the evaluation of automated classifiers?* is decomposed into four sub-questions, which we address in the following.

RQ1. What are the Challenges with the Current Research Practices with Classifiers in SE Research? We answer *RQ1* by investigating recent research output in one of the most prominent software engineering research conferences. In Section 3, we identify several challenges related to the ML classifiers in software engineering research. These include the risk for SE researchers of using ML tools without fully grasping the theory behind them due to the scattered ML knowledge, and the lack of arguments for selecting performance metrics. Additionally, we note that in several cases (including the two case studies in Sections 5 and 6), the evaluation of classifiers is focused on validation data rather than test data, resulting in findings and conclusions that might be misleading and not generalize well, because the validation sets are often extracted from (hence, they often reflect some properties and patterns of) the training set. Finally, we identify the difficulties in verifying and reproducing the metrics or calculating other metrics. This is due to the omission of the confusion matrix and the lack of analysis of the statistical significance of the results in most recent studies.

RQ2. What is an Easy-to-Use, Tangible Artifact that can Assist SE Researchers When Conducting and Reporting Research on Classifiers? To mitigate the challenges identified with *RQ1*, we introduce *ECSER*, a pipeline for evaluating classifiers in SE research. *ECSER* is a tangible artifact, in the form of a 10-step process, that brings together the scattered ML knowledge and presents it in an organized way. When conducting and reporting research on automated classifiers, SE researchers can consult the pipeline to identify the key steps to follow for a reproducible and accurate evaluation. For each step, *ECSER* provides alternatives to choose from based on particular cases, alongside references for further details. The pipeline is also accompanied by examples from the SE field, which are typically not part of general ML guidelines.

RQ3. How Applicable is the Pipeline to ML4SE Research? We apply *ECSER* for replicating and extending two studies. The first study concerns the automated classification of functional and quality requirements (Hey et al. 2020a; Dalpiaz et al. 2019; Kurtanovic and Maalej 2017), while the second study focuses on detecting test case flakiness (Pinto et al. 2020; Alshammari et al. 2021a). Since *ECSER* is agnostic to the underlying classification

algorithms and the features used to characterize the data, we successfully apply it to compare six different classifiers (3 per case study) with different features and underlying models.

RQ4. How useful is the Pipeline When Applied to ML4SE Research? The application of *ECSER* to the two case studies allows us to demonstrate also its usefulness. In addition to solidifying the results from the original papers, through the explicit definition of an independent test set (S1), the analysis of overfitting and degradation (S8), and the execution of statistical significance tests (S10), we identify additional findings that are not mentioned in the original studies. For the requirements classification task, for example, the *norbert* classifier proved to be the best overall, although the statistical results show that the differences for the *isQ* task are mostly not significant. For the flaky test classification task, the results are even more striking: contrary to the original results, they indicate that the baseline classifier (*Voc*) achieves significantly higher recall, and comparable precision, on the test set. By comparing the two cases in terms of overfitting and degradation (Table 10 vs. Table 14), we conclude that ML-based flaky test identification is an area where more research is necessary. Our results reveal limited predictive power for the used features, especially in terms of precision (in the 9%-15% range).

8.2 Limitations and Future Work

Applying all the steps of the pipeline might not be trivial. This is particularly true for step S10: statistical significance tests. In some cases, the available data might be insufficient for providing meaningful statistical results. This is why step S10 is indicated as optional in *ECSER*. However, the lack of sufficient data for providing meaningful statistical results should not be ignored but reflected in the discussion and conclusions drawn from the results reported in the paper. Additionally, while we attempted to provide an accessible overview of the state-of-the-art methods for testing statistical significance (e.g., see Table 6) with both recommendations and references to available implementations (e.g., the Autorank Python package (Herbold 2020), and our online appendix (Dell'Anna et al. 2021)), the selection of an appropriate statistical test and the interpretation of the results needs a basic understanding of statistics and of the different tests. Future work should study the applicability of this step through human evaluations with SE practitioners having different degrees of familiarity with statistics.

Step S3 (hyper-parameter tuning and validation) might require more computational power and time from the researchers who apply the pipeline. The degree to which the classifiers should be optimized depends on the maturity of the evaluated classifiers. This is why also steps S3-S4 of *ECSER* are indicated as optional. Hyper-parameter tuning, however, has been shown to lead to simpler classifiers that perform better than more complex non-tuned ones (Fu et al. 2016; Tantithamthavorn et al. 2019), so additional effort put in executing these steps might lead to better and more interpretable outcomes. In general, applying *ECSER* requires more effort from the researchers than just reporting, for example, the performance obtained on the test set. However, we believe that this effort is necessary for reporting solid, accurate, and non-misleading results and supporting replicability and reproducibility in ML4SE (Liu et al. 2021). In this sense, this paper attempts to minimize practitioners' efforts. In the future, we intend to conduct a controlled experiment with SE students about evaluating automated classifiers with and without the help of *ECSER*, to provide additional evidence about the pipeline's usefulness (also in the context of education).

ECSER aims to concisely present the steps required to evaluate classifiers. This paper, however, does not replace ML and statistics textbooks. In terms of completeness, for example, *ECSER* covers the evaluation of multiple classifiers but does not discuss essential aspects for constructing classifiers, such as data set selection and curation, feature engineering, and algorithm selection. These are currently considered as an input of the pipeline (leftmost block in Fig. 1). We intend to study if such tasks present a challenge for the SE practitioners by investigating whether appropriate classifiers and training sets are used for ML4SE research. This may lead to additional guidelines in that direction. Moreover, given the depth of each technique and concept discussed, the description of the several steps of *ECSER* might lack details for curious readers. We provide several references for every step of *ECSER* to satisfy their curiosity. Our two case studies also illustrate how the pipeline is applied. As an empirical evaluation of the pipeline, we intend to assess the most critical and less obvious steps of the pipeline. In the long run, we aim to construct a comprehensive resource for researchers. Our online appendix is the first step in this direction.

Acknowledgements We would like to thank Sercan Çevikol for his assistance in tagging the data sets. We also thank Dr. Arzucan Özgür (Boğaziçi University) for her comments on comparing information retrieval systems.

Funding The second author has been partially supported by the Scientific and Technological Research Council of Turkey through BIDEB 2232 grant no. 118C255.

Data Availability The datasets and source code generated during and/or analysed during the current study are available in the Zenodo repository, https://doi.org/10.5281/zenodo.6266675.

Declarations

Conflict of Interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) Tensorflow: large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/. Software available from tensorflow.org

Adams NM, Hand DJ (2000) Improving the practice of classifier performance assessment. Neural Comput 12(2):305–311

Agrawal A, Menzies T (2018) Is "better data" better than "better data miners"? In: IEEE/ACM international conference on software engineering, pp 1050–1061

Agrawal A, Yang X, Agrawal R, Yedida R, Shen X, Menzies T (2021) Simpler hyperparameter optimization for software analytics: why, how, when. IEEE Trans Softw Eng 48:2939–2954

Alonso-Betanzos A, Bolón-Canedo V, Heyndrickx GR, Kerkhof PL (2015) Exploring guidelines for classification of major heart failure subtypes by using machine learning. Clin Med Insights: Cardiol 9:CMC–s18746

Alshammari A, Morris C, Hilton M, Bell J (2021a) Flakeflagger: predicting flakiness without rerunning tests. In: IEEE/ACM international conference on software engineering, pp 1572–1584
Alshammari A, Morris C, Hilton M, Bell J (2021b) Flaky test dataset to accompany "FlakeFlagger: predicting flakiness without rerunning tests". https://doi.org/10.5281/zenodo.5014076
Benavoli A, Corani G, Mangili F (2016a) Should we really use post-hoc tests based on mean-ranks? J Mach Learn Res 17(1):152–161
Benavoli A, Corani G, Demšar J, Zaffalon M (2017b) Time for a change: A tutorial for comparing multiple classifiers through Bayesian analysis. J Mach Learn Res 18(1):2653–2688
Berry DM (2021) Empirical evaluation of tools for hairy requirements engineering tasks. Empir Softw Eng 26(6):1–77
Bishop CM (2006) Pattern recognition and machine learning. Springer, New York
Boyd K, Eng KH Jr (2013) C.D.P.: area under the precision-recall curve: point estimates and confidence intervals. In: European conference on machine learning and principles and practice of knowledge discovery in databases, LNCS, vol 8190. Springer, pp 451–466
Cawley GC, Talbot NL (2010) On over-fitting in model selection and subsequent selection bias in performance evaluation. J Mach Learn Res 11:2079–2107
Cleland-Huang J, Settimi R, Zou X, Solc P (2006) The detection and classification of non-functional requirements with application to early aspects. In: IEEE International requirements engineering conference, pp 39–48
Cleland-Huang J, Settimi R, Zou X, Solc P (2007) Automated classification of non-functional requirements. Requir Eng 12(2):103–120
Cleland-Huang J, Czauderna A, Gibiec M, Emenecker J (2010) A machine learning approach for tracing regulatory codes to product specific requirements. In: IEEE/ACM international conference on software engineering, pp 155–164
Cohen BH (2008) Explaining psychological statistics. Wiley, New York
Dalpiaz F, Dell'Anna D, Aydemir FB, Ćevikol S (2019) Requirements classification with interpretable machine learning and dependency parsing. In: IEEE International requirements engineering conference, pp 142–152
de Oliveira Neto FG, Torkar R, Feldt R, Gren L, Furia CA, Huang Z (2019) Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. J Syst Softw 156:246–267
Dell'Anna D, Aydemir FB, Dalpiaz F (2021) Supplementary material for "evaluating classifiers in SE research: the ECSER pipeline and two replication studies". https://doi.org/10.5281/zenodo.6266675
Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
Devlin J, Chang MW, Lee K, Toutanova K (2018) BERT: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805
Dong G, Liu H (2018) Feature engineering for machine learning and data analytics. CRC Press
Duboue P (2020) The art of feature engineering: essentials for machine learning. Cambridge University Press, Cambridge
Fagerholm F, Kuhrmann M, Münch J (2017) Guidelines for using empirical studies in software engineering education. PeerJ Comput Sci 3:e131
Fawcett T (2006) An introduction to ROC analysis. Pattern Recogn Lett 27(8):861–874
Flach P (2012) Machine learning: the art and science of algorithms that make sense of data. Cambridge University Press
Fu W, Menzies T, Shen X (2016) Tuning for software analytics: is it really necessary? Inf Softw Technol 76:135–146
Garousi V, Felderer M (2017) Experience-based guidelines for effective and efficient data extraction in systematic reviews in software engineering. In: International conference on evaluation and assessment in software engineering, pp 170–179
Garousi V, Felderer M, Mäntylä MV (2019) Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf Softw Technol 106:101–121
Ghotra B, McIntosh S, Hassan AE (2015) Revisiting the impact of classification techniques on the performance of defect prediction models. In: IEEE/ACM International conference on software engineering, pp 789–800
Goadrich M, Oliphant L, Shavlik JW (2006) Gleaner: creating ensembles of first-order clauses to improve recall-precision curves. Mach Learn 64(1–3):231–261
Good P (2013) Permutation tests: a practical guide to resampling methods for testing hypotheses. Springer Science & Business Media

Greener JG, Kandathil SM, Moffat L, Jones DT (2022) A guide to machine learning for biologists. Nat Rev Mol Cell Biol 23(1):40–55
Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. ACM SIGKDD Explor Newsl 11(1):10–18
Hall T, Beecham S, Bowes D, Gray D, Counsell S (2012) A systematic literature review on fault prediction performance in software engineering. IEEE Trans Softw Eng 38(6):1276–1304
Herbold S (2020) Autorank: a Python package for automated ranking of classifiers. J Open Source Softw 5(48):2173
Herbold S, Trautsch A, Trautsch F (2020) On the feasibility of automated prediction of bug and non-bug issues. Empir Softw Eng 25(6):5333–5369
Hey T, Keim J, Koziolek A, Tichy WF (2020a) Norbert: transfer learning for requirements classification. In: IEEE International requirements engineering conference, pp 169–179
Hey T, Keim J, Koziolek A, Tichy WF (2020b) Supplementary material of "NoRBERT: transfer learning for requirements classification. https://doi.org/10.5281/zenodo.3874137
Huff D (1993) How to lie with statistics. WW Norton & Company
Hutchinson B, Smart A, Hanna A, Denton E, Greer C, Kjartansson O, Barnes P, Mitchell M (2021) Towards accountability for machine learning datasets: practices from software engineering and infrastructure. In: ACM Conference on fairness, accountability, and transparency, pp 560–575
Japkowicz N, Shah M (2011) Evaluating learning algorithms: a classification perspective. Cambridge University Press
Jedlitschka A, Ciolkowski M, Pfahl D (2008) Reporting experiments in software engineering. Springer, London, pp 201–228
Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu TY (2017) LightGBM: a highly efficient gradient boosting decision tree. Advances in Neural Information Processing Systems 30
Kitchenham B (2004) Procedures for performing systematic reviews. Tech Rep. 2004. Keele University, Keele
Kitchenham B, Madeyski L, Budgen D, Keung J, Brereton P, Charters S, Gibbs S, Pohthong A (2017) Robust statistical methods for empirical software engineering. Empir Softw Eng 22(2):579–630
Korkmaz S, Goksuluk D, Zararsiz G (2014) MVN: an r package for assessing multivariate normality. R J 6(2):151–162
Kuhrmann M, Fernández DM, Daneva M (2017) On the pragmatic design of literature studies in software engineering: an experience-based guideline. Empir Softw Eng 22(6):2852–2891
Kurtanovic Z, Maalej W (2017) Automatically classifying functional and non-functional requirements using supervised machine learning. In: IEEE International requirements engineering conference, pp 490–495
Lever J (2016) Classification evaluation: it is important to understand both what a classification metric expresses and what it hides. Nat Methods 13(8):603–605
Li F, Horkoff J, Mylopoulos J, Guizzardi RSS, Guizzardi G, Borgida A, Liu L (2014) Non-functional requirements as qualities, with a spice of ontology. In: IEEE International requirements engineering conference, pp 293–302
Liu C, Gao C, Xia X, Lo D, Grundy J, Yang X (2021) On the reproducibility and replicability of deep learning in software engineering. ACM Trans Softw Eng Methodol (TOSEM) 31(1):1–46
Lones MA (2021) How to avoid machine learning pitfalls: a guide for academic researchers. CoRR arXiv:2108.02497
Luo W, Phung D, Tran T, Gupta S, Rana S, Karmakar C, Shilton A, Yearwood J, Dimitrova N, Ho TB et al (2016) Guidelines for developing and reporting machine learning predictive models in biomedical research: a multidisciplinary view. J Med Internet Res 18(12):e323
Mahadi A, Ernst NA, Tongay K (2022) Conclusion stability for natural language based mining of design discussions. Empir Softw Eng 27(1):1–42
Mardia KV (1970) Measures of multivariate skewness and kurtosis with applications. Biometrika 57(3):519–530
Menzies T (2001) Practical machine learning for software engineering and knowledge engineering. In: Handbook of software engineering and knowledge engineering: volume I: fundamentals. World Scientific, pp 837–862
Menzies T, Shepperd M (2019) "Bad smells" in software analytics papers. Inf Softw Technol 112:35–47
Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Bener A (2010) Defect prediction from static code features: current results, limitations, new approaches. Autom Softw Eng 17(4):375–407
Montgomery L, Damian D, Bulmer T, Quader S (2018) Customer support ticket escalation prediction using feature engineering. Requir Eng 23(3):333–355
Moser R, Pedrycz W, Succi G (2008) A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: IEEE/ACM International conference on software engineering, pp 181–190

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830

Petersen K, Feldt R, Mujtaba S, Mattsson M (2008) Systematic mapping studies in software engineering. In: International conference on evaluation and assessment in software engineering, pp 1–10

Petersen K, Vakkalanka S, Kuzniarz L (2015) Guidelines for conducting systematic mapping studies in software engineering: an update. Inf Softw Technol 64:1–18

Pinto G, Miranda B, Dissanayake S, d'Amorim M, Treude C, Bertolino A (2020) What is the vocabulary of flaky tests? In: International conference on mining software repositories, pp 492–502

Rajbahadur G, Wang S, Kamei Y, Hassan AE (2021) Impact of discretization noise of the dependent variable on machine learning classifiers in software engineering. IEEE Trans Softw Eng 47(7):1414–1430

Ralph P, Baltes S, Bianculli D, Dittrich Y, Felderer M, Feldt R, Filieri A, Furia CA, Graziotin D, He P, Hoda R, Juristo N, Kitchenham BA, Robbes R, Méndez D, Molleri J, Spinellis D, Staron M, Stol K, Tamburri DA, Torchiano M, Treude C, Turhan B, Vegas S (2020) Empirical standards for software engineering research. CoRR arXiv:2010.03525

Read J, Pfahringer B, Holmes G, Frank E (2011) Classifier chains for multi-label classification. Mach Learn 85(3):333–359

Salzberg SL (1997) On comparing classifiers: pitfalls to avoid and a recommended approach. Data Min Knowl Disc 1(3):317–328

Sheskin DJ (2020) Handbook of parametric and nonparametric statistical procedures. CRC Press

Siebert J, Joeckel L, Heidrich J, Nakamichi K, Ohashi K, Namba I, Yamamoto R, Aoyama M (2020) Towards guidelines for assessing qualities of machine learning systems. In: International conference on the quality of information and communications technology. Springer, pp 17–31

Sorower MS (2010) A literature survey on algorithms for multi-label learning. Tech. rep., Oregon State University

Stapor K (2017) Evaluating and comparing classifiers: review, some recommendations and limitations. In: International conference on computer recognition systems. Springer, pp 12–21

Sullivan GM, Feinn R (2012) Using effect size—or why the P value is not enough. J Grad Med Educ 4(3):279–282

Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2019) The impact of automated parameter optimization on defect prediction models. IEEE Trans Softw Eng 45(7):683–711

Tanwani AK, Afridi J, Shafiq MZ, Farooq M (2009) Guidelines to select machine learning scheme for classification of biomedical datasets. In: European conference on evolutionary computation, machine learning and data mining in bioinformatics. Springer, pp 128–139

Tian C, Manfei X, Justin T, Hongyue W, Xiaohui N (2018) Relationship between omnibus and post-hoc tests: an investigation of performance of the F test in ANOVA. Shanghai Arch Psychiatry 30(1):60

Tran N, Schneider JG, Weber I, Qin A (2020) Hyper-parameter optimization in classification: to-do or not-to-do. Pattern Recogn 103:107245

Wang AYT, Murdock RJ, Kauwe SK, Oliynyk AO, Gurlo A, Brgoch J, Persson KA, Sparks TD (2020) Machine learning for materials scientists: an introductory guide toward best practices. Chem Mater 32(12):4954–4965

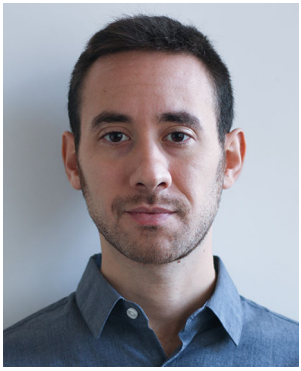
Wieringa RJ (2014) Design science methodology for information systems and software engineering. Springer, London

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer Science & Business Media

Yao J, Shepperd M (2020) Assessing software defection prediction performance: why using the matthews correlation coefficient matters. In: International conference on the evaluation and assessment in software engineering, pp 120–129

Zhang D, Tsai JJ (2003) Machine learning and software engineering. Softw Qual J 11(2):87–119

Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, pp 91–100



Dr. Davide Dell'Anna is a postdoc in the Department of Control and Operations of Delft University of Technology, The Netherlands. He received his Ph.D. degree from Utrecht University, The Netherlands. His research mainly focuses on the automated control and coordination of autonomous and intelligent systems in contexts such as smart cities, socially assistive robotics, and social simulation. He regularly serves as a program committee member of conferences such as REFSQ and AAMAS. He co-organized the NLP4RE workshop of REFSQ 2022, and was involved in the organization of REFSQ 2018 and RE 2023.

Dr. Fatma Başak Aydemir is a faculty member in the Department of Computer Engineering at Boğaziçi University, Istanbul, Turkey. She received her Ph.D. degree from the University of Trento, Italy. Her research focuses on applying artificial intelligence methods to increase the efficiency of requirements engineering processes and artifacts. She regularly serves as a program committee member for RCIS, ER, REFSQ, and RE conferences. She participated in the H2020 PACAS project and currently leads the Requirements Engineering for Digital Transformation (RE4DigiTR) (BIDEB 2232 118C255) project funded by The Scientific and Technological Research Council of Turkey.

Dr. Fabiano Dalpiaz is a faculty member in the Department of Information and Computing Sciences at Utrecht University in the Netherlands. He is principal investigator in the department's Requirements Engineering lab. Dalpiaz acts and acted as program co-chair of RE 2023, REFSQ 2021, RCIS 2020, was the organization chair of REFSQ 2018 conference, and is an associate editor for the Requirements Engineering Journal and the Business & Information Systems Engineering Journal.