Software and Systems Modeling
https://doi.org/10.1007/s10270-021-00929-3

THEME SECTION PAPER

ModelSet: a dataset for machine learning in model-driven engineering

José Antonio Hernández López[1] · Javier Luis Cánovas Izquierdo[2] · Jesús Sánchez Cuadrado[1]

Received: 19 March 2021 / Revised: 8 August 2021 / Accepted: 14 September 2021

Abstract
The application of machine learning (ML) algorithms to address problems related to model-driven engineering (MDE) is currently hindered by the lack of curated datasets of software models. There are several reasons for this, including the lack of large collections of good quality models, the difficulty to label models due to the required domain expertise, and the relative immaturity of the application of ML to MDE. In this work, we present ModelSet, a labelled dataset of software models intended to enable the application of ML to address software modelling problems. To create it we have devised a method designed to facilitate the exploration and labelling of model datasets by interactively grouping similar models using off-the-shelf technologies like a search engine. We have built an Eclipse plug-in to support the labelling process, which we have used to label 5,466 Ecore meta-models and 5,120 UML models with its category as the main label plus additional secondary labels of interest. We have evaluated the ability of our labelling method to create meaningful groups of models in order to speed up the process, improving the effectiveness of classical clustering methods. We showcase the usefulness of the dataset by applying it in a real scenario: enhancing the MAR search engine. We use ModelSet to train models able to infer useful metadata to navigate search results. The dataset and the tooling are available at https://figshare.com/s/5a6c02fa8ed20782935c and a live version at http://modelset.github.io.

Keywords Dataset · Machine learning · Model-driven engineering

1 Introduction

Model-driven engineering (MDE) is a software development paradigm that advocates the use of models as active elements in the development cycle. Such models can be created with general-purpose modelling languages (e.g., UML) or using a domain-specific language (DSL). At the same time, artificial intelligence (AI) and machine learning (ML), its most current branch, have shown their potential to enhance software engineering approaches in many areas [2,5,50,56], but their application for addressing tasks in the modelling domain is still relatively recent. For instance, a neural network has been used to classify meta-models into application domains [39], clustering techniques have shown its usefulness in organizing collections of models [9,11], and graph kernels have been proposed as a means to characterize similar models [17].

An important limitation of current applications of ML for MDE is the lack of large datasets (either labelled or unlabelled) from which rich ML models can be trained. While there exist a few model datasets freely available, their quality is not adequate. Most datasets have a small size (e.g., 555 labelled meta-models [7,39]), while others are not curated (e.g., the UML dataset proposed in [47] contains 90,000 models, but neither availability nor navigability are guaranteed and only 23,000 models can be downloaded, and only 3000 of them are EMF-valid; the rest crashed or it is not possible determine the tool to edit them easily). This scenario contrasts with the situation in other application areas. For instance, one of the milestones of the ML community was the creation of large datasets, like ImageNet [18], which contains thousands of manually labelled images. In the software engineering domain, many approaches (either ML-based or not) rely on public datasets designed for concrete applica-

Communicated by L. Burgueño, J. Cabot, M. Wimmer & S. Zschaler.

✉ José Antonio Hernández López
joseantonio.hernandez6@um.es

Javier Luis Cánovas Izquierdo
jcanovasi@uoc.edu

Jesús Sánchez Cuadrado
jesusc@um.es

1 Facultad de Informática, Universidad de Murcia, Murcia, Spain

2 UOC - IN3, Castelldefels, Spain

tions. A well-known example is the Defects4J [28] dataset, which has fueled research in program repair (e.g., [15,36]).

In this paper, we tackle the creation of labelled datasets of software models. The main difficulty is that labelling a single model can be hard and time consuming due to the domain expertise required to explore and understand the model and assign a proper label. To address this shortcoming, we have devised an interactive, semi-automatic labelling method based on grouping similar models using a search engine. We have created an Eclipse plug-in as a concrete instantiation of the method, including features like automatic model grouping by similarity, visualizations, and label review. We have used this tool to label Ecore meta-models and UML models with their category (e.g., DSLs for Petri nets, UML for modelling an ATM). Moreover, we provide a number of additional labels, like tags to describe the main topics of a model. Altogether, this paper makes the following contributions:

- We propose a methodology to speedup the process of labelling software models. It is accompanied with a supporting tool implemented as an Eclipse plug-in.
- We contribute the first version of ModelSet, a large dataset of labelled models which comprises 5,466 Ecore meta-models and 5,120 UML models.
- To assess the usefulness of the labelled part of the dataset in practice, we show three applications of different nature and apply them to enhance the MAR search engine[1] [35]: detection of dummy models, single-label classification to infer model categories, as well as multi-label classification to infer relevant tags.

Ultimately, our aim is that this work fosters new interesting applications for MDE, in particular, related to machine learning and empirical studies, and it becomes a milestone for the development of other datasets for models (or extensions of this one).

1.1 Organization

Section 2 motivates the main challenges when creating datasets. Section 3 presents the methodology to create datasets of models. Section 4 describes ModelSet. Section 5 shows a case study using ModelSet. Section 6 presents the related work and Sect. 7 concludes and describes the further work.

2 Background and motivation

Machine learning is a branch of artificial intelligence which encompasses techniques to make computers learn from data. The availability of high-quality datasets is essential for the application of ML techniques in a given domain. In this paper, our target domain is software modelling, and therefore our focus is the creation of a dataset consisting of models. Our underlying motivation is to push the development of ML methods in the software modelling domain.

2.1 Machine learning and datasets

In general, ML techniques can be classified according to the amount and shape of the provided data (i.e., the dataset).

In *supervised* learning, the dataset includes the labels that the ML algorithm needs to learn. A particular type of learning task is *regression*, in which the system learns to predict a numerical variable (e.g., a person height). Another type of task is *classification*, in which the system learns to predict a categorical variable (e.g., hair color). The existence of a dataset, such as the one proposed in this work, will enable classification applications associated with the management of large model repositories [19], like attaching tags automatically to models to help the user's navigation and the automatic detection of anomalous models to discard them, among others. Moreover, the labels of a dataset are not only useful for supervised tasks, but they also play a role in stratified sampling to make sure that, when the data is split, the models within each split preserve the percentage of sample for each class.

In contrast, in *unsupervised* learning, the dataset does not need to be labelled since the system tries to identify patterns by itself. A typical task is clustering, in which the system identifies groups of similar examples according to some criteria [11]. Other unsupervised tasks include learning modelling patterns which arises in a dataset, for instance, with the aim of creating smart modelling environments, including recommenders for model editors [31,33] and supporting the interaction with bots [42,43]. It is important to note that a labelled dataset is useful for testing clustering techniques since there are quality metrics that require the ground truth to be computed (e.g., Rand Index, NMI, AMI, etc). On the other hand, reinforcement learning approaches have been used in the modelling domain to apply automatic repairs [26].

Regarding clustering techniques, many algorithms have been proposed. Among the most popular ones, there are K-Means, hierarchical clustering and DBSCAN. All of them requires a distance measure and the first two admit the number of clusters as hyperparameter. DBSCAN does not require setting the number of clusters upfront, but it requires other parameters. We use these three algorithms as a baseline to compare our labelling method.

Altogether, the existence of high-quality datasets is a pre-requisite for applying some of the ML techniques mentioned above to modelling. Moreover, the availability of curated datasets will enable the development of model

[1] http://mar-search.org.

analytics and related empirical approaches (e.g., maturity analysis [14,46], model characterization [24] or technical debt assessment [21,27]).

2.2 Challenges for creating labelled modelling datasets

The technical underpinning of our work is the Eclipse Modeling Framework (EMF) [44], which is a *de facto* industrial standard to create modelling languages. Ecore is the meta-modelling language provided by EMF and allows us to represent the main concepts and relationships of an application domain or a DSL (i.e., its abstract syntax). On the other hand, UML is a well-known modelling language proposed by the OMG [40], for which there is an EMF-based implementation. The language includes different diagram definitions (e.g., class, interaction, or use case diagrams) and has become the main general-purpose modelling language in software development. Thus, we focus on creating labelled datasets of Ecore and UML models extracted from public sources, but the techniques that will be proposed in this work can be applicable to other types of models as well.

As noted in the introductory section, there is a lack of datasets specific to software models, which hinders the application and adaptation of existing ML algorithms to deal with software models. This contrasts which the situation in other domains, in which there has been much research focused on the creation of datasets [48]. Many labelled datasets are general-purpose in the sense that most individuals can contribute or correct labels. This fact facilitates the use of crowdsourcing tools like Amazon Mechanical Turk (e.g., images [18] or questions [45]). However, one of the main challenges in labelling software models is that it requires modelling expertise and specialized tooling for inspection. For instance, to label models in a dataset of Ecore meta-models one needs to have experience about EMF and its ecosystem. Similarly, annotating UML diagrams requires knowledge about the different UML diagrams.

Another important challenge is that, even with the required experience, labelling a single model can be very time-consuming. Let us suppose that we are labelling Ecore meta-models, and we come across a model similar to the one in Fig. 1a. For many modellers it might be challenging to find out at first glance what type of model it represents[2]. A good strategy is to look for more information from the tool source. In this case, neither the GitHub README file nor the website of the tool (osate2) includes specific information about this meta-model. Therefore, we must spend some time looking in alternate sources (e.g., Wikipedia), or we might try to find

[2] Unless the modeller is familiar with safety modelling techniques. In our case, only one of the authors of the paper knew what a fault tree is, but only superficially.

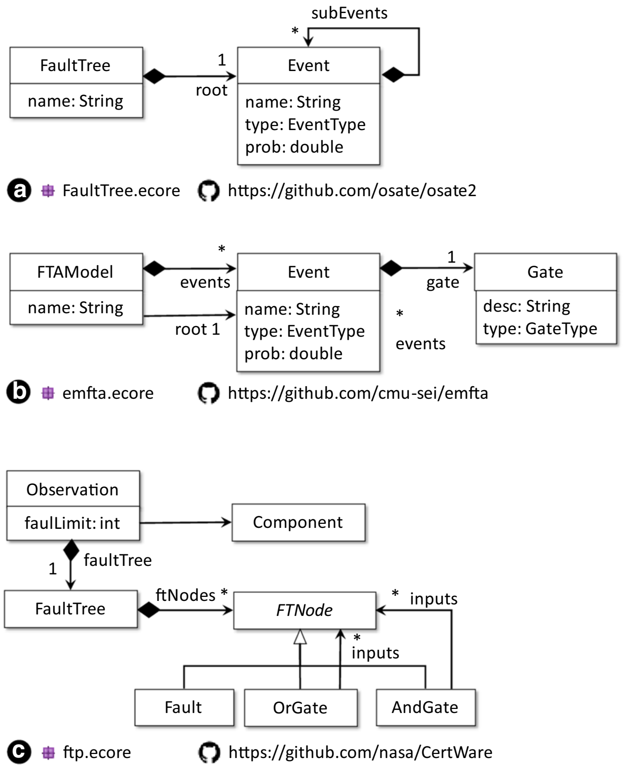


Fig. 1 Excerpts of fault tree meta-models

similar meta-models in the dataset in order to find out more about this modelling domain. Only when we understand the usage of the meta-model and we have explored similar meta-models, we are ready to assign a label that properly identifies the category of this and similar meta-models. In this example, one could find meta-models like the ones shown in Fig. 1b and Fig. 1c. Using these two additional examples and the information in the respective GitHub pages we could learn that a fault tree is a formalism used in safety analysis in which the key concepts are events which may be present in a fault or hazard, and which are arranged in a tree structure according to different types of gates and with probabilities assigned. If we are assigning a category to the models, we could annotate these three models with fault-tree. Moreover, we might want to add additional tags like hazards and safety.

In our experience creating labelled datasets of models, this situation happens often and it is a major hindrance to its construction. To address this problem, our key observation is that the time spent understanding a model is likely unavoidable. However, if we are able to show similar models at once, it would be easier to understand these models by comparison, and it is possible to label similar models in a row, thus speeding up the whole process. Hence, we have devised a semi-automatic labelling method that applies this idea, which is explained in the next section.

3 Building model datasets

Building a dataset of software models generally implies two main steps. First, models must be retrieved from known model repositories, validated and organized. A dataset like this can already be useful, for instance, for software analytics, and to apply unsupervised ML. If we aim at using the dataset for supervised ML we need to label the models, but many times this requires manual intervention, which is a time-consuming task, as discussed in the previous section. Thus, to address the labelling task, we have devised a generic methodology aimed at grouping likely similar models to speed up the process by allowing the user to label several models at once.

3.1 GMFL: a greedy methodology for fast labelling

Our methodology is based on the observation that the effort of labelling can be split into two tasks. The costly and essentially complex task is the identification of the proper label for a category of models not seen before (e.g., fault tree models). The other task is to find similar models for which the same label is adequate. If one is labelling a given model, the sooner such similar models are found the better, since there is less cognitive load in labelling them because there is no need to re-think about this type of models and labelling becomes a matter of reassigning the label.

A conventional approach could be to perform clustering using widely known techniques like K-means or hierarchical clustering (HC) [8,11], and use the clusters to explore the dataset. The disadvantage is that the number of clusters needs to be set upfront, but this value is generally not known in advance. Although there exist techniques to estimate it, they are computationally expensive and still provide sub-optimal results for large and irregular datasets. Some clustering algorithms, like DBSCAN [20], do not require setting the number of clusters, but other parameters are needed. A key disadvantage of using clustering methods to help in the labelling of datasets is that they require to split the data in groups upfront, and thus give little control to the user. Even if a dendrogram is used to perform the splits interactively and assign labels, if a model is deemed “outside” a cluster by the person in charge of labelling, it will not appear again when labelling other clusters.

This situation is illustrated in Fig. 2. Let us suppose that we are given a set of clusters to label the models contained in each one in turn. We could start labelling Cluster #5 with category fault-tree as mentioned before (because FaultTree.ecore, emfta.ecore and ftp.ecore represent fault tree models). This cluster is formed due to the coincidence of words like FaultTree, Event, Gate. However, the model ui_events.ecore is of different nature and requires a different category, although it includes similar words like Event. We may label the model

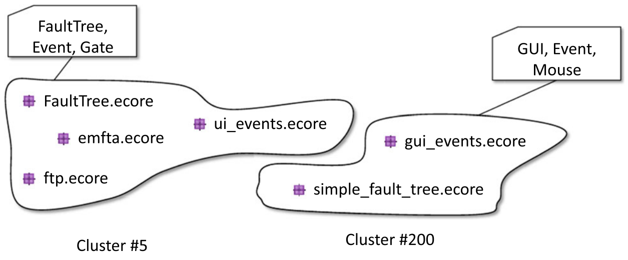


Fig. 2 Incorrect grouping of models in clusters. Model ui_events.ecore should not belong to cluster #5, and model simple_fault_tree.ecore should not belong to cluster #200

with category gui when processing the current cluster, but it means that we will label it in isolation. The trouble is that the model will not appear again when manually processing other clusters, thus if we do not label it now we lose the opportunity to label it. Similarly, when we arrive at cluster #200, we will find simple_fault_tree.ecore and we will need to remember the category that we have previously used when labelling cluster #5.

To address this issue, we have devised a greedy, interactive algorithm intended to perform a form of dynamic, user-driven clustering, which is outlined in Algorithm 1. Given a dataset consisting of a set of models $\mathcal{M} = \{m_1, \ldots, m_t\}$, we want the user to assign each model a main label plus a set of additional labels.[3] At the end of the process, we will have a set of labels $\mathcal{L} \neq \emptyset$ and a set of tuples with the same cardinality of $\mathcal{M}$, that is $\mathcal{T} = \{(m_1, l_1), \ldots, (m_t, l_t)\}$ where $l_i \in \mathcal{L}$ for all $1 \leq i \leq t$. The algorithm attempts to maximize the number of models with the same label assigned in a row, without changing the focus to another label. Thus, we define a *labelling streak* as a set of models that have been assigned the same label without interruption. As noted before, the idea is to help model identification by analyzing several models together, and also amortize this cost by labelling them at once.

The algorithm has three customizable parts, which are indicated by comments. The rationale is to allow the adaption of the algorithm to the concrete artifact being labelled and the available technology in order to make the implementation cost as modest as possible since the real value is in the labels. These three parts are explained next focusing on how we apply them to software models.

3.1.1 Exploration order

sets the order in which the user will be presented with new unlabelled models. A simple approach is to make it ran-

[3] For the sake of simplicity in the presentation we show only the main label, but it can be extended by replacing label l with a tuple $(l, additional)$ where l is the main label and $additional$ represent a list of additional labels.

```
   Data: $\mathcal{M}$: Models in the dataset
   Result: $\mathcal{T}$: set of labelled models along their labels
1  $\mathcal{T} \leftarrow \emptyset$
2  while $\mathcal{M}$ has unlabelled models do
3      // Exploration order
4      $m \leftarrow$ pick unlabelled model from $\mathcal{M}$
5      Manually inspect $m$ to assign label $l$
6      $\mathcal{T} \leftarrow \mathcal{T} \cup \{(m, l)\}$
7      $\mathcal{V} \leftarrow \{m\}$
8      while not isEmpty($\mathcal{V}$) do
9          $m \leftarrow pop(\mathcal{V})$
10         // Similar model retrieval
11         $\mathcal{F} \leftarrow$ search for non-labelled models $m_1, ..., m_n$ sorted by
           similarity to $m$;
12         Manually inspect $m_1, ..., m_n$ to assign label $l$
13         $\mathcal{A} = \{(m_1, l), ..., (m_n, l)\}$
14         $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{A}$
15         // Model retrieval refinement
16         m' $\leftarrow$ pick relevant models from $\mathcal{A}$
17         add m' to $\mathcal{V}$
18     end
19 end
```

Algorithm 1: Sketch of our labelling algorithm. $\mathcal{T}$ is the resulting set of pairs $(model, label)$, $\mathcal{V}$ keeps unvisited models.

dom (e.g., choose an unlabelled model randomly). Other configurations are possible to follow specific strategies, like splitting models into distinct groups to minimize conflicts when labelling collaboratively. We opted to explore first those models which may have more similar models in the dataset. The rationale was to gain confidence at the beginning by having many similar examples to understand how to label, as well as encouraging ourselves by feeling productive.

To define this order we converted the models to text documents by considering the values of string attributes (i.e., typically names of model elements). We use a TF-IDF approach to transform each document (model) into a vector. For each document, we define its density as the average similarity of the k nearest neighbors (the k more similar documents using cosine similarity), and we sort the list in descending density order. In this way, models with more similar models are selected first for exploration.

3.1.2 Similar model retrieval

. Given a model m selected to be labelled, a set of potentially similar models are retrieved, $\mathcal{F}$, so that the user may decide whether the same label is applicable to them (i.e., this is the basis of a streak). We propose to use off-the-shelf search engines to perform model similarity. The idea is to approximate similarity by the notion of relevance offered by search engines, so that the user inspects a prioritized list of models where the first ones are more similar to m.

To label Ecore meta-models, we have used MAR [35], a search engine of models which takes into account the model

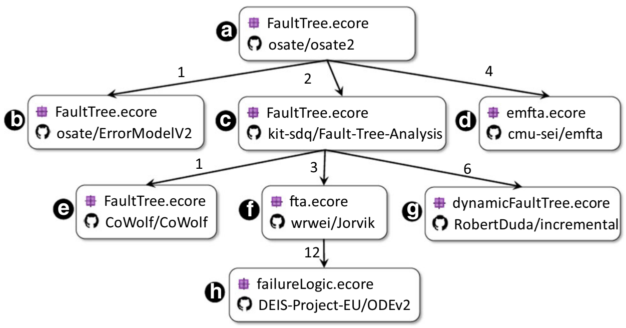


Fig. 3 Algorithm execution starting with a fault tree model. Edges are annotated with the position in the search in which the result appeared

structure to perform accurate searches. It takes a model as an input query and returns a ranked list of models sorted by their relevance, so that more similar models are ranked first. The first models of the list will be very similar to m so it is highly probable that they are assigned exactly the same label as m. At some point the models in the list will not share the same label as m, and the user may decide to move on to the next suggestion (see *retrieval refinement* below).

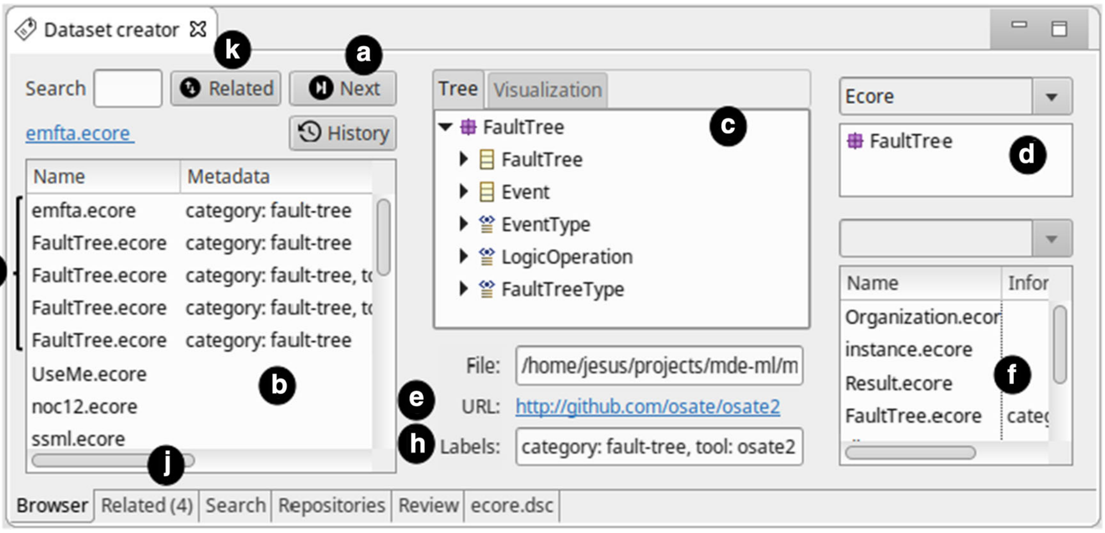
If there is not an available search engine specific for the type of artifact being labelled, it is still possible to obtain good results using a text-based search engine like Whoosh [54] or Apache Lucene [6]. In the case of software models, the strategy would be to generate text documents with the values of string attributes of the model and index these documents using the search engine. We have used this strategy to label UML models using Whoosh.

3.1.3 Model retrieval refinement

. This part aims to *continue* the current streak by identifying additional models that can still be annotated with the same label used for the previous models. We are interested on selecting one or more models already labelled as candidates for a new similarity search.

In our case, we select for refinement all models annotated by the user[4] (e.g., all models of $\mathcal{A}$ of line 11). Figure 3 illustrates the behavior of the algorithm for the running example by showing the models obtained as part of the search (the edges are annotated with the rank of the model in the search). First, a model is picked to start a new streak (model ⓐ). From this model, a ranked list of unlabelled models to explore is obtained. We label with fault-tree the original model plus three models in the list (models ⓑ, ⓒ and ⓓ which were ranked first, second and fourth). This starts a streak of four elements, which is now further refined by using the already labelled

[4] This is possible when the search engine is responsive enough to handle this load interactively. In other cases, only a few models should be selected.



Name	Metadata
emfta.ecore	category: fault-tree
FaultTree.ecore	category: fault-tree
FaultTree.ecore	category: fault-tree, to
FaultTree.ecore	category: fault-tree, to
FaultTree.ecore	category: fault-tree
UseMe.ecore	
noc12.ecore	
ssml.ecore	

File: /home/jesus/projects/mde-ml/m

URL: http://github.com/osate/osate2

Labels: category: fault-tree, tool: osate2

Name	Infor
Organization.ecor	
instance.ecore	
Result.ecore	
FaultTree.ecore	categ

Fig. 4 Dataset creator. Main labelling window

models as input queries for new searches. In the example, we focus on ⓒ. Three new models can be labelled with the category fault tree, and thus they belong to the same streak (which has now seven models). If we take model ⓕ the search now produces several non-related models, except a model ranked in position 12 (model ⓖ). If the user selects it for exploration, the streak would be expanded. If not, the model would appear later within another streak.

The goal of our approach is to let the user interactively expand the set of models that may end up with the same label, ideally without interruption (i.e., in the same streak). This is a form of dynamic, interactive clustering. It mirrors the style of DBSCAN [20], but with no need to establish any parameters (ϵ and the minimum number of points required to form a dense region, *minPts*) because the user interactively creates a cluster (labelling a model in a streak) and expands it (labelling models in related searches).

3.2 Tool support

We have implemented an Eclipse plug-in which provides a concrete instantiation of GMFL to create model datasets. The plug-in relies on EMF and extends Eclipse to provide a rich user interface to follow the methodology steps. The current implementation supports the connection to a MAR server or a Whoosh engine to perform the searches. The labelling data is stored in a SQLite database file and there is a dedicated Java API to easily access the dataset programmatically.

Figure 4 shows a screenshot of the main labelling screen. A labelling session starts by clicking the Next button (marker ⓐ). It re-starts the algorithm by picking the next unlabelled model in the exploration order (emfta.ecore in this case), and it shows the user this model followed by a ranked list of potentially similar unlabelled models (marker ⓑ) based on the chosen similarity method (using the MAR search engine in this case). The user starts inspecting the list from the beginning and tries to label the models. A tree view to inspect each model individually is available (marker ⓒ) plus specific visualizations crafted for each type of model (e.g., UML class diagrams, state machines, activity diagrams and interaction diagrams). To navigate large models an outline is also presented (marker ⓓ). To help understand the model we present two sources of information when available: (i) the URL of the model (marker ⓔ), to explore it and read some documentation (if any); and (ii) other models available in the same project (marker ⓕ, which shows models obtained from the same GitHub repository). With this information, the user writes one or more labels for the model with the format label: value (marker ⓗ). For ModelSet, we have the convention of using the label category as the primary label, and have additional labels like tool and tags.

The user inspects the first models and try to do a *labelling streak* (the tool provides shortcuts to navigate and duplicate labels easily). For instance, Fig. 4 (marker ⓘ) shows the beginning of a streak of 5 models. Each time that a model of the list is labelled, the system automatically schedule a search in the background to find other similar models, in an attempt to provide more unlabelled models related to it. The results of the search are stacked in another tab (marker ⓙ) as they become available and the user can inspect them. By clicking on the "Related" button (marker ⓚ) the current model list is replaced by the top of the search stack. The user repeats the process with this new list of related models. This implementation of the *model retrieval refinement* step attempts to use the time that the user is inspecting models to execute the inner loop of the algorithm in parallel and asynchronously.

In addition, the tool includes a GUI for on-demand searches (*Search* tab) which is often useful to inspect models similar to a given one. Moreover, the *Review tab* provides a GUI to inspect the labels and the models, to refactor labels, to export the data to CSV files and to compute statistics. Figure 5 shows a number of label values (for the *category* label in this case) and for the particular case of fault-tree the set of models that has been annotated with this value.

3.3 Evaluation

We want to systematically evaluate GMFL with respect to its ability to enable *labelling streaks*, that is, to produce lists of similar models that the user would label together. To this end, we have performed a simulation of the labelling process using the labels assigned to the models in the dataset to emulate the action performed by a real user. To perform the simulation, we use the category of the model as its main label (see Sect. 4). The simulation emulates the manual actions in Algorithm 1 (lines 5 and 12) with a lookup in the dataset to obtain the category of the model and pretends that the model is annotated with such category. The procedure is sketched in Algorithm 2. The main idea of the simulation is that once a model is annotated with category c_{base} using the set of already labelled models $\mathcal{T}$ (see line 6), we count the number of times that subsequent models in the sorted search results (obtained in line 13) are annotated with the same category. Each time that this happens (line 20) we add the model to the current streak (line 21). When subsequent models are not annotated with the same category (line 23), we register the number of times this happens, referred to as $window$ (line 24). We define a maximum value for $window$, called $WindowSize$, which represents the number of models without the same label that we are allowed to skip before finishing a streak (controlled in lines 26–27). This emulates the behavior of a user inspecting models in the result list until she or he finds that there is no point in further inspecting the current list because they are now too dissimilar.

We propose two evaluation metrics. The *streak size* S is the number of models annotated in a row with the same category allowing a certain $WindowSize$ tolerance value. For instance, in Fig. 3 using as tolerance $WindowSize = 2$ the streak size is 6 (i.e., includes models a–f). We report standard statistics S_{avg} and S_{max}. The other metric is *repetition size*, that is, the number of times that a streak re-appears with the same label. We report the ratio of the repetition size per category, R_{cat}. Ideally, R_{cat} is 1, meaning that all labels are assigned to the corresponding models in a single streak.

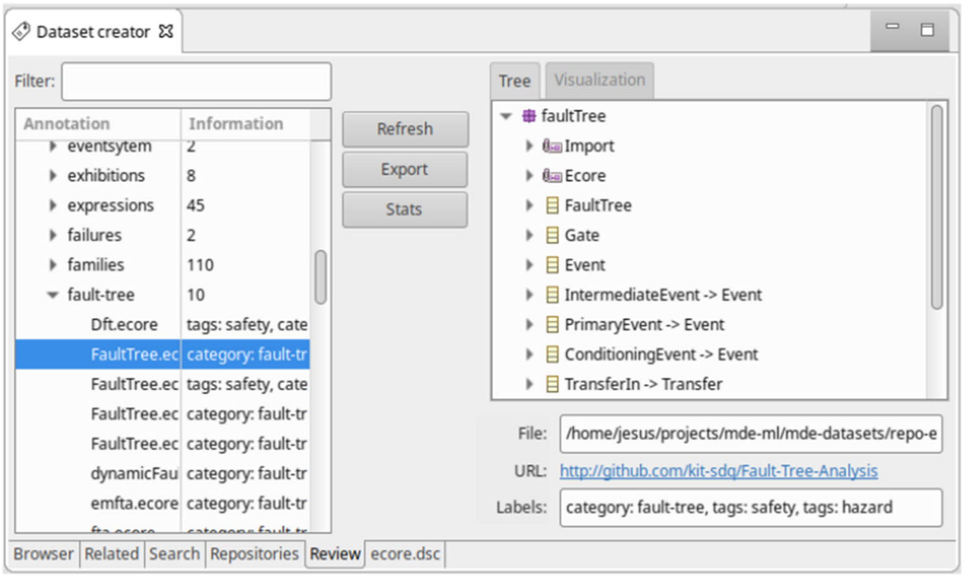
As a baseline we have simulated the algorithm using a completly random method (model selection is random and search returns random models), which is roughly equivalent to what a user could do without any method. We also compare against using three clusterings approaches based on transforming models into vectors using TF-IDF: K-means (with TF-IDF vectors normalized), hierarchical clustering (with cosine distance and complete linkage), and DBSCAN (with cosine distance). To establish the number of clusters we use the number of categories already identified, that is, we model the best scenario for these algorithms which is a perfect estimation of the number of clusters[5].

```
   Data: $\mathcal{M}$: Models in the dataset
   Data: $\mathcal{T}$: Set of labelled models along their labels
   Result: $\mathcal{S}$: list of created streaks
 1 // Simulate the labelling of the (already
      labelled) models in the dataset
 2 while $\mathcal{M}$ has unlabelled models do
 3     // Exploration order
 4     $m \leftarrow$ pick unlabelled model from $\mathcal{M}$
 5     // Simulate a user annotating the model
 6     $c_{base} \leftarrow$ get category for $m$ through $\mathcal{T}$
 7     label $m$ with $c_{base}$
 8     $s \leftarrow$ create streak for category $c$ with size 1
 9     add $s$ to $\mathcal{S}$
10     $\mathcal{V} \leftarrow \{m\}$
11     $\mathcal{A} \leftarrow \emptyset$
12     while not isEmpty($\mathcal{V}$) do
13         $m \leftarrow pop(\mathcal{V})$
14         // Similar model retrieval
15         $\mathcal{F} \leftarrow$ search for non-labelled models $m_1, ..., m_n$ sorted by
           similarity to $m$;
16         $window \leftarrow 0$
17         foreach $m_i$ in $\mathcal{F}$ do
18             $c \leftarrow$ get category for $m_i$ through $\mathcal{T}$
19             label $m_i$ with $c$
20             if $c_{base} = c$ then
21                 increase size of streak $s$ by 1
22                 $window \leftarrow 0$
23             else
24                 $window \leftarrow window + 1$
25             $\mathcal{A} = \mathcal{A} \cup \{(m_i, c)\}$
26             if $window > WindowSize$ then
27                 break
28
29         end
30         // Model retrieval refinement
31         m' $\leftarrow$ pick relevant models from $\mathcal{A}$
32         add m' to $\mathcal{V}$
33     end
34 end
```

Algorithm 2: Sketch of the simulation algorithm. $\mathcal{T}$ is the resulting set of pairs $(model, label)$, $\mathcal{V}$ keeps unvisited models.

Table 1 shows the main results of the evaluation, using $WindowSize = 3$. A random method has a very poor performance since the streak size is typically one, meaning that the user is often "jumping" between labels (i.e., $R_{cat} = 19$ means that each category is re-visited 19 times before it is completely annotated). DBSCAN also shows a poor performance because it identifies too many models as noise. Using

[5] For DBSCAN we adjusted the parameters until we found the expected number of categories.



Annotation	Information
eventsytem	2
exhibitions	8
expressions	45
failures	2
families	110
fault-tree	10
Dft.ecore	tags: safety, cate
FaultTree.ec	category: fault-tr
FaultTree.ec	tags: safety, cate
FaultTree.ec	category: fault-tr
FaultTree.ec	category: fault-tr
dynamicFau	category: fault-tr
emfta.ecore	category: fault-tr

File: /home/jesus/projects/mde-ml/mde-datasets/repo-e

URL: http://github.com/kit-sdq/Fault-Tree-Analysis

Labels: category: fault-tree, tags: safety, tags: hazard

Fig. 5 Dataset creator. Review window

Table 1 GMFL evaluation results

	ECORE			UML		
	S_{avg}	S_{max}	R_{cat}	S_{avg}	S_{max}	R_{cat}
Random	1.03	5	19.53	1.06	4	29.95
K-Means	3.51	93	5.74	3.76	373	8.41
HC	3.98	165	5.06	2.57	209	12.30
DBSCAN	1.46	53	13.76	1.26	84	24.99
GMFL	4.45	183	4.52	4.58	277	6.90

K-means or hierarchical clustering (HC), the performance increases (S_{avg} is larger and R_{cat} is smaller), but this is the best possible scenario in which the number of categories is already known. In large collections of models, like our case, this approach is not possible since the categories are discovered as the dataset is explored. In this setting, GMFL provides even greater performance than clustering approaches without the need of computing the clusters in advance. As can be observed, both for Ecore and UML the values of the average streak size (S_{avg}) is larger than other methods, meaning that the user can label more models in a row, and more importantly the number of times that categories reappear is smaller (R_{cat}), thus reducing the cognitive load in the process.

4 ModelSet: a dataset for MDE

In this section, we describe MODELSET, a large labelled dataset of software models, which we have built applying GMFL and using the provided tool support. We first explain how we collected the set of models to be labelled, as well as the labelling process we have followed, and then we detail the contents of the dataset. We end this section with an assessment of the experience creating MODELSET.

4.1 Collection process

We have collected models from GitHub and GenMyModel repositories. In particular, we have retrieved Ecore and UML models serialized as XMI files. In the first case, we used GitHub's public API to search for files with extension .ecore, which corresponds to Ecore models. As GitHub Search API returns a maximum of 1000 elements per query, we performed searches iteratively slighlty varying the file size (e.g., 100–124 bytes, 125–149 bytes, etc.) This method was intended to ensure that the number of returned elements is within the query limits. We kept querying the API until no more models were returned. On the other hand, GenMyModel is an online modelling service which hosts thousands of models of several types, like UML, Entity-Relationship, Ecore models, etc. It provides a public API[6] to query the catalogue and download the files selectively by type. In our case, we focused on the UML models available at this service.

We collected 83,009 valid Ecore models from which we discarded duplicated files by computing its MD5 hash. This produced 17,694 files which were the input of the labelling process. For UML, we downloaded 96,370 models from GenMyModel. It is not possible to compare file contents to discard duplicates because GenMyModel adds project specific metadata to each file. Therefore, we discarded small models which we found likely to be duplicates (e.g., example models) or just automatically created templates (e.g., a very simple model created by GenMyModel for the user to

[6] The API's entry point is https://app.genmymodel.com/api/projects/public.

complete). We also discarded models which contain non-latin characters since, unfortunately, the authors do not have enough knowledge to inspect models written in languages like Korean, Arabic, etc. Finally, we considered 53,266 UML models.

4.2 Labelling definition and process

The main label of our dataset is *category*, which represents a type of models sharing a similar application domain. For instance, in the running example the different variations of a fault tree are labelled with the fault-tree category, since all of them define a modelling language with a similar application. Given that Ecore is a meta-modelling language, many categories represent technical domains (e.g., relational models, feature models, etc.) but there are also domain models (e.g., company). On the other hand, in UML the values for the category label typically represent non-technical domains (e.g., bank or restaurant).

In some cases, we were unable to identify a proper category for a model and thus we assigned the value unknown. We have also identified models which contain mock data or are clearly created just for testing purposes, in which case we assigned the category dummy. Figure 6 shows several examples of dummy models. Models ⓐ and ⓑ are clearly test models (we additionally tag this model with testing), whereas model ⓒ is a slice of the UML meta-model used to create experiments (we additionally tag this model with experiment). We also checked the corresponding repositories to find out additional evidence that confirm that these are dummy models. For instance, typically these type of models are stored in folders named *test*, *experiment*, etc.

During the annotation process we also assigned the label *tags*, which specifies keywords characterizing the model and frequently allows us to specialize the value of the category. In the Ecore models of the running example, *tags* may include safety, but the model in Fig. 1c also includes tags electronics and components since it is a special fault tree for physical systems. In UML, a model categorized as *computer-videogames* may include the tag poker to reflect the type of game.

For Ecore models, we included additional labels as we were able to perform a deeper analysis by exploring the original GitHub repository. The label *purpose* indicates the intended usage of the model (e.g., assignment, for models used in teaching; or benchmark, for models specifically created for benchmarking). The label *notation* specifies if there is an associated concrete syntax and the tool used to create it (e.g., xtext or sirius). Finally, the label *tool* indicates whether the meta-model is part of a tool. For the model in Fig. 1c, it has value CertWare since this meta-model is part of such tool.

In the case of UML models, we found that a model may have several diagrams but in the labelling process we focused on the ones that provided more information. Thus, we used the label *main-diagram* to indicate which diagrams were used to derive the category of the model.

Finally, we used the label *confidence* in both Ecore and UML models to indicate the confidence level of the coder when labelling a model. For instance, the label *confidence* with value low indicates that the coder thinks that there is a high probability of being wrong. If no *confidence* label is set, it is then implicitly considered high.

The labelling process was carried out by the second and third authors of the paper, who have more than 10 years of experience in modelling. An author worked with Ecore and the other with UML. While the labelling process was performed individually, several milestones were established during the process to discuss the usage of the labels and unify the labelling criteria. We are aware that this process may present threats to validity, as identifying the category of a model is a subjective task and only two participants were performing such a process. A wrong perception or misunderstanding on author's side may result in a mislabelling of the model. To address this threat, we applied a cross-validation of the outcome of the labelling process. For the cross-validation, authors met together and each one reviewed a random sample of size 25% of the models labelled by the other author. All disagreement cases were discussed between the authors to reach consensus. As a result of the validation process, the labels of a 3% and 5% of Ecore and UML models were modified, respectively.

4.3 Dataset description

The current version of ModelSet is composed of 5,466 Ecore and 5,120 UML labelled models, making a total of 10,586 models. We believe that the current size of the dataset is enough to validate the methodology and explore potential future applications, but there are still thousands of models which will be labelled in future versions of the dataset.

In total, 28,719 labels were used (15,288 and 13,431 labels in Ecore and UML, respectively), with an average number of labels per model of 2.71 (2.80 and 2.62 in Ecore and UML, respectively). Only a 12.13% and 0.94% of Ecore and UML models, respectively, were labelled with low or medium *confidence* values. The lower value for UML models was because labels were usually easier to identify due to UML models being domain models (e.g., it is typically easier to find out that a model represents a *hotels* domain than to find out that a meta-model represents a distributed system).

Table 2 shows the use of labels in ModelSet in Ecore and UML. The table shows the number of different values for the label (*#values* column) and the percentage of models annotated with such a label (*coverage* column). As expected, the label *category* has extensively been used and reaches a coverage of 100% for Ecore and UML models. On the other hand, the *tags* label has also been used extensively, but given

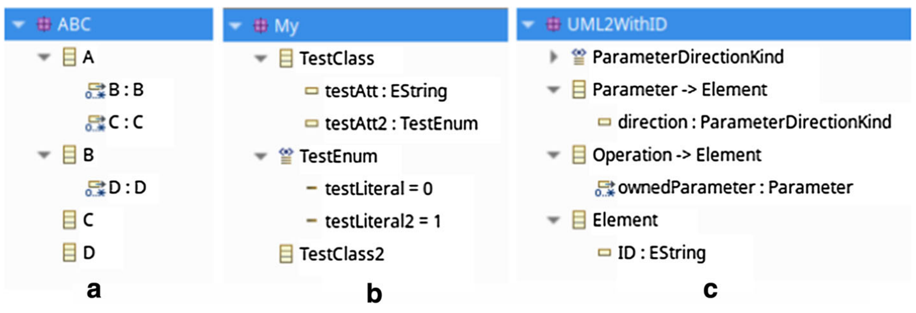


Fig. 6 Examples of dummy models

Table 2 Usage of labels in ModelSet

Label	Ecore		UML	
	# values	*coverage*	*# values*	*coverage*
Category	224	100.00%	135	100.00%
Tags	387	117.53%	92	52.99%
Purpose	29	31.16%	–	–
Notation	8	11.09%	–	–
Tool	188	7.78%	–	–
Main-diagram	–	–	6	108.03%

coverage > 100% when label is assigned more than once

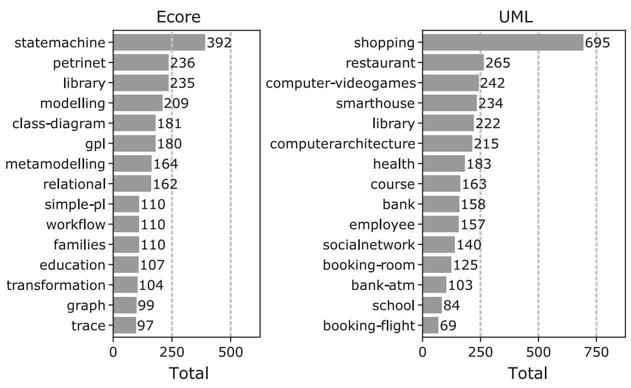


Fig. 7 Top 15 categories of Ecore and UML models in ModelSet, and total number of models per category

that there is more variability in Ecore than UML, the number of different values is greater in Ecore. Regarding the Ecore-specific labels, *purpose* was approximately assigned to 30% of the models, when it was possible to determine it. Labels *notation* and *tool* have a lower applicability since they are only available for some projects. For UML, all models have been labelled with *main-diagram* to specify the diagram/s used to infer the category.

Figure 7 shows the top 15 categories of Ecore and UML models. As can be seen, categories in Ecore models generally cover application domains related to conceptual modelling and DSLs (e.g., statemachine, petrinet or class-diagram). On the other hand, categories in UML models are generally related to business (e.g., shopping, restaurant or bank). It is also important to note that a number of Ecore and UML models are classified as dummy (13.34% and 11.84%, respectively), thus containing mock data or revealing the presence of models for testing purposes. Furthermore, 2.85% and 8.69% of Ecore and UML models, respectively, were categorized as unknown.

To provide some insights about the contents of the models in the dataset, we have computed several statistics about the number and type of model elements in Ecore and UML. The average model size is 205.84 and 143.57 elements in Ecore and UML models, respectively. Tables 3 and 4 shows a detailed analysis of each model element type. We distinguish between the normal and dummy models since the latter are a special category. In both Ecore and UML, the average amount of elements in normal models is always larger than in dummy models, as expected. Only in UML models, we observe a higher number of classes in dummy models; however, the ratio of properties per class reaches much lower (approximately 1 in dummy models *versus* 4.8 in normal models).

An easy way to explore the full report of ModelSet contents can be found at the companion website of the paper http://modelset.github.io.

4.4 Assessment

In this section, we highlight our experience in the application of GMFL. Altogether, we found it useful as it generally allowed us to easily discover similar models which might be difficult to find manually. In practice, we encountered three main situations, which we illustrate below.

4.4.1 Friends of my friends

This is one of the the main scenarios fostered by GMFL (illustrated in Fig. 3). Given a model m some similar models $m_1, \ldots, m_n$ are annotated with the same category. Then, additional models related to m_1 are annotated thanks to the model retrieval refinement part of the algorithm, and so on. We also observed that many times the models obtained by the retrieval refinement part start belonging to a related, but different category. For example, a labelling streak that started

Table 3 Composition of Ecore Models in ModelSet

Element Type	Normal (n=4737)		Dummy (n=729)	
	mean	*std. dev.*	*mean*	*std. dev.*
Elements	232.87	429.08	30.16	67.62
Packages	1.51	2.54	1.15	0.64
Classes	28.90	45.56	5.20	8.70
Enums	1.41	5.87	0.15	0.48
Datatypes	1.40	5.72	0.17	1.17
Attributes	18.25	35.44	3.35	13.39
References	30.14	53.94	4.09	10.59

with the category iot (i.e., arduino) evolved through robots, drones, logo and grafcet. In these cases it was natural to keep labelling with the new categories.

4.4.2 The gold mine

This is the other scenario boosted by GMFL. This situation refers to the identification of a large group of very similar models which can be annotated with the same labels with very little inspection work. The labelling task is therefore very fast and becomes an encouraging situation for the coder, who is motivated to keep on. We found that in the UML repository this happens very often, mainly because there are a lot of similar models (i.e., model clones) in specific categories. For instance, it was common to find large groups of models aimed at modeling ATM operations (e.g., enter pin, withdraw money, make a deposit, etc.).

4.4.3 No clue

It arises when the model under inspection is not known by the coder. In this case, some investigation about the nature of the model must be done. In the worst case, we had to label the model as unknown. This situation was generally easier to address in Ecore models, as we had the link to the GitHub repository to further explore the nature of the model. For instance, to label the model dmtc.ecore we visited the GitHub site, which explains that KlaperSuite is a tool for the analysis of component-based systems. The meta-model was “a kind of state machine”, but we were unsure. We googled “KlaperSuite dmtc” and we find a paper [16], from which we found that DMTC stands for “Discrete-Time Markov Chain” and we used it as a category. Moreover, using this information, we could fix some models that we had labelled incorrectly because they looked like a state machine.

Table 4 Composition of UML Models in ModelSet

Element Type	Normal (n=4514)		Dummy (n=606)	
	mean	*std. dev.*	*mean*	*std. dev.*
Elements	153.48	135.68	69.72	37.17
States	0.14	1.68	0.00	0.12
Transitions	0.20	2.41	0.00	0.04
Interactions	0.26	0.73	0.04	0.24
Activities	0.53	1.53	0.14	0.45
Components	0.25	0.78	0.09	0.56
Packages	1.29	1.13	1.13	0.59
Classes	5.64	6.92	12.00	8.84
Enums	0.28	0.76	0.01	0.11
Datatypes	0.75	2.86	0.31	1.76
Properties	27.11	26.20	12.47	8.05
Operations	8.56	16.98	4.42	5.99
Generalizations	2.20	4.38	1.83	2.27
Actors	1.52	2.71	0.40	1.29
UseCases	5.69	10.16	1.08	3.24
Associations	8.30	8.32	3.04	2.34

5 Case study: enhancing the MAR search engine

MAR is a search engine specifically designed for models [35][7], which allows users to locate relevant models by providing example-based queries. MAR currently indexes more than 500,000 models of different type, which makes their manual treatment infeasible. For instance, should we want to implement faceted search using categories it would be infeasible to manually annotate all the models. Therefore, an interesting direction is the use of ML techniques to train models which help us maintain and enhance MAR. In this section, we focus on how to implement three features using ModelSet:

- *Detecting dummy models* MAR blindly collects models from repositories like GitHub, GenMyModel, etc. Some of these models are too low quality to be useful in a general search. We have devised a method to detect them automatically.
- *Inferring categories for faceted search* Faceted search is a method to allow users to interactively search complex information spaces. The user is presented with controls to refine queries by means of tweaking facets [52]. An important facet that we are interested in is the main category of the model. We have trained a classification model to determine the category of a model.

[7] MAR is available at http://mar-search.org

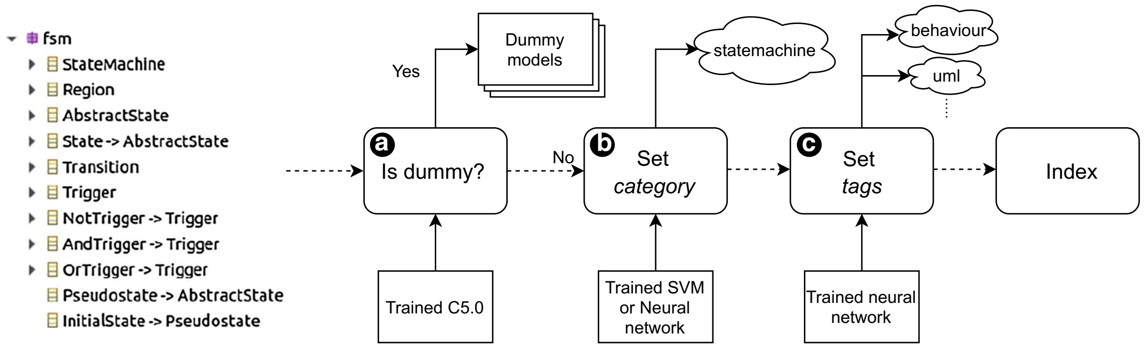


Fig. 8 Pipeline for applying trained ML models to MAR models. Dashed lines represent the actual route of the input model in the pipeline

- *Inferring tags* Some of the models indexed by MAR have tags attached which provide insights about what the model is about. For instance, topics obtained from GitHub repositories are tags used by MAR. However, most models in MAR do not have any tags, which is a shortcoming to easily inspect them. We have trained a ML model able to produce relevant tags for models.

Figure 8 shows how the trained ML models are applied to the models collected by MAR before being indexed. In this case, we focus on Ecore models. First, each input model is introduced into the dummy classifier (marker a). If it determines that the model is a dummy model, the artifact will be put in quarantine; otherwise, the model will be introduced as input in the classification model trained to infer the category of a model (marker b). Then, the model is introduced in the ML model trained to infer tags (marker c). Finally, the model is indexed in MAR, storing the meta-data provided by the ML models in order to show it to the user when needed.

Figure 9 illustrates how the trained ML models have been integrated into MAR. On the left, the user writes a query (marker a) which is used by MAR to perform a search. The results are listed on the right and include those models matching the query. Dummy models are discarded as MAR internally uses the trained ML classifier. For each model, MAR shows information like structural data (e.g., number of elements) and quality evidences (e.g., number of smells), which can be used to filter the models (see top part of results). Notably, in this work, we are interested on metadata that has been inferred using our trained ML models. The main category of the model (marker b) provides a simple guideline to categorize the results, and we also use it to allow user to filter the results (marker c). For instance, the main category of Ale.ecore is simple-pl. In addition, for each model in the results, there are several informative tags which represent its main topics (marker d). Some of these tags come from GitHub (e.g., many of the tags shown for Ale.ecore), but most models do not have tags attached originally, and so the tags that the user sees have been inferred using a trained ML model. For instance, for kermeta.ecore the inferred tags are imperative and classes.

The following sections describe how the ML models have been built.

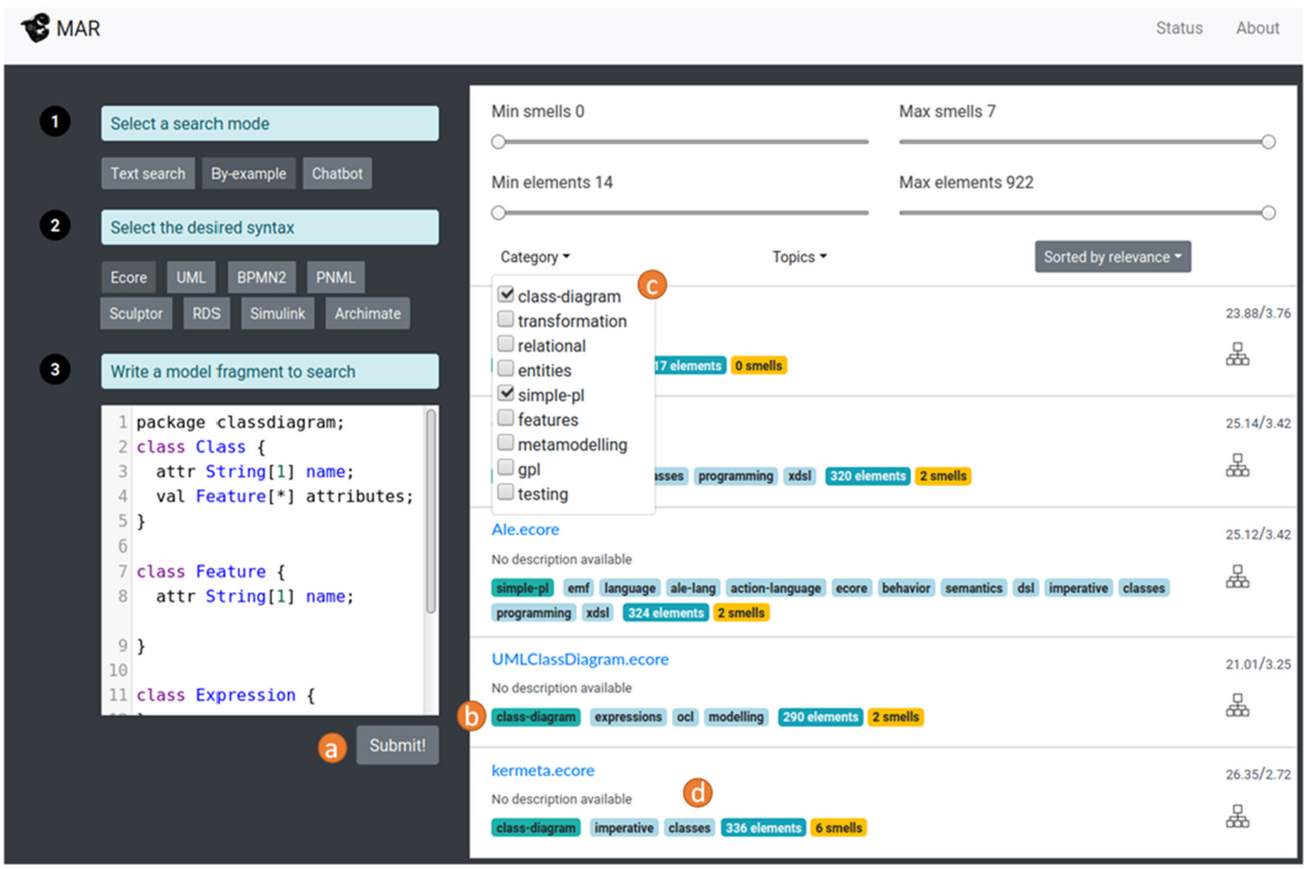
5.1 Detecting dummy models

The detection of dummy models helps us to automatically filter out those models whose purpose is not to represent in a faithful way some domain of interest, and thus they only add noise to the search results. In the following, we describe how we build a ML model that is used in MAR to detect dummy models and filter them out from the search results.

5.1.1 Extracting features of the models

To implement the task of detecting dummy models, we have considered the following input features for a classification model:

- *Counts of the number of elements*
  For each concrete meta-class, we count the number of times they appear in the model. For instance, in Ecore, we have *EClass*, *EAttribute* or *EReference* features, among others.
  We use these features because during the labelling process we noted that most of the dummy models are small, as commented in Sect. 4.3. For instance, for the model in Fig. 6a we would have *EPackage* = 1, *EClass* = 4, *EReference* = 3, *EAttribute* = 0, *EDataType* = 0.
- *Median of the number of characters in string attributes*
  We collect the string attributes of a model, and then we compute the median of their length. This feature is considered since many times the string attributes of the dummy models are short (e.g., abbreviations). For the model in Fig. 6a, we would have a median value of 1.
- *Count of dummy names* Given an string attribute, we consider it a dummy name if it contains the name of a meta-class or a significant part of it. For instance, in UML



```
package classdiagram;
class Class {
  attr String[1] name;
  val Feature[*] attributes;
}

class Feature {
  attr String[1] name;

}

class Expression {
```

☑ class-diagram
☐ transformation
☐ relational
☐ entities
☑ simple-pl
☐ features
☐ metamodelling
☐ gpl
☐ testing

17 elements 0 smells — 23.88/3.76

sses programming xdsl 320 elements 2 smells — 25.14/3.42

Ale.ecore — 25.12/3.42
No description available
simple-pl emf language ale-lang action-language ecore behavior semantics dsl imperative classes programming xdsl 324 elements 2 smells

UMLClassDiagram.ecore — 21.01/3.25
No description available
class-diagram expressions ocl modelling 290 elements 2 smells

kermeta.ecore — 26.35/2.72
No description available
class-diagram imperative classes 336 elements 6 smells

Fig. 9 Screenshot of MAR illustrating how the inferred metadata is integrated

a class whose name is *ClassA* is considered a dummy name. In Ecore we consider *TestClass* (see Fig. 6b) a dummy name as well.

Considering these features, we derive 16 features in ModelSet-Ecore and 195 from ModelSet-UML. This difference between the number of features is caused by the fact that the UML meta-model is larger than the Ecore meta-model.

5.1.2 Training and preparation phase

Once the feature extraction process is performed, we remove *non-English* models whose category is *unknown*. In Ecore, we have 5290 samples ($\sim$ 14% of the models belong to the *dummy* category and $\sim$ 86% of the models are not *dummy*). In UML, we have 4479 samples ($\sim$ 13% of the models belong to the *dummy* category and $\sim$ 87% of the models are not *dummy*).

Both datasets are split into the train set (80%) and test set (20%). Using the train set, we eliminate features whose variance is close to zero (in order not to consider non-informative features). As a result, for training we consider 9 in ModelSet-Ecore and 39 in ModelSet-UML.

We apply 10-fold cross-validation with three repetitions and with upsampling of the minoritary class in order to select the hyperparameters of the classifiers. We use upsampling to avoid issues caused by an unbalanced dataset (e.g., non-informative gradients in neural networks). The paired t-test is used to check whether there is a difference between the performance of two models. The considered hyperparameters and classifiers are summarized in Table 5, where $k-$NN stands for k-nearest neighbors, RF for Random Forest, NN is a neural network and C5.0 is an algorithm to create tree-based models and rule-based models.

5.1.3 Results

Since our main aim is to detect dummy models and ModelSet is unbalanced, we take F_1 as evaluation metric considering the class *dummy* as the positive class. This metric is the harmonic mean of the precision and recall. In this context, precision, recall and F_1 are defined as:

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Table 5 Models and hyperparameters considered

ML Model	Hyperparameters
k-NN	$k \in \{5, 7, 9, 11, 13\}$
Random Forest	Ecore$\rightarrow$ $m \in \{2, 3, 4, 5, 6, 7, 8, 9\}$
	UML$\rightarrow$ $m \in \{2, 6, 10, 14, 18, 22, 26, 30, 34, 39\}$
NN	units$\in$ {10, 20, 50, 100, 150}
(1 hidden layer)	
C5.0	model$\in$ {rule, tree}
	winnow$\in$ {F, T}
	trials$\in$ {1, 10, 20, 30, 40, 50, 60, 90}

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

where fp are the false positives, fn are the false negatives and tp are the true positives. F_1 will be used as evaluation metric to select the hyperparameters in the cross-validation.

The results of the cross-validation are presented in Table 6. In ModelSet-Ecore, C5.0 outperforms the Neural Network, $k-$NN and Random Forest (RF) in terms of F_1 and precision (with statitical diferences $p-$value< 0.05). In terms of recall, C5.0 is the worst. Despite that it achieves a high score (0.8136).

In ModelSet-UML, C5.0 outperforms the Neural Network, $k-$NN and RF in terms of F_1 and precision (with statistical differences $p-$value< 0.05). In terms of recall, C5.0 is not the best model but it achieves a high score (0.9255).

For this task, we are interested on ML models with high precision since we do not want to make the mistake of discarding models that are not dummy (that is, we want low false positives). In other words, it is preferable to show the user a dummy model than to fail to show a non-dummy model. Thus, according to the experiments we select the C5.0 model.

Finally, we evaluate the C5.0 model in the test set with the selected hyperparameters. Figure 10 shows the confusion matrix of C5.0 in the test set using ModelSet-Ecore. C5.0 achieves a precision of 0.8258, a recall of 0.7517 and a F_1 of 0.7870. Furthermore, it achieves an accuracy of 0.9442 which is greater than the proportion of non-dummy models in the test set (0.8628). On the other hand, Fig. 11 shows the confusion matrix of C5.0 in the test set using ModelSet-UML. The model achieves a precision, a recall and F_1 of 0.9573. The accuracy is 0.9888 which is higher than the proportion of non-dummy models in the test set (0.8693).

5.2 Model classification

Model classification is an important task to automate the organization of model repositories and enhance exploration facilities for end-users by allowing the automatic filtering of large collections of models. In our case study, we use model classification to help end-users navigate search results shown by MAR.

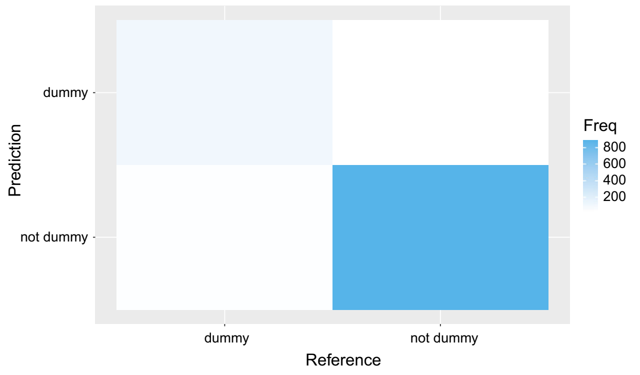


Fig. 10 Confusion matrix of the C5.0 in the test set using ModelSet-Ecore

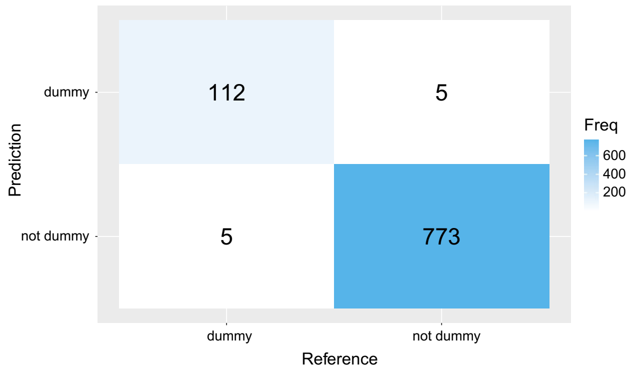


Fig. 11 Confusion matrix of the C5.0 in the test set using ModelSet-UML

We kick-start the construction of our model classifier by looking for other similar works in the literature. In [39] a feed-forward neural network with one hidden layer was trained to classify Ecore meta-models according to its category using a dataset of 555 meta-models [7]. To build our classifier, we propose to follow a similar approach and replicate the original experiment but using ModelSet, which also allows us to evaluate whether we obtain more accurate conclusions as our dataset is larger.

5.2.1 Training and preparation phase

We use the category of the models as the target variable of the classification problem. Three datasets are considered:

1. The dataset of 555 meta-models [7] used in [39] (referred to as Ecore-555)
2. The 5,466 Ecore models available in ModelSet referred to as ModelSet-Ecore.
3. The 5,120 UML models available in ModelSet referred to as ModelSet-UML.

Table 6 Results of the cross-validation in terms of F_1, precision and recall

	Model	Best parameters	Metrics		
			F_1	Precision	Recall
ModelSet-Ecore	$k-$NN	5	0.6637	0.5349	0.8771
	RF	4	0.8144	0.7881	0.8444
	NN	10	0.6832	0.5714	0.8536
	C5.0	rules, F, 80	0.8303	0.8498	0.8136
ModelSet-UML	$k-$NN	5	0.8369	0.7545	0.9411
	RF	10	0.9359	0.9485	0.9248
	NN	50	0.9240	0.9116	0.9382
	C5.0	rules, F, 30	0.9443	0.9653	0.9255

From these datasets, we consider categories with more than 7 elements to make the learning process as faithful as the process performed in [39] (i.e., the smallest category in Ecore-555 includes 7 meta-models). Thus, Ecore-555 is composed of 9 categories and 555 models, ModelSet-Ecore is composed of 80 categories and 4230 models and ModelSet-UML is composed of 50 categories and 3768 models.

The models of each dataset are split into train and test sets (with a distribution of 70%/30% for train/test). We apply 10-fold cross-validation (as it is done in [39]) in order to select the hyperparameters of the classifiers and the paired t-test to check whether there is a difference between the performance of two models. The classifiers that we train are: (i) linear SVM, (ii) a feed-forward neural network, and (iii) $k-$NN. The hyperparameters considered are $k \in \{1 \dots 10\}$ for $k-$NN (number of neighbors), $C \in \{0.01, 0.1, 1, 10, 100, 1000\}$ for SVM (C controls how much you want to avoid misclassifying each training example and behaves as a regularization parameter) and one hidden layer with {5, 10, 20, 50, 100, 150, 200} units for the neural network (size of the hidden layer). To select the hyperparameters, we consider *Accuracy* as the evaluation metric (number of correct predictions divided by the number of total cases).

Each software model is encoded as a vector of features computed using a TF-IDF, treating all string attributes as words. (This is the encoding technique used in [39].)

5.2.2 Results

The results of the cross-validation are shown in Fig. 12 for the considered datasets. In Table 7, for each dataset, we display the best hyperparameters and the average of the accuracy of the 10 folds. Once the cross-validation is done with the hyperparameter selection, we train the models using all the train set and then the test set is used to evaluate the models with the chosen hyperparameters. The last column of Table 7 shows the results of the test sets.

In Ecore-555 and Modelset-Ecore, SVM and the neural network model outperform $k-$NN ($p-$value<0.05). SVM outperforms the neural network ($p-$value<0.05) in Modelset-Ecore. However, in Ecore-555, there are no differences between the performance of the SVM and the neural network. In general, all models have better accuracy in the Ecore-555 dataset (in cross-validation and the test set). This is caused by the fact that the classification problem is more difficult in Modelset-Ecore (80 versus 9 different categories).

Table 7 Results of crossvalidation and test. NN stands for neural network and CV for cross-validation

	Model	Best parameter	Accuracy	
			CV	Test
	$k-$NN	5	0.9278	0.9401
Ecore-555	SVM	1	0.9512	0.9520
	NN	50	0.9563	0.9401
	$k-$NN	3	0.8639	0.8873
ModelSet-Ecore	SVM	10	0.9236	0.9393
	NN	100	0.9148	0.9369
	$k-$NN	1	0.9127	0.9168
ModelSet-UML	SVM	10	0.9343	0.9442
	NN	150	0.9355	0.9434

According to these results, we can say that this classification problem seems not to be difficult because a simple machine learning model ($k-$NN) gets good results in both datasets. The encoding technique used (TF-IDF) is a good choice for this task. Regarding UML, the cross-validation results show that SVM and the neural network model outperform $k-$NN ($p-$value$<$ 0.05). Furthermore, there are no differences between the performance of the SVM and the neural network.

If we compare the performance of the classifiers in ModelSet-UML and in ModelSet-Ecore (in cross-validation and the test set), we see that the performance is better in ModelSet-UML. This is caused by the fact that the classification problem in ModelSet-UML is easier than the classification problem in ModelSet-Ecore (50 categories in UML against 80 in Ecore) and because the UML models tend to be more similar.

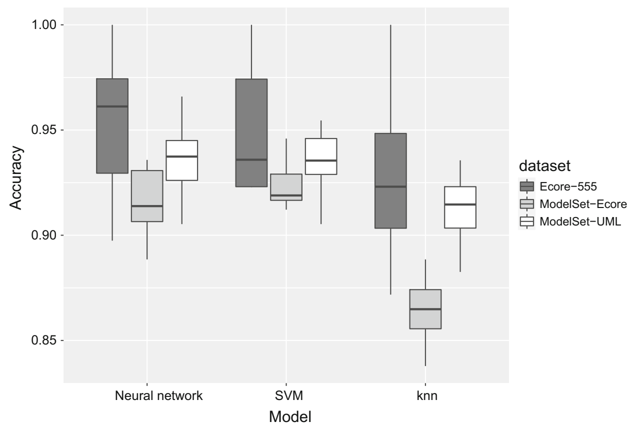


Fig. 12 Boxplots of the crossvalidation measures of the models with the best hyperparameters

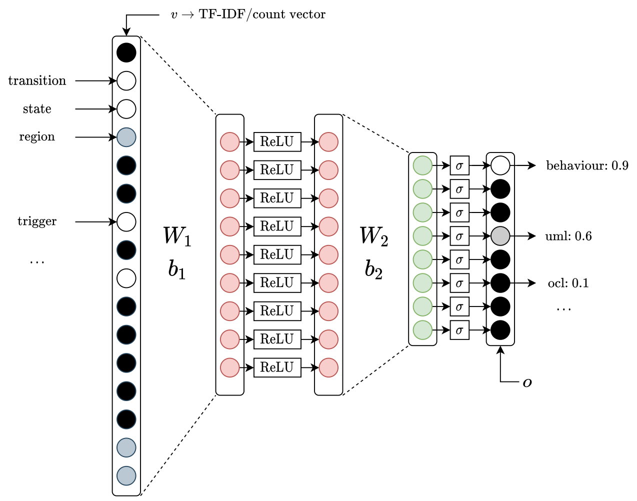


Fig. 13 Proposed neural model to infer tags

Finally, we have to remark that the scalability seems adequate in the sense that increasing the complexity of the problem (i.e., number of different categories) does not impact too much in the performance of the machine learning model.

As a concrete example of the usage of this ML model, Fig. 8 shows that if we introduce the Ecore model which contains concepts like StateMachine, Region, etc., the trained classifier predicts that this model belongs to category statemachine.

5.3 Inferring tags

Similarly to model classification based on categories, inferring tags is also an important task to provide insights about the contents of a model. Given a model, we want to infer its tags automatically based on the tags annotated in the dataset. This classification problem differs from inferring categories in the fact that a model can have more than one tag. Therefore, the problem of inferring tags is a multi-label and multi-class classification problem [51], which we describe in the following.

5.3.1 Neural model

To tackle this problem, we use a simple neural model that learns the weights of two layers: $W_1 \in \mathrm{R}^{1024 \times d_{in}}, b_1 \in \mathrm{R}^{1024}$, $W_2 \in \mathrm{R}^{d_{out} \times 1024}$ and $b_2 \in \mathrm{R}^{d_{out}}$. Given a model as input represented by its TF-IDF vector or its word count vector $v \in \mathrm{R}^{d_{in}}$, the neural model does the following:

$$o = \sigma\left(W_2 \mathrm{ReLU}\left(W_1 \cdot v + b_1\right) + b_2\right)$$

where ReLU and σ are the ReLU and sigmoid activations, respectively. d_{in} is the input dimension (number of words in the vocabulary which is computed by taking all unique string values in the dataset models) and d_{out} is the output dimension (number of tags). The coordinate i of the output vector represents the probability of m to belong to the tag i. Figure 13 shows the architecture of the proposed neural network. At inference time, if o_i > threshold, then the tag i will be assigned to the model m (by default we establish a $threshold = 0.5$). This neural model is trained using the binary cross entropy and Adam [30] as optimizer.

5.3.2 Training and preparation phase

From Modelset-Ecore and ModelSet-UML, we consider models that have at least one tag and whose language is English. As a result, Modelset-Ecore and ModelSet-UML will be composed by 3782 and 1867 models, respectively. Both datasets are split into three sets training (72%), test (20%) and validation (8%). For each dataset, the validation set is used to perform early stopping (i.e., we stop training when there is no improvement over the validation set) and the test set is used to evaluate the model.

5.3.3 Results

As evaluation metrics we use precision and recall that are measured using the test set. Precision is the fraction of output tags that are correctly inferred, recall is the fraction of correct tags that are successfully inferred and F_1 is the harmonic mean of precision and recall. Given the test set T, these metrics are computed using the following formulas [51]:

$$\text{Precision} = \frac{1}{|T|}\sum_{i=1}^{|T|}\text{Prec}_i,\ \text{ where } \text{Prec}_i = \frac{|Y_i \cap Z_i|}{|Z_i|}$$

$$\text{Recall} = \frac{1}{|T|}\sum_{i=1}^{|T|}\text{Rec}_i,\ \text{ where } \text{Rec}_i = \frac{|Y_i \cap Z_i|}{|Y_i|}$$

$$F_1 = \frac{1}{|T|}\sum_{i=1}^{|T|} F_{1,i}, \text{ where } F_{1,i} = 2 \cdot \frac{\text{Prec}_i \cdot \text{Rec}_i}{\text{Prec}_i + \text{Rec}}$$

where Z_i is the set of the inferred tags and Y_i is the set of real tags. For instance, let us assume that Fig. 8 shows a model belonging to the test set and that the trained neural network predicts two tags: *behavior* and *uml*. However, the actual tag of the input model is just *behavior*. Therefore, in this case, $\text{Prec}_i = 0.5$ (since *uml* does not belong to the real set of tags) and $\text{Rec}_i = 1$ (since we retrieve all actual tags, that is, *behavior*). Finally, computing the harmonic mean between Prec_i and Rec_i, we get $F_{1,i} = 0.67$.

After training, the network achieves a precision of 0.8791, a recall of 0.8593 and F_1 of 0.8621 using the test set in ModelSet-Ecore, which tell us that this model has a good performance. In particular, it has a high precision, thus implying that the inferred tags by the neural network are likely to be correct. On the other hand, in ModelSet-UML, the model achieves a precision of 0.9117, a recall of 0.9135 and F_1 of 0.9121.

6 Related work

The research line that studies the application of ML to software engineering has made a lot of progress in recent years [50,56]. To handle source code there are three types of ML models typically considered: code-generating models [4, 55], representational models of code [5,57] and pattern mining models [3]. These models have many applications such as recommender systems (e.g., code autocompletion [4]), inferring coding conventions [3], clone detection [57], code to text and text to code [34], etc. We believe that most of these ML models can be extended to be applied to modelling artifacts, but this requires the existence of large and high-quality datasets of software models.

Up to date the possibilities of applying ML to MDE have not been fully explored yet [13]. Therefore, with this work, we attempt to encourage researchers to use this first version ModelSet to adapt existing ML models to handle modelling artifacts. In the following we discuss works related to datasets of software models, as well as concrete applications already developed.

6.1 Datasets

The closest dataset to ours is [7] which contains 555 Ecore models labelled with its category. ModelSet provides more than 10,000 labelled models. The LindholmenDataset contains about 93,000 UML models [47]. An important shortcoming of this dataset is the difficulty of processing its models in practice, due to a number of reasons including: variety of formats and versions (e.g., EMF, StarUML, etc.), invalid models and models of poor quality (i.e., many models are just toy examples, other extracted from images, etc.). A curated dataset of 2420 meta-models is reported on [10]. Although the meta-models are not labelled, it comes with an analysis tool chain to facilitate experiments. A large dataset of OCL expressions is contributed in [37]. The goal is to enhance OCL-related research, which is demonstrated by replicating several studies about OCL with the dataset. A dataset of 8904 BPMN models is mined from Github in [25]. Its goal is to foster empirical research about business process models. The dataset is not labelled but the models have been validated for correctness. A dataset of APIs classified using Maven Central tags is used to analyze the effectiveness of hierarchical clustering[22]. Some of the identified tags for APIs may be used to enrich ours. Works about empirical studies on the usage of MDE artifacts have built ad-hoc datasets to perform specific analysis. In [32], a large number of MDE artifacts retrieved from GitHub are analysed. The analysis is performed by collecting information about specific file types at the commit level. The usage of EMF in Open Source projects hosted in GitHub is addressed in [23]. Similarly, the usage of graph query languages in Java projects is studied in [49], and the projects are classified according to its application domain.

6.2 Applications

There are several research lines in the application of AI and ML to address MDE problems. One direction is applying search-based algorithms, for instance to address co-evolution problems [29]. Another direction is reinforcement learning, which has been recently applied to address model repair [26]. These approaches do not need a dataset. We focus on those which require the support of a dataset.

The task of classifying UML class diagrams between manually created (for forward engineering) and reverse engineered is tackled in [41]. Several classification and features are tried, over a dataset of 999 UML models. AURORA [39] is a classifier of Ecore meta-models according to its category. It uses a TF-IDF approach to train a neural network with a dataset of 555 Ecore meta-models. We have replicated this work with our dataset, obtaining comparable results.

The application of clustering techniques to relatively large collections of models, in particular Ecore meta-models, has been researched in some depth [8,9,11]. However, these models were not made available and the results are barely replicable. Our dataset would improve the replicability of new experiments. Moreover, it could be used to analyze the effectiveness of existing approaches.

A recommender system for UML activities is presented in [33]. It is based on a predefined catalogue of possible suggestions. EXTREMO is a meta-model recommendation tool which recommends interesting terms based on flexible queries evaluated over Wordnet[38]. Similarly, DoMoRe uses semantic networks to aid in domain modelling tasks [1]. Kögel proposes the use of model history as a means to identify possible actions to be recommended [31]. Our dataset would be applicable to implement alternative approaches based on neural models. Additionally, the labels associated to the models in the dataset can be used to enhance the selection of models. For instance, when training a ML system (e.g., a recommender system [53]) it is typically advisable to use stratified sampling to make sure that the split of the models is balanced in terms of the categories of the dataset. In the same line, for model-set selection approaches [12], having a balanced set in terms of the labels can be used as a another criteria of the search process.

7 Conclusions

In this paper we have presented the initial version of ModelSet, a large labelled dataset of software models composed of 5,466 Ecore meta-models and 5,120 UML models. To speed up the labelling process, we have devised a novel labelling method, named GMFL, and created a supporting tool. ModelSet is freely available and we hope it becomes a milestone in development of ML applications for MDE. To this end, we have shown how ModelSet has been used in a case study to address the detection of dummy models and inference of categories and tags.

As future work we plan to continue enhancing ModelSet. We also want to create a web version of our tooling and apply it to use crowdsourcing and collaborative approaches for labelling new models. We believe that crowdsourcing and collaborative approaches would allow us to better address the subjectivity threat when labelling models, as several label proposals would help on identifying the most commonly accepted one for a model. Finally, we aim at analysing the dataset from other perspectives, like model quality, tool usage, etc.

Acknowledgements Work funded by a Ramón y Cajal 2017 Grant (RYC-2017-237) funded by MINECO (Spain) and co-funded by the European Social Fund. José Antonio Hernández López enjoys a FPU grant funded by the Universidad de Murcia.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availibility The database, the Eclipse plug-in, the ML models and scripts to replicate the results are available at https://figshare.com/s/5a6c02fa8ed20782935c. We also provide a live web-based interface to easily explore the dataset at http://modelset.github.io.

References

1. Agt-Rickauer, H.: supporting domain modeling with automated knowledge acquisition and modeling recommendations. Ph.D. thesis (2020)
2. Allamanis, M., Barr, E.T., Devanbu, P., Sutton, C.: A survey of machine learning for big code and naturalness. ACM Comput. Surv. **51**(4), 1–37 (2018)
3. Allamanis, M., Sutton, C.: Mining Idioms from Source Code. In: International symposium on foundations of software engineering, pp. 472–483 (2014)
4. Alon, U., Sadaka, R., Levy, O., Yahav, E.: Structural language models of code. In: International Conference on Machine Learning, PMLR, pp 245–256 (2020)
5. Alon, U., Zilberstein, M., Levy, O., Yahav, E.: Code2vec: learning distributed representations of code. ACM Program. Lang. **3**(POPL), 1–29 (2019)
6. Apache Software Foundation: Lucene. https://lucene.apache.org
7. Babur, Ö.: A labeled Ecore metamodel dataset for domain clustering. https://doi.org/10.5281/zenodo.2585456
8. Babur, Ö., Cleophas, L., van den Brand, M.: Hierarchical clustering of metamodels for comparative analysis and visualization. In: European conference on modelling foundations and applications, pp. 3–18 (2016)
9. Babur, Ö., Cleophas, L., van den Brand, M.: Metamodel clone detection with SAMOS. J. Comput. Lang. **51**, 57–74 (2019)
10. Barriga, A., Di Ruscio, D., Iovino, L., Nguyen, P.T., Pierantonio, A.: An extensible tool-chain for analyzing datasets of metamodels. In: Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings, pp. 1–8 (2020)
11. Basciani, F., Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Automated clustering of metamodel repositories. In: International conference on advanced information systems engineering, pp. 342–358 (2016)
12. Batot, E., Sahraoui, H.: A generic framework for model-set selection for the unification of testing and learning MDE tasks. In: Proceedings of the ACM/IEEE 19th international conference on model driven engineering languages and systems, pp. 374–384 (2016)
13. Cabot, J., Clarisó, R., Brambilla, M., Gérard, S.: Cognifying model-driven software engineering. In: International conference on software technologies: applications and foundations, pp. 154–160 (2017)
14. Cánovas Izquierdo, J., Cosentino, V., Cabot, J.: An empirical study on the maturity of the eclipse modeling ecosystem. In: International conference on model driven engineering, pp. 292–302 (2017)
15. Chen, Z., Kommrusch, S.J., Tufano, M., Pouchet, L.N., Poshyvanyk, D., Monperrus, M.: Sequencer: sequence-to-sequence learning for end-to-end program repair. IEEE Trans. Softw.

Engi. **47**(9), 1943–1959 (2021). https://doi.org/10.1109/TSE.2019.2940179

16. Ciancone, A., Drago, M.L., Filieri, A., Grassi, V., Koziolek, H., Mirandola, R.: The KlaperSuite framework for model-driven reliability analysis of component-based systems. Softw. Syst. Model. **13**(4), 1269–1290 (2014)
17. Clarisó, R., Cabot, J.: Applying graph kernels to model-driven engineering problems. In: International workshop on machine learning and software engineering in symbiosis, pp. 1–5 (2018)
18. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: a large-scale hierarchical image database. In: Conference on computer vision and pattern recognition, pp. 248–255 (2009)
19. Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Collaborative repositories in model-driven engineering. IEEE Softw. **32**(3), 28–34 (2015)
20. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. Kdd **96**, 226–231 (1996)
21. Giraldo, F.D., España, S., Pineda, M.A., Giraldo, W.J., Pastor, O.: Conciliating model-driven engineering with technical debt using a quality framework. In: Information systems engineering in complex environments: CAiSE forum, LNCS, vol. 204, pp. 199–214 (2014)
22. Härtel, J., Aksu, H., Lämmel, R.: Classification of APIs by hierarchical clustering. In: International Conference on Program Comprehension (ICPC), pp. 233–23310 (2018)
23. Härtel, J., Heinz, M., Lämmel, R.: EMF patterns of usage on GitHub. In: European conference on modelling foundations and applications, pp. 216–234. Springer (2018)
24. Heijstek, W., Chaudron, M.R.V.: Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process. In: Euromicro conference on software engineering and advanced applications, pp. 113–120 (2009)
25. Heinze, T.S., Stefanko, V., Amme, W.: Mining BPMN Processes on GitHub for tool validation and development. In: Nurcan, S., Reinhartz-Berger, I., Soffer, P., Zdravkovic, J. (eds) Enterprise. Business-Process and Information Systems Modeling, pp. 193–208. Springer International Publishing, Cham (2020)
26. Iovino, L., Barriga, A., Rutle, A., Heldal, R.: Model repair with quality-based reinforcement learning. J. Object Technol. https://doi.org/10.5381/jot.2020.19.2.a17
27. Izurieta, C., Rojas, G., Griffith, I.: Preemptive management of model driven technical debt for improving software quality. In: International conference on quality of software architectures, pp. 31–36 (2015)
28. Just, R., Jalali, D., Ernst, M.D.: Defects4J: a database of existing faults to enable controlled testing studies for java programs. In: International symposium on software testing and analysis, pp. 437–440 (2014)
29. Kessentini, W., Sahraoui, H., Wimmer, M.: Automated metamodel/model co-evolution: a search-based approach. Inf. Softw. Technol. **106**, 49–67 (2019)
30. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
31. Kögel, S.: Recommender system for model driven software development. In: Joint meeting on foundations of software engineering, pp. 1026–1029 (2017)
32. Kolovos, D.S., Matragkas, N.D., Korkontzelos, I., Ananiadou, S., Paige, R.F.: Assessing the use of eclipse MDE technologies in open-source software projects. In: OSS4MDE@ MoDELS, pp. 20–29 (2015)
33. Kuschke, T., Mäder, P., Rempel, P.: Recommending auto-completions for software modeling activities. In: International conference on model driven engineering languages and systems, pp. 170–186 (2013)
34. LeClair, A., Jiang, S., McMillan, C.: A neural model for generating natural language summaries of program subroutines. In: International conference on software engineering, pp. 795–806 (2019)
35. López, J.A.H., Cuadrado, J.S.: Mar: a structure-based search engine for models. In: Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems, pp. 57–67 (2020)
36. Martinez, M., Durieux, T., Sommerard, R., Xuan, J., Monperrus, M.: Automatic repair of real bugs in java: a large-scale experiment on the Defects4j dataset. Emp. Softw. Eng. **22**(4), 1936–1964 (2017)
37. Mengerink, J.G., Noten, J., Serebrenik, A.: Empowering OCL research: a large-scale corpus of open-source data from GitHub. Emp. Softw. Eng. **24**(3), 1574–1609 (2019)
38. Mora Segura, Á., Pescador, A., de Lara, J., Wimmer, M.: An extensible meta-modelling assistant. In: International conference on enterprise distributed object computing, pp. 1–10 (2016)
39. Nguyen, P.T., Di Rocco, J., Di Ruscio, D., Pierantonio, A., Iovino, L.: Automated classification of metamodel repositories: a machine learning approach. In: International conference on model driven engineering languages and systems, pp. 272–282 (2019)
40. OMG: OMG Unified Modeling Language (OMG UML), Version 2.5.1 (2017). http://www.omg.org/spec/UML/2.5.1
41. Osman, M.H., Ho-Quang, T., Chaudron, M.: An automated approach for classifying reverse-engineered and forward-engineered UML class diagrams. In: Euromicro conference on software engineering and advanced applications, pp. 396–399 (2018)
42. Pérez-Soler, S., Daniel, G., Cabot, J., Guerra, E., de Lara, J.: Towards automating the synthesis of chatbots for conversational model query. In: International conference on enterprise, business-process and information systems modeling, pp. 257–265 (2020)
43. Pérez-Soler, S., González-Jiménez, M., Guerra, E., de Lara, J.: Towards conversational syntax for domain-specific languages using chatbots. J. Object Technol. **18**(2), 5-1 (2019)
44. Project, T.E.: Eclipse modeling framework (2020). http://www.eclipse.org/emf
45. Rajpurkar, P., Zhang, J., Lopyrev, K., Liang, P.: SQuAD: 100,000+ questions for machine comprehension of text. In: Conference on empirical methods in natural language processing, pp. 2383–2392 (2016)
46. Rios, E., Bozheva, T., Bediaga, A., Guilloreau, N.: MDD maturity model: a roadmap for introducing model-driven development. In: European conference on model driven architecture-foundations and applications, Lecture Notes in Computer Science, vol. 4066, pp. 78–89 (2006)
47. Robles, G., Ho-Quang, T., Hebig, R., Chaudron, M.R., Fernandez, M.A.: An extensive of UML models in GitHub. In: International conference on Mining Software Repositories (MSR), pp. 519–522 (2017)
48. Roh, Y., Heo, G., Whang, S.E.: A survey on data collection for machine learning: a big data-AI integration perspective. IEEE Trans. Knowl. Data Engi. **33**(4), 1328–1347 (2021). https://doi.org/10.1109/TKDE.2019.2946162
49. Seifer, P., Härtel, J., Leinberger, M., Lämmel, R., Staab, S.: Empirical study on the usage of graph query languages in open source java projects. In: Proceedings of the 12th ACM SIGPLAN international conference on software language engineering, pp. 152–166 (2019)
50. Shafiq, S., Mashkoor, A., Mayr-Dorn, C., Egyed, A.: Machine learning for software engineering: a systematic mapping. arXiv preprint arXiv:2005.13299 (2020)
51. Tsoumakas, G., Katakis, I.: Multi-label classification: an overview. Int. J. Data Warehous. Min. (IJDWM) **3**(3), 1–13 (2007)
52. Tunkelang, D.: Faceted search. Synth. Lect. Inf. Concepts Retr. Serv. **1**(1), 1–80 (2009)

53. Weyssow, M., Sahraoui, H., Syriani, E.: Recommending metamodel concepts during modeling activities with pre-trained language models. arXiv preprint arXiv:2104.01642 (2021)
54. Whoosh. https://whoosh.readthedocs.io/en/latest/
55. Yin, P., Neubig, G.: A syntactic neural model for general-purpose code generation. arXiv preprint arXiv:1704.01696 (2017)
56. Zhang, D., Tsai, J.J.: Machine learning and software engineering. Softw. Qual. **11**(2), 87–119 (2003)
57. Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., Liu, X.: A novel neural source code representation based on abstract syntax tree. In: International conference on software engineering, pp. 783–794 (2019)

José Antonio Hernández López is a predoctoral researcher at the University of Murcia. He obtained a BSc in Mathematics, a BSc in Computer Science and a MSc in Big Data from the University of Murcia. Currently, he is enrolled in a PhD program in Computer Science supervised by Jesús Sánchez Cuadrado. His research interests include model-driven engineering (MDE), recommender systems and machine learning for software engineering.

Javier Luis Cánovas Izquierdo is an associate professor at the IT, Multimedia and Telecommunications Department of Universitat Oberta de Catalunya (UOC). He is also a member of the SOM Research Lab within the Internet Interdisciplinary Institute (IN3-UOC). His research interests fall into the areas of software engineering, web engineering and socio-technical analysis of software systems. You can contact him at jcanovasi@uoc.edu or visit https://jlcanovas.es/.

Jesús Sánchez Cuadrado is a Ramón y Cajal researcher at the Languages and Systems Department of the University of Murcia. His research is focused on model-driven engineering (MDE) topics, notably model transformation languages, meta-modelling and domain specific languages. Lately, he has been involved in the construction of the MAR search engine (http://mar-search.org). On these topics, he has published several articles in journals and peer-reviewed conferences, and developed several open source tools. His e-mail address is jesusc@um.es and his Web page is http://sanchezcuadrado.es.