Evaluation of a machine learning classifier for metamodels

Phuong T. Nguyen¹ · Juri Di Rocco¹ · Ludovico Iovino² · Davide Di Ruscio¹ · Alfonso Pierantonio¹

Received: 1 May 2020 / Revised: 3 March 2021 / Accepted: 9 July 2021 / Published online: 9 September 2021
© The Author(s) 2021, corrected publication October 21, 2021

Abstract

Modeling is a ubiquitous activity in the process of software development. In recent years, such an activity has reached a high degree of intricacy, guided by the heterogeneity of the components, data sources, and tasks. The democratized use of models has led to the necessity for suitable machinery for mining modeling repositories. Among others, the classification of metamodels into independent categories facilitates personalized searches by boosting the visibility of metamodels. Nevertheless, the manual classification of metamodels is not only a tedious but also an error-prone task. According to our observation, misclassification is the norm which leads to a reduction in reachability as well as reusability of metamodels. Handling such complexity requires suitable tooling to leverage raw data into practical knowledge that can help modelers with their daily tasks. In our previous work, we proposed AURORA as a machine learning classifier for metamodel repositories. In this paper, we present a thorough evaluation of the system by taking into consideration different settings as well as evaluation metrics. More importantly, we improve the original AURORA tool by changing its internal design. Experimental results demonstrate that the proposed amendment is beneficial to the classification of metamodels. We also compared our approach with two baseline algorithms, namely gradient boosted decision tree and support vector machines. Eventually, we see that AURORA outperforms the baselines with respect to various quality metrics.

Keywords Model-driven engineering · Machine learning · Neural networks · GBDT · SVM

1 Introduction

The all-important goal of delivering better software systems at lower cost has led to a crucial quest for ways to leverage abstraction conveniently. In software development, models are the primary abstractions to understand, communicate, and analyze software-intensive systems [79]. Model-driven engineering [71] (MDE) research has gained traction over

the years by shifting the focus of development from coding to modeling. With the emerging trend of globalization, modeling repositories [28,38,42,43] are useful platforms to locate artifacts for reuse [39]. In recent years, there is an increasingly large amount of metamodels uploaded to open repositories. Such repositories contain a valuable knowledge base that offers the wisdom of the crowd. In this context, engineers can benefit or learn from what the others have done by searching over the repositories. Nevertheless, locating relevant information in a repository induces technical challenges as the related artifacts need to be appropriately classified, which has been done manually even until recently. Such a task is both time-consuming and prone to error that may lead to accuracy issues.

We aim at giving support to modelers/developers in their metamodeling tasks. For the sake of presentation, we consider a developer who wants to create new metamodels for *database* domain, even though this can be generalized to any domain in practice. Then, a category consisting of metamodels for working with databases would come in handy. It helps the developer quickly approach the most suitable ones by narrowing down the search scope by considering only

Communicated by Tao Yue, Man Zhang, and Silvia Abrahao.

✉ Davide Di Ruscio
davide.diruscio@univaq.it
Phuong T. Nguyen
phuong.nguyen@univaq.it
Juri Di Rocco
juri.dirocco@univaq.it
Ludovico Iovino
ludovico.iovino@gssi.it
Alfonso Pierantonio
alfonso.pierantonio@univaq.it

¹ Università degli studi dell'Aquila, L'Aquila, Italy

² Gran Sasso Science Institute, L'Aquila, Italy

metamodels in the same category. Given such a category, the developer might browse and review those metamodels that best fit her needs. Starting or getting inspired by existing metamodels helps tame the complexity of developing new artifacts by reducing the development time and increasing efficiency.

The manual annotation of metamodels is typically tedious and usually inapplicable to a large amount of data. In this sense, it is essential to have convenient machinery to automate such a process. Clustering techniques have been exploited to group similar metamodels [10]. Nevertheless, the performance of clustering techniques heavily relies on the chosen similarity measure. In other words, clustering is considered to suffer from two main disadvantages: (i) timing performance and (ii) identifying the appropriate number of clusters [58].

Machine learning (ML) has gained a profound momentum in recent years [35], thereby fostering a plethora of applications and yielding a superior performance compared to that of conventional approaches. An ML algorithm attempts to acquire real-world knowledge by autonomously simulating human's learning activities [66]. ML systems can automatically extract meaningful patterns from training data. Given the circumstances, we anticipate a high potential of ML in the MDE context. Interestingly, though learning algorithms have been applied to solve different issues, such as refactoring rules [54], or domain modeling [36], still the presence of machine learning in MDE is not commensurate with its potential.

In our previous work [58], we designed and implemented AURORA, an automated supervised classifier for metamodel Repositories based on a feed-forward neuRAI network. Our work is among the first attempts to automate the classification of metamodel repositories. Metamodels are considered data points in our approach, and they are classified using supervised learning techniques. In this way, we overcome the main limitation of different unsupervised clustering techniques where it is necessary to specify the number of clusters in advance. With supervised learning, the number of clusters depends on the training data, and it has been specified ex ante in the labeled data. In our experiment, we used a dataset with nine categories; however, such a number can easily be changed if there are more categories previously classified. In this respect, there is no need to dictate any number in advance with our approach. By experimenting on the dataset of manually classified metamodels using the tenfold cross-validation methodology, we obtained an encouraging performance in terms of success rate, precision, recall, and F₁ score.

This paper further extends our work by conducting a comprehensive evaluation on different machine learning classifiers for metamodels. First, we investigate the mathematical background and propose an amendment in the internal design of AURORA, with the final aim of making it

more effective. Second, we study AURORA's performance in detail, investigating different configurations by means of a thorough evaluation. Third, since there exist several supervised learning techniques and they have been widely used for different purposes, in the evaluation we compare AURORA with two well-founded classification algorithms, namely gradient boosted decision tree [31] (GBDT) and support vector machines [74] (SVM). Both techniques have been exploited in a wide range of applications, such as pattern classification [22], face recognition [65], or recommender systems [40], to name a few. However, to the best of our knowledge, they have never been utilized to classify metamodels. This motivates us to perform an investigation on the practicality of GBDT and SVM in metamodel classification.

In summary, our paper makes the following contributions:

- implementation of a supervised classifier for categorizing metamodels based on a neural network;
- investigation of various factors that affect AURORA's classification performance;
- an experimental validation on a dataset of metamodels that has been collected from GitHub and manually categorized afterward;
- comparison of AURORA with gradient boosted decision tree and support vector machines;
- we published the conceived tool together with the parsed metadata to facilitate future research.¹

Paper's structure We organize the paper into the following sections. The background and motivations of this work are introduced in Sect. 2. The supervised classifier for metamodels is detailed in Sect. 3. The evaluation is presented in Sect. 4, and the main results are analyzed in Sect. 5. Afterward, Sect. 6 presents some related discussions. The threats to validity of our findings are discussed in Sect. 7. We review the related work and conclude the paper in Sects. 8 and 9, respectively.

2 Motivations and background

As a base for further presentation, Sect. 2.1 briefly describes metamodels, while Sect. 2.2 reports a motivating example to clarify the usefulness of classifying metamodels. Afterward, in Sect. 2.3, we provide a background on clustering and classification of metamodels. Finally, Sect. 2.4 emphasizes the need to find a suitable grouping technique.

¹ <https://github.com/MDEGroup/AURORA/>

Fig. 1 An excerpt of the *ecore* metamodel

2.1 A brief introduction to metamodels

A metamodel specifies the abstract concepts of a domain where concepts and the relationships among them are expressed by modeling infrastructure [72]. As an example, we depict in Fig. 1 a simplified representation of the *Ecore* metamodel. The Eclipse Modeling Framework (EMF)² allows modelers to design, define, and edit models. EMF contains *Ecore* as a meta-metamodel for describing metamodels. All the represented metaclasses are named elements. *EPackage* is composed of sub-packages as well as classes, while *EClass* is made of *EStructuralFeatures*, i.e., *EAttributes* and *EReferences*. Moreover, *EClass* can inherit structural features from other classes. *EAttribute* and *EReference* inherit the *lowerbound*, and *upperbound* attributes in order to define the cardinalities of structural features. *EReference* links the container to *EClass* by *eReferenceType* reference. Moreover, containment is a Boolean attribute, which allows one to specify if the linked *EClass* is contained, e.g., to trigger cascade deletion, or not in the container being modeled. *EAttributes* are typed as *EDataType* instance by *eAttributeType* reference.

2.2 Motivating example

Model repositories offer a valuable resource for users who are interested in acquiring knowledge from already developed modeling artifacts. In recent years, the adoption of repositories is increasingly becoming commonplace in different application domains including power grids [63], biology, e.g., BioModels Database,³ CellML⁴; and software development, e.g., GitHub,⁵ SourceForge⁶; or MDE development, e.g., ReMoDD [29], ATL Zoos,⁷ MDEForge [11], to name

a few. Since the searchability and discoverability of metamodels strongly impact on the reuse of repositories, such platforms have to provide domain experts with advanced searching algorithms and facilities, which are suitable to manage large datasets and to enable quick identification of stored items [63]. To give an intuition about the possible extension of a publicly available dataset of models, we refer to GenMyModel,⁸ an online modeling platform that offers a predefined set of modeling formalisms, e.g., UML, BPMN, Flowchart. Currently, the platform hosts more than one million of models, and it has more than 900,000 registered users. Concerning duplicates in modeling repositories, a recent work [50] reveals that there is a substantial amount of clones. In particular, through an empirical study to identify duplicate code files on 4.5 million of GITHUB repositories, including those for metamodels, the authors show that more than 40% of the files are duplicated in the analyzed repositories. In contrast, some specific platforms impose mechanisms to filter duplicates. For instance, using a hash function, MDEForge [11] checks for clones before adding new metamodels, and this considerably reduces the number of duplicates in the platform. Altogether, this implies that the level of duplications may vary from platform to platform, depending on their purpose.

By focusing on the management of metamodel repositories, the grouping of the stored metamodels into independent categories facilitates future lookup as well as maintenance [8]. In particular, ReMoDD allows repository contributors to tag an artifact with predefined sets of labels, for example, artifact type, keywords, or System/Software Domains. Meanwhile in ATL Zoos, repository maintainers can assign a weak domain classification to metamodels.

However, manual classification is not only time-consuming but also susceptible to error. A misspecification of metadata, such as artifact description, labels, or domains, may substantially compromise the reachability and thus potential reuse of a metamodel. We take an example as follows: Fig. 2 depicts the metadata stored into the ATL Ecore Zoo repository for four different metamodels, i.e., *xasm*, *AsmL*, *UMLDI-StateMachines*, and *Sync Charts*. By a careful inspection, we realized that all the metamodels share many elements in common, and they are related to the *State Machine* domain. In particular, as shown in Figs. 3 and 4, there are pairwise similarities between *xasm* and *AsmL*, or between *UMLDI-StateMachines* and *SyncCharts*. From the figures, it is evident that the metamodels should be classified in the same category, because of the large commonalities among them. However, in fact three of them have been classified by the modelers of ATL Ecore Zoo into three independent domains, while *xasm* is left unclassified. Through this example, we see that the wrong categories deteriorate

² <https://www.eclipse.org/modeling/emf/>

³ <http://www.ebi.ac.uk/biomodels-main/>

⁴ <http://models.cellml.org/cellml>

⁵ <https://github.com>

⁶ <https://sourceforge.net/>

⁷ <https://bit.ly/39Re0HG>

⁸ <https://www.genmymodel.com/>

XASM 1.15	AsmL 1.0	UMLDI State Machine 0.1	Sync Charts
date : 2006/05/1	date : 2006/04/12	date : 2006/02/21	date : 2006/07/10
Domain :	Domain : Microsoft DotNET	Domain : UML	Domain : Sync Chart
Description : This is a metamodel for XASM (eXtensible Abstract State Machine) an Abstract State Machines (ASMs) based language	Description : Abstract State Machine Language	Description : This is a simplified metamodel of UML State Machines diagram according to the UML Diagram Interchange standard. The source metamodel of UML is well described by the OMG. It has been voluntarily simplified to be used more easily.	Description : This metamodel fragment describes SyncCharts See : This metamodel fragment has been extracted from information available in the following article : Article Integrating the Synchronous Paradigm into UML : Application to Control-Dominated Systems by Charles André, Marie-Agnès Peraldi-Frati, Jean-paul Rigault Page 173, Section 3.3, Fig. 2. A UML meta-model for SyncCharts Lecture Notes in Computer Science 2460 Jean Marc Jézéquel, Heinrich Hussman, Stephen Cook UML 2002 - The Unified Modeling Language

Fig. 2 ATL zoo metadata for four metamodels

Fig. 3 Commonalities between xasm and AsmL

the reachability as well as the possibility of the metamodels of being reused.

2.3 Clustering and classification of metamodels

The motivating example in Sect. 2.2 highlights an urgent need to find suitable machinery to automate the process of grouping metamodel repositories. In other words, advanced techniques and tools to automatically organize modeling artifacts are needed [7], especially when metamodels are added or changed frequently, e.g., in case of cloud-based⁹ settings

[11]. Clustering and classification are some of the critical operations that are used to dig deep into available data for gaining knowledge and for identifying repetitive patterns. Both classification and clustering have been widely used in different fields, e.g., medicine, biology, physics, geology, to name a few. Clustering is an *unsupervised learning* technique, which aims at distributing a set of objects in different groups according to some similarity function [61]. Thus, the number of classes is not known ex ante, as in the case of supervised learning. In software engineering, such techniques have been employed, e.g., in reverse engineering and software maintenance for categorizing software artifacts [47]. In the context of open-source software, categorizing

⁹ <https://www.eclipse.org/che/>

Fig. 4 Commonalities between UMLDI-StateMachines and SyncCharts

available projects by understanding their similarities allows for reusing source code or learning how existing and similarly complex systems have been developed, thereby improving software quality. Meanwhile, classification is a *supervised learning* technique, which relies on the existence of predefined classes of objects and aims at understanding which class a new item belongs to [45]. In this respect, labeled training data are a prerequisite for classification, as it is needed to guide the learning process.

The possibility of applying clustering techniques has been investigated to organize reusable metamodels automatically, aiming to solve the daily task in MDEForge [11]. As an example, Fig. 5 depicts the resulting dendrogram by clustering a real set of metamodels [64] (whose characteristics are going to be introduced in detail in Sect. 4.2). In particular, there are nine categories of metamodels and they are grouped using the clustering technique presented in our previous work [10]. Modelers are provided with structured overviews of available metamodels, which instead are typically shown as mere lists of stored elements. Metamodels are automatically organized in views consisting of connected graphs. Each graph represents identified clusters, where nodes correspond to metamodels and the similarity among nodes is represented as the thickness of the edges.

2.4 On the need for a suitable grouping technique

Since both classification and clustering are suitable for our purpose, i.e., automatic grouping of metamodel repositories, we need to select the most feasible method to be deployed in practice. Despite the fact that unsupervised categorization obtains encouraging results, as well as it has been exploited as the main engine to group metamodels in MDEForge [10], we realized that there are two main issues that obstruct the adoption of the hierarchical clustering technique to categorize metamodels in the long term, as discussed below:

Fig. 5 Dendrogram obtained by hierarchical clustering algorithm (a picture with higher resolution is available online at: <https://bit.ly/3pSkRG5>)

- *Usability*: Our previously conceived mechanism can automatically group metamodels according to their content and structure, relying on a similarity function [10]. We applied the hierarchical agglomerative clustering by shifting the dendrogram cutoff from 0.1 to 1.0 with 0.05 as the step and 0.25 as the dendrogram cutoff value. Given such settings, the number of clusters varies enormously (cf. Fig. 5). Thus, the repository administrator, i.e., the person who configures the similarity function, has to specify a concrete number of clusters. Moreover, since various similarity functions are available, the selection of such functions might have a profound effect on the final clusters. In this context, the administrator may face some difficulties in grasping the application domains formalized by the available metamodels and getting a bird's eye view of them. In this sense, we can see that usability of the clustering technique is substantially compromised;
- *Efficiency*: Some clustering techniques suffer from a low efficiency. For instance, on the Ecore Zoo dataset consisting of around 300 metamodels, the clustering takes around four hours to finish on top of an ordinary computer [10]. Moreover, it is worth noting that every time

Fig. 6 Neuron and network **(a)** A simple perceptron with 3 inputs**(b)** A three-layer neural network

a new metamodel needs to be categorized, and the same procedure must be replicated on all the existing metamodels. Such a categorization activity turns to be not only a time-consuming but also an inefficient process.

On the quest for a mechanism that transcends the limitations of hierarchical clustering and automates the grouping process by exploiting the available data, we came up with a supervised learning technique, using a neural network to learn from existing labeled data and automatically classify unknown items afterward. The AURORA tool [58] has been conceived as the first effort to realize a proper classification tool for MDEForge. In the next section, we provide more details on its design and implementation.

3 A neural network to classify metamodel repositories

When developers create or update a repository, they specify one or more categories to the contained metamodels, and afterward, the categories serve as a means to help other developers narrow down the search scope and efficiently approach the repository. We hypothesize that the perception of humans toward the relationship among metamodels and categories is encoded in the manually classified categories. In this respect, we attempt to simulate human's cognition toward the metamodels–categories relationship, exploiting the manually categorized data. To this end, the features of a metamodel such as classes, attributes, and the corresponding label are used as input to tune the internal parameters, so as to eventually assign the metamodel to a specific category.

Supervised learning techniques attempt to mimic the learning activities from humans by generalizing through labeled data and deducing conclusions for new incoming data. The ability to learn from data allows neural networks to have a wide range of applications, e.g., pattern recognitions [14] or forecasting [80]. We designed and implemented AURORA [58] based on a neural network, which learns

from a training set consisting of labeled metamodels and effectively classifies incoming unlabeled ones. As a base for further presentations, Sect. 3.1 systematically explains the mathematical background behind the training process of a feed-forward neural network [62]. Afterward, Sect. 3.2 introduces the proposed architecture to realize the automated classifier. In Sect. 3.3, we briefly introduce gradient boosted decision tree and support vector machines as two baseline algorithms for our approach.

3.1 Training and testing

The atomic element of a neural network is called *perceptron*. Each perceptron receives a set of inputs and produces an output. Figure 6a depicts a simplified perceptron with three inputs. For each input x_i , there is a weight vector $\omega_i = \{\omega_{i1}^{(1)}, \omega_{i2}^{(1)}, \omega_{i3}^{(1)}\}$ associated with it. A bias $b_1^{(1)}$ is attached to increase the flexibility in adjusting the output.

Considering Fig. 6a, the output of the perceptron is $f(\sum_{i=1}^3 \omega_{i1}^{(1)} \cdot x_i + b_1^{(1)})$, where f is the *activation function* which maps a set of inputs onto the final output. There are various activation functions, and one of the most popular ones is ReLU (rectified linear unit) [1], which returns 0 given a negative input, and returns the input itself if it is larger than 0, i.e., $f(x) = \max(0, x)$. In this sense, ReLU is computationally efficient as it performs only a simple operation. However, it also suffers from a setback as follows: If too many activations are smaller than zero, then accordingly the output neurons will be rounded as zero. Given the circumstances, the network cannot produce correct predictions. We anticipate that the application of ReLU, or other activation functions, would be more beneficial to the final classification given a large amount of training data. Thus, we plan to consider ReLU and other activations in our future study. In this work, we make use of the *sigmoid function* since it is simple, yet effective. More importantly, the function has been widely exploited in machine learning studies [62].

A feed-forward neural network is made of several connected layers of neurons, and the output of one layer is fed

as input for the next layer's neurons, except for the output layer. The edges of the network convey information in a single direction [68], e.g., from left to right. Depending on the purpose, the number of neurons in a hidden layer, as well as the number of hidden layers, may vary. Figure 6b depicts a simple example of a neural network with one input layer, one output layer, and one hidden layer. For the sake of clarity, most of the constituent weights, biases as well as the activation functions of the perceptrons are omitted from the figure, and only some weights are kept. The input layer consists of L neurons, corresponding to the number of input features, i.e., $X = (x_1, x_2, \dots, x_L)$. The hidden layer consists of M perceptrons, i.e., $H = (h_1, h_2, \dots, h_M)$. There are N perceptrons in the output layer, corresponding to N output categories, i.e., $\hat{y} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_N)$.

Algorithm 1 The learning process

```

1: procedure LEARNING(training_data, epochs)
2:    $e = 0$ ;
3:   model = INITIALIZATION();
4:   while  $e < \text{epochs}$  do
5:     training_data = SHUFFLE(training_data);
6:      $X, y$  = SPLIT(training_data);
7:      $\hat{y}$  = PREDICT( $X, \text{model}$ );
8:     error = CALCULATE_ERROR( $y, \hat{y}$ );
9:     model = REFINE_MODEL(model, error);
10:     $e++$ ;
11:   end while
12:   return model
13: end procedure

```

We illustrate the learning process using the pseudo-code in Algorithm 1. In this listing, *epoch* is one round of learning performed on the training data, i.e., introducing all the input vectors. For each epoch, only some mini-batches of the training inputs are selected, and the training is done on these samples. At the beginning of each iteration, the training set is shuffled (Line 5) to randomize the input data, thus avoiding the problem of *being stuck in local minima* [62]. We consider a set of training inputs (X, y), and T is the batch size, i.e., the number of training samples fed to the network in one epoch.

The network is fed with input data X and labels y obtained after being split (Line 6), which is going to be detailed in Sect. 4.3.

The final aim of the learning process is to find a function that best fits the input data with the output data. We consider a single training sample ($X^{(t)}, y^{(t)}$), and for the sake of presentation, it is called (X, y) in the rest of the paper. The difference between the real category and the predicted value $\hat{y}$ (Line 7) is computed using an error function (mean square error, or cost function in various studies) as follows (Line 8).

$$E(\Omega, b) = \frac{1}{2} \|\hat{y}_k - y_k\|^2 \quad (1)$$

The error converges to zero when $\hat{y} \approx y$, i.e., the prediction matches with the real label. This boils down to minimize the error function $E(\Omega, b)$ by choosing a suitable set of weights and biases [14], which now become the function's variables, while the input training vector and its label become constants. Such a minimization is done by applying stochastic gradient descent (SGD) [17, 62]. We consider the example in Fig. 6b to illustrate how SGD works. The set of weights that bridges the input layer to the hidden layer is called $\omega^{(1)} = \{\omega_{ij}^{(1)}\}$, $i = \overline{1, L}$, $j = \overline{1, M}$, $\omega_{ij}^{(1)}$ is the weight connecting neuron i to neuron j ; the set of weights from the hidden layer to the output layer is called $\omega^{(2)} = \{\omega_{jk}^{(2)}\}$, the set of biases $B = \{b_j\}$, $b_j \in R$, $j = \overline{1, M}$, $k = \overline{1, N}$.

The predicted value $\hat{y}_k$ for neuron k of the output layer is computed using the following formula.

$$\hat{y}_k(h_j) = \sigma(b_k^{(2)} + \sum_{j=1}^M \omega_{jk}^{(2)} \cdot h_j) = \sigma(z_k^{(2)}) \quad (2)$$

where σ is the sigmoid function, which is defined as:

$$\sigma(z_k^{(2)}) = \frac{1}{1 + \exp(-z_k^{(2)})} \quad (3)$$

furthermore:

$$h_j(x_i) = \sigma(b_j^{(1)} + \sum_{i=1}^L \omega_{ij}^{(1)} \cdot x_i) = \sigma(z_j^{(1)}) \quad (4)$$

The gradient is computed exploiting the chain rule for differentiation:

$$\frac{\partial E}{\partial \omega_{jk}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial \omega_{jk}^{(2)}} \quad (5)$$

and

$$\frac{\partial E}{\partial b_k^{(2)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial b_k^{(2)}} \quad (6)$$

From Eqs. (1), (2), (3) the constituent derivatives are computed as:

$$\frac{\partial E}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \frac{1}{2} \|\hat{y}_k(h_j) - y_k\|^2 = \hat{y}_k^t(h_j) - y_k^t \quad (7)$$

$$\begin{aligned} \frac{\partial \hat{y}}{\partial z_k^{(2)}} &= \frac{\partial}{\partial z_k^{(2)}} \frac{1}{1 + \exp(-z_k^{(2)})} \\ &= \hat{y}_k^t(h_j) \cdot (1 - \hat{y}_k^t(h_j)) \end{aligned} \quad (8)$$

$$\frac{\partial z_k^{(2)}}{\partial \omega_{jk}^{(2)}} = \frac{\partial}{\partial \omega_{jk}^{(2)}} (b_k^{(2)} + \sum_{j=1}^M \omega_{jk}^{(2)} \cdot h_j) = h_j \quad (9)$$

In practice, the reduction of error becomes slow, and even with a large number of epochs, there is no improvement in the learning [55]. To this end, the cross-entropy cost function has been introduced to increase the learning efficiency, and it is defined as follows.

$$E_{CE} = - \sum_{k=1}^N \sum_{j=1}^M y_{jk} \cdot \log(f_j(x_i)) \quad (10)$$

The model performs prediction on the training data; as a new epoch, the gradient descent for the new weights and biases is updated by incorporating the error [37]. The error between the actual outcome and the predicted results is used to tune the model, and eventually to minimize errors (Line 9).

$$\omega_{jk}^{(2),(new)} = \omega_{jk}^{(2),(old)} - \gamma \sum_{t=1}^T \frac{\partial E}{\partial \omega_{jk}^{(2),(old)}} \quad (11)$$

and

$$b_k^{(2),(new)} = b_k^{(2),(old)} - \gamma \sum_{t=1}^T \frac{\partial E}{\partial b_k^{(2),(old)}} \quad (12)$$

where γ is called the learning rate, or *step size* which can be empirically set during the training; $t \in 1, \dots, T$ indicates an individual training sample; and T is the batch size, i.e., the total number of training instances fed as input in one iteration.

The outcome of the training phase is *model* with *weights* and *biases*, i.e., $\omega^{(1)}$, $\omega^{(2)}$, $B = \{b_j\}$, $b_j \in R$, $j = 1, M$ (Line 12 of **Algorithm 1**) that can be used to approximately produce the outputs from the inputs.

Algorithm 2 The testing process

```

1: procedure TESTING(model, testing_data)
2:    $X = \text{testing\_data}$ ;
3:    $results = \text{PREDICT}(X, \text{model})$ ;
4: end procedure

```

As it is shown in **Algorithm 2**, in contrast to learning, the testing process is much more straightforward where only the testing data, i.e., the metamodels that need to be classified, is fed to *model* that has been obtained from the training phase. The outcome is the predicted labels for the input data.

Summary. Learning is the process of refining the constituent weights and biases, by introducing all the instances of the training data. Meanwhile, testing is to predict a label for an input metamodel by means of the weights and biases obtained through learning.

Neural networks can also deal with multi-label classification by imposing some adaptations, e.g., using Bernoulli distributed variables instead of a single categorical variable. In this respect, the labels that belong to an instance have a

higher ranking than those that do not belong to the instance [81]. In the scope of this work, however, we focus on multi-class (or multinomial) classification [2], i.e., each instance of the data needs to be classified into one of the nine categories. During the prediction phase, for each output of the neural network in Fig. 6b, the classifier returns a real score, and we assign the output with the highest score to the metamodel. Eventually there is only a label for each testing metamodel.

3.2 Realization

The framework for classification is depicted in Fig. 7 and it consists of the training and deployment phases which are going to be clarified in this subsection.

The training phase starts with collecting metamodels from public repositories like GITHUB^① using the CRAWLER component ^②. In particular, while metamodels can be collected from different sources, in the scope of this paper we make use of GitHub as the main data provider, since the platform offers a huge number of metamodel repositories. Afterward, expert modelers are asked to inspect the metamodels and manually classify them into independent groups based on their experience. By means of labeled data including metamodels and labels assigned by humans, the training process is in charge of identifying the right weights to configure the WEIGHTS CALCULATOR component. Then, the NLP PROCESSOR component ^③ parses terms from the input metamodel and normalizes the extracted raw terms by performing on all the named elements, which are contained in the collected metamodels the following natural language processing (NLP) steps:

- *Stemming*: this phase is conducted to squeeze all words that share a same root to a common term;
- *Lemmatization*: it is performed to remove inflectional endings, attempting to restore words to their dictionary form;
- *Stop words removal*: articles or supporting words such as prepositions, e.g., *the*, *is*, *to*, *which*, are removed from the corpus.

The DATA ENCODER component ^④ takes as input the manipulated metamodels and encodes them as matrices enabling the subsequent phases of the approach. Input *configuration* directives are fed to select the scheme to be used for encoding the metamodels, i.e., uni-gram, bi-gram, and n-gram, and they are explained below.

- *uni-gram* represents a simple collection of terms which do not preserve any metamodel structure;
- *bi-gram* partly represents the structure of the metamodels by considering the containment relations between named

Fig. 7 AURORA's architecture

elements e.g., classes vs. structural features, packages vs. classes, etc.;

- *n-gram* represents a metamodel structure by enriching bi-grams with attribute properties, e.g., typing information, cardinality, and reference multiplicity.

To filter out those terms that do not occur frequently, *cutoff* values are given: They define the least number of term occurrences in all the metamodels. Less frequent terms represent singularities that must be avoided. On the other hand, higher cutoff values give place to accuracy problems. Afterward, the DATA ENCODER component ④ yields *filtered* feature vectors whose terms have a frequency higher than the cutoff values. The resulting feature vectors, also called term document matrix (TDM), is a two-dimensional matrix where each row represents a metamodel in the repository and columns are the extracted terms, so each entry (i, j) represents the frequency of term j in metamodel i .

Subsequently, the feature vectors, and the corresponding labels are used to train AURORA through the WEIGHTS CALCULATOR component ⑤. The result consists of the weights and biases that are going to be used by the neural network to classify any incoming metamodel in the deployment phase. Whenever an unlabelled metamodel ⑥ is fed to AURORA, it is parsed through the NLP PROCESSOR ⑦ and the DATA ENCODER ⑧ components; the generated feature vector is then put into the CLASSIFIER component ⑨ that, in turn, performs the needed classification and assigns a label to the vector. Finally, the outcome is a label that classifies the metamodel given as input.

We take an example to illustrate how the data processing phase works by considering the *Library* metamodel shown in Fig. 8. The metamodel is used to generate models, where a *Library* may contain a set of *Books* together with *Writers*. A writer in turn can write books, and a book can be composed by different writers. A writer is determined by a *firstname* and a *lastname*, while a book is featured by a *title* together with the number of *pages*.

The constituent terms in the metamodel are extracted and encoded using the aforementioned encodings. We give a sum-

Fig. 8 Library metamodel

mary in Table 1 the final results by showing all the terms of *Library*, as well as the corresponding encodings for the raw vectors and the normalized feature vectors. Each element name is characterized by an identifier and its type. For example, the package name *library* is (i) individually encoded in uni-grams; (ii) in bi-grams as part of the terms of the meta-classes #2, #6, and #10; and (iii) in n-grams as part of the terms of the structural features #3, #4, #5, #7, #8, #9, #11, #12, and #13. With respect to the normalized feature vectors, the terms need to be parsed through the following phases: stemming, lemmatization, and stop words removal. As a result, the term *books* is converted into *book*, containing both the metaclass *Book* and the feature *books*. Furthermore, additional information about cardinalities (e.g., 0.1) and whether the node is subject to containment (e.g., TRUE) is eventually incorporated.

3.3 Baselines: gradient boosted decision tree and support vector machines

In recent years, several machine learning approaches [45] have been proposed to address the classification problem. In this section, we review two techniques, i.e., gradient boosted decision tree [31] and support vector machines [78], that are exploited as additional classification engines for metamodels in this work. These two techniques have been widely adopted

Table 1 Extraction of the *Library* metamodel

Metamodel element Type	id	Name	Raw feature vector			Normalized feature vector		
			uni-gram	bi-gram	n-gram	uni-gram	bi-gram	n-gram
Package	#1	Library	Library	-	-	Library*	-	-
Metaclass	#2	Library	Library	Library.Library	-	Library*	Library.library	-
Attribute	#3	Name	Name	Library.name	Library.name.EString.0.1	Name	Library..name	Library.name.estring.0.1
Reference	#4	Writers	Writers	Library.writers	Library.writers.Writer.false.0.-1	Writ**	Library.writ	Library.writ.false.0.-1
Reference	#5	Books	Books	Library.books	Library.books.Book.false.0.-1	Book***	Library.book	Library.books.book.false.0.-1
Metaclass	#6	Writer	Writer	Library.Writer	-	Writ**	Library.writ	-
Attribute	#7	FirstName	FirstName	Writer.firstName	Writer.firstName.EString.0.1	FirstName	Writ.firstName	writ.firstName.estring.0.1
Attribute	#8	LastName	LastName	Writer.lastName	Writer.lastName.EString.0.1	lastName	Writ.lastName	writ.lastName.estring.0.1
Reference	#9	BBooks	Books	Writer.books	Writer.books.Book.false.0.-1	Writ***	Writ.book	Writ.books.book.false.0.-1
Metaclass	#10	Book	Book	Library.Book	-	Book***	Library.book	-
Attribute	#11	Title	Title	Book.title	Book.title.EString.0.1	Titl	Book.titl	Book.titl.estring.0.1
Attribute	#12	Pages	Pages	Book.pages	Book.pages.EInt.0.1	Page	Book.pag	Book.pag.eint.0.1
Reference	#13	Writers	Writers	Book.writers	Book.writers.Writer.false.0.-1	Writ**	Book.writ	Book.pag.eint.0.1

in various domains [22,53,65], and we further study their performance in MDE for the classification of metamodels.

- In machine learning, weak classifiers are those that are modified to enhance their prediction capability [49]. This can be done by restraining classifiers from using all the available information, or by simplifying their internal structure. Such an intervention enables classifiers to learn from different parts of a training set, resulting in a reduction in error correlation. *Gradient boosted decision tree* [31] (GBDT) is an ensemble model of decision tree to predict a target label. During its learning phase, GBDT populates a series of weak decision trees, where each of them attempts to correct errors from the previous one. In other words, a GBDT learns by relying on the following assumption: the observations that a weak decision tree classifier can handle are left, while the remaining difficult observations are subject to developing new weak decision trees.
- The *support vector machine* algorithm [74] (SVM) can also be used to classify metamodels. First, it represents metamodels as labeled points in an N-dimensional space; then, to learn from data, it extracts hyperplanes that better split the points among the categories. A hyperplane can be understood as a border between two classes. Its dimension varies according to the number of input features, i.e., N. In this way, the learning phase is an optimization process that attempts to find hyperplanes maximizing the distance among points in independent classes. Whenever an unlabeled metamodel is fed into SVM, it is transformed into a data point; then, the generated hyperplanes are used to assign a label to the metamodel.

In the evaluation, we investigate the performance of both classifiers in comparison with our proposed approach.

4 Evaluation

In this section, we analyze the performance of our proposed classifier employing a thorough evaluation based on two datasets as well as various quality indicators. The section is organized as follows: Sect. 4.1 presents the research questions. The dataset used for evaluation is introduced in Sect. 4.2. Section 4.3 explains the applied methodology, and Sect. 4.4 recalls the evaluation metrics.

4.1 Research questions

We study the performance of our proposed approach by considering the following research questions:

- **RQ₁**: Which encoding scheme is most useful for classifying metamodel repositories? We experiment with

Table 2 Network configurations

Enc.	k	# of perceptrons			1# of params
		Layer 1	Layer 2	Layer 3	
uni-gr.	2	4,356	10	9	43,699
	4	2,412	10	9	24,229
	6	1,715	10	9	17,259
	8	1,240	10	9	12,509
bi-gram	2	6,895	10	9	69,509
	4	2,665	10	9	26,579
	6	1,747	10	9	17,579
	8	1,174	10	9	11,849
n-gram	2	10,837	10	9	108,749
	4	4,166	10	9	41,769
	6	2,749	10	9	27,599
	8	1,733	10	9	17,439

various configurations to study which encoding technique between uni-gram, bi-gram, and n-gram facilitates a better prediction performance.

- **RQ₂**: *How does the cutoff value affect the classification's outcome?* Parsing metamodels with a small cutoff value results in a big feature set as given in Table 2, thus increasing the computational complexity. We are interested in ascertaining whether a compact feature set can yield a good trade-off between accuracy and efficiency.
- **RQ₃**: *Does the cross-entropy cost function account for a gain in performance?* In our previous work [58], we employed the quadratic cost function, and still, the obtained results were promising. In this paper, we incorporated the cross-entropy error function, which is supposed to be more effective and efficient in computing errors, with the ultimate aim of further improving the performance.
- **RQ₄**: *Is an increase in the network's width or depth beneficial to the performance?* We investigate whether changing the depth or width of the network significantly influences the prediction performance.
- **RQ₅**: *How does AURORA compare with GBDT and SVM?* Neural networks are a well-founded machine learning technique. In this paper, we compare AURORA with two more supervised learning algorithms, namely gradient boosted decision trees (GBDT) and support vector machines (SVM).

4.2 Dataset

We made use of an existing dataset¹⁰ which consists of metamodels related to domain-specific languages (DSLs). The metamodels were inspected and manually classified by

expert modelers based on their experience [64]. The final dataset consists of 555 metamodels distributed in nine independent categories. The results of the classification process are summarized in Fig. 9a. In contrast, Fig. 9b depicts the distribution of the metamodels with respect to the number of metamodels, the number of metaclasses, and the number of features. According to Fig. 9a, the most populous category is *State machine DSLs* with 159 metamodels, accounting for 29% of the total amount. Meanwhile, the smallest category is *Issue Tracker DSLs* with only 7 metamodels. Such a category may render the classification difficult since there is a limited amount of data for training. The distribution of metamodels with respect to the number of metaclasses, attributes, and references is depicted in Fig. 9b, where most of the metamodels contain a low number of metaclasses as well as attributes and references. The metamodels are concentrated on the bottom left corner of the tensor. Only those of the category *Review system DSLs* contain a large number of metaclasses, and they are scattered across the 3D space.

The metamodels have been manually classified following humans' perception. Such a classification might vary from modelers to modelers in practice when it comes to the application domain. For simplicity, we consider the application domain as the engineered intent in various metamodels, even if some could be classified into various categories, depending on the granularity level. To be concrete, *Office-tools DSLs*, *State Machine DSLs* can be considered as general-purpose w.r.t. *DSLs*, which implement specific domains by definition. Moreover, we can see that there might be a possible overlap between the categories; for instance, the *Review systems DSLs* category appears to have much in common with *Bibliography DSLs*. Data drives the performance of AURORA, and, once being trained, the tool is expected to do exactly what has been dictated in the training data. In this way, we assume that depending on the input data, we can get different granularity levels in the classification outcomes.

To discard those terms that are considered sparse for the whole corpus (and as such not too representative), the following cutoff values are considered $k = \{2, 4, 6, 8\}$. The scheme is inspired by one of the preprocessing phases done in the domain of Document Representation to reduce the dimensionality of a set of documents [77]. Larger values of c would miss the intrinsic features of the dataset by excluding too many terms. Table 2 describes the network configuration used in the experiments. The number of input features corresponds to the number of perceptrons in the first layer of the neural network depicted in Fig. 6b. It is worth noting that using n-gram as encoding scheme results in the largest number of input perceptrons, i.e., 10,837 neurons when $k = 2$, while uni-gram needs much fewer neurons to represent a metamodel, e.g., 1,240 neurons when $k = 8$. For the most basis configuration, we use only 10 perceptrons in the second layer. In Sect. 5.4, such number is going to be adjusted to

¹⁰ <https://doi.org/10.5281/zenodo.2585456>

Fig. 9 A summary of the dataset

study the influence of the network's depth and width on the performance. The last column of Table 2 specifies the number of parameters corresponding to the considered network. It is evident that among the configurations, parsing data with n -gram and $c = 2$ results in the largest number of parameters, i.e., 108,749. In essence, the computational complexity should fluctuate according to the chosen encoding techniques as well as the number of internal parameters.

4.3 Experimental settings

The tenfold cross-validation technique has been utilized to study AURORA's performance, as it is retained among the best methods for model selection in machine learning [44]. In such a setting, the original dataset is divided into ten equal parts, so-called *folds*, and numbered from 1 to 10. For each test configuration, the validation is conducted in ten rounds. For each round i , $fold_i$ is used as *test set* and the remaining nine folds $fold_j$ ($j \neq i, 1 \leq i, j \leq 10$) are combined to form a *training set*. The training data is used to build the neural network following the paradigm presented in Sect. 3, i.e., to identify a set of weights and biases that best produce the corresponding outputs, given the inputs, while the testing data is used to validate the system. Like in the training set, every metamodel in the test set does have a category (label) associated with it. However, for the validation phase, such a label is removed and saved as ground-truth data. Only the feature vector of a testing metamodel is built using one of the encoding schemes mentioned in Sect. 4.2. Each vector is then fed into the network, which eventually generates a label. This label is compared against the one stored as ground-truth data to see if they match. It is expected that the labels for all testing metamodels will match with those stored as ground-truth data.

The validation process simulates a real deployment as follows. The training data corresponds to the repositories being available for mining; they are collected from different sources

Fig. 10 Relationship between TP_c , TN_c , FP_c , and FN_c [69]

and manually classified by humans, i.e., when developers create a metamodel. Meanwhile, the testing data represents the repositories that need to be classified. Thus, the validation process attempts to investigate whether AURORA applies to real deployment, e.g., being suitable for classifying meta-model crawled from GitHub.

For the sake of reproducibility, we published in GitHub the AURORA tool together with all metadata parsed from the original dataset.¹¹

4.4 Metrics

We consider S as a set of metamodels and they need to be classified into C independent categories. For a category $c \in C$, we call $\hat{Y}_c$ as the set of metamodels that has been classified to c by any of the recommendation engines, i.e., GBDT, SVM, and AURORA (cf. Sects. 3.1 and 3.3), and Y_c as the actual set of metamodels of c in the ground-truth data, i.e., the data manually labeled by humans.

First, we explain the notations used in this paper by referring to the Venn diagram in Fig. 10. Given a category $c \in C$, we consider the following definitions:

¹¹ <https://github.com/MDEGroup/AURORA/>

- *True positive (TP)*: the number of metamodels that are classified to $\mathbf{c}$, and they belong to the ground-truth data of $\mathbf{c}$, i.e., $TP_c = |\hat{Y}_c \cap Y_c|$.
- *True negative (TN)*: the number of metamodels that are classified to other categories rather than $\mathbf{c}$, and they do not belong to the ground-truth data of $\mathbf{c}$, i.e., $TN_c = \sum_{d \in C, d \neq c} |\hat{Y}_d|$.
- *False positive (FP)*: the number of metamodels that are classified to $\mathbf{c}$, but they do not belong to the ground-truth data of $\mathbf{c}$, i.e., $FP_c = |\hat{Y}_c \setminus Y_c|$.
- *False negative (FN)*: the number of metamodels that are not classified to $\mathbf{c}$, but they belong to the ground-truth data of $\mathbf{c}$, i.e., $FN_c = |Y_c \setminus \hat{Y}_c|$.

In the scope of this paper, we are interested in understanding how well the produced categories match with those that have been manually classified. In particular, we measured the relevance of the classified results obtained by running the three tools, i.e., GBDT, SVM, and AURORA with the ground truth using *accuracy*, *precision*, *recall*, *F₁-score*, *ROC*, and *AUC*, which are defined as follows.

4.4.1 Accuracy

Accuracy is the ratio of the number of correctly classified items to the total number of items, regardless of their categories.

$$Accuracy = \frac{\sum_{c \in C} TP_c}{|S|} \times 100\% \quad (13)$$

4.4.2 Precision and Recall

Precision measures the fraction of the number of correctly classified metamodels to category $\mathbf{c}$ to the total number of classified metamodels in this category. At the same time, Recall is the ratio of the number of correctly classified items to $\mathbf{c}$ to the total number of items in the ground-truth data of $\mathbf{c}$. Recall is also named as true positive rate (TPR).

$$P_c = \frac{TP_c}{TP_c + FP_c} \quad (14)$$

$$R_c = TPR_c = \frac{TP_c}{TP_c + FN_c} \quad (15)$$

4.4.3 F₁-score (or F₁)

The metric is a combination of precision and recall.

$$F_{1c} = \frac{2 \cdot P_c \cdot R_c}{P_c + R_c} \quad (16)$$

4.4.4 False positive rate (FPR)

This metric measures the ratio of the number of items that are falsely classified into category $\mathbf{c}$, to the total number of items that are either correctly not classified, or falsely classified into the category.

$$FPR_c = \frac{FP_c}{TN_c + FP_c} \quad (17)$$

Finally, we compute the evaluation metrics by averaging out the corresponding scores for all the categories.

$$P = \frac{\sum_{c \in C} P_c}{|C|} \quad (18)$$

$$R = \frac{\sum_{c \in C} R_c}{|C|} \quad (19)$$

$$F_1 = \frac{\sum_{c \in C} F_{1c}}{|C|} \quad (20)$$

4.4.5 ROC curve and AUC

We represent the relationship between FPR and TPR using a receiver operating characteristic (ROC) in a 2D space [27], where the x-axis and the y-axis correspond to FRP and TPR, respectively. An ROC curve spans from (0,0) to (1,1) in the space.

In this paper, we deal with multi-class classification, where a metamodel needs to be assigned to one (and only one) of the nine considered categories. However, ROC curves are used for binary classification by their nature. Therefore, we need to transform the multi-class problem into binary classification using the following workaround. For each specific category (among the nine considered categories), i.e., c_i , we consider it the positive class, while all the remaining eight categories are considered negative. That means, if a metamodel $\mathbf{m}$ is correctly classified into its positive class, we count it as *true*. Similarly, any metamodels that do not belong to c_i and are not classified to c_i are also marked as *true*. In contrast, if $\mathbf{m}$ is classified into any of the negative classes, it is counted as *false*; or vice versa, if any metamodels that do not belong to c_i but are classified to c_i are also marked as *false*. In this way, instead of multi-class classification, we end up with binary classification, where there are two distinct classes: **True** (i.e., true positive and true negative) and **False** (i.e., false positive and false negative). A good classifier should assign a high probability to the former while giving a low probability to the latter. In other words, the classifier is expected to have a low FPR and obtain a high TPR. We can define a threshold to predict the classes with high confidence, given a certain probability of being distinctive. In this way, an ROC curve is

plotted in the 2D space by varying such a threshold.¹² This allows one to easily comprehend the performance at different levels of the decision that a classifier's outcome is accepted.

Altogether, this implies that an ROC curve locating closer to the upper left corner corresponds to a better prediction performance than the one that resides at the lower right corner in the 2D space. The area under the ROC curve (AUC) is an explicit indication of how good a classifier is. A *dummy* classifier, i.e., the one that randomly assigns a label to meta-models, has an AUC value of 0.5, while a perfect classifier has an AUC value of 1.0.

In the next section, we present in detail the experimental results by referring to the research questions in Sect. 4.1.

5 Experimental results

By executing the experiments, we computed accuracy, precision, recall, and F₁ scores for every test configuration with respect to the error function (EF) in two cases: quadratic (QD) and cross-entropy (CE). The final results are depicted in Table 3. We answer the aforementioned research questions by referring to the table. For **RQ₁** and **RQ₂**, we consider only the configurations with the quadratic error function. Meanwhile, for **RQ₃**, the results obtained by using the cross-entropy function are incorporated to study the effect of changing the error function. In **RQ₄**, we analyze whether an increase in the number of neurons or network depth enhances the overall accuracy. Finally, we compare AURORA with GBDT and SVM in **RQ₅**.

5.1 RQ₁: Which encoding scheme is most useful for classifying metamodel repositories?

We evaluate which technique between uni-gram, bi-gram, and n-gram yields the best prediction outcome, given that the quadratic error (QD) function is used. According to Table 3, we see that between bi-gram and n-gram, the former always contributes to better performance. For instance, when $k = 2$, using n-gram yields 84.30% as the accuracy, meanwhile the corresponding value by bi-gram is 85.90%. This applies also to the other cutoff values, i.e., $k = \{4, 6, 8\}$. The lowest accuracy for bi-gram obtained by $k = 8$ is 77.40%, which is better than 72.40%, the corresponding value by n-gram. By other quality metrics, i.e., precision, recall, and F₁ score, bi-gram always brings in a superior prediction outcome. The best precision obtained with bi-gram is 0.899 when $k = 2$, while using n-gram yields 0.841 as precision. Similarly, the best recall by bi-gram and n-gram is 0.782 and 0.759, respectively. The best F₁ scores by using bi-gram and n-

Table 3 Evaluation results

Enc.	Metric	EF	Cutoff value (k)			
			2	4	6	8
uni-gram	Acc. (%)	QD	93.44	92.70	93.00	91.70
		CE	94.30	95.60	94.50	93.70
	Precision	QD	0.941	0.926	0.935	0.903
		CE	0.945	0.958	0.952	0.941
	Recall	QD	0.842	0.902	0.835	0.900
		CE	0.874	0.907	0.892	0.885
	F ₁ -score	QD	0.889	0.914	0.882	0.902
		CE	0.910	0.932	0.921	0.912
bi-gram	Acc. (%)	QD	85.90	80.60	77.12	77.40
		CE	86.80	81.40	79.90	78.10
	Precision	QD	0.899	0.814	0.781	0.811
		CE	0.903	0.820	0.785	0.827
	Recall	QD	0.782	0.768	0.725	0.711
		CE	0.804	0.781	0.761	0.724
	F ₁ -score	QD	0.828	0.791	0.752	0.758
		CE	0.849	0.800	0.773	0.772
n-gram	Acc. (%)	QD	84.30	79.40	75.10	72.40
		CE	85.60	80.30	75.20	74.60
	Precision	QD	0.841	0.791	0.761	0.737
		CE	0.874	0.783	0.767	0.771
	Recall	QD	0.759	0.748	0.678	0.643
		CE	0.802	0.758	0.699	0.671
	F ₁ -score	QD	0.798	0.769	0.717	0.687
		CE	0.837	0.771	0.732	0.717

Bold values represent the better evaluation results

gram are 0.828 and 0.798, respectively, implying that bi-gram surpasses n-gram by all evaluation metrics.

Among others, uni-gram helps gain the best performance concerning all quality metrics. In particular, the maximum accuracy yielded by using uni-gram is 93.44%, which is considerably higher compared to 85.90% and 84.30% by bi-gram and n-gram, respectively. Being consistent with the accuracy scores, uni-gram is the most useful encoding since it gets a precision being larger than 0.90. Again, uni-gram still contributes to the best performance in comparison with bi-gram and n-gram. Using uni-gram with a large cutoff value can still obtain a good performance. For example, when $k = 4$ the accuracy is 95.60% which is better than 94.30%, the corresponding score when $k = 2$.

Finally, we investigate whether the observed differences are statistically significant. To this end, we perform ANOVA tests on the results obtained by the encoding schemes. In particular, we compare the three encodings pairwise, resulting in three comparison configurations as follows. C₁: n-gram vs. bi-gram; C₂: uni-gram vs. bi-gram; and C₃: uni-gram vs. n-gram. Table 4 reports which encodings have signifi-

¹² This is conveniently done in Python using the `roc_curve()` function of the `sklearn.metrics` package.

Table 4 Tukey's range test for $k = \{2, 4, 6, 8\}$

	Conf.	diff	lwr	upr	p-adj
P	C ₁	-0.035	-0.116	0.047	0.485
	C ₂	0.116	0.034	0.198	0.008
	C ₃	0.151	0.069	0.233	0.001
R	C ₁	-0.034	-0.113	0.044	0.479
	C ₂	0.123	0.044	0.201	0.005
	C ₃	0.157	0.078	0.235	0.001
F₁	C ₁	-0.034	-0.109	0.040	0.435
	C ₂	0.120	0.046	0.194	0.003
	C ₃	0.155	0.080	0.229	0.001

cant differences in evaluation metrics compared to others. For each pairwise comparison, the column *diff* shows the actual difference between the means, the *p-adj* column represents the adjusted p-value, while the *lwr* and *upr* columns collect the lower-end and upper-end points of the interval, respectively. Two groups may be different, and the null hypothesis can be rejected if the interval between *lwr* and *upr* does not include 0, or if the value of *p-adj* < 0.05. For instance, there is a significant difference in precision values between uni-gram and n-gram, i.e., adjusted p-value *p-adj* = 0.001, as well as between uni-gram and bi-gram, i.e., *p-adj* = 0.008, while there is no significant difference in precision values between n-gram and bi-gram encodings, i.e., *p-adj* = 0.485.

Besides the three aforementioned encoding algorithms, i.e., uni-gram, bi-gram, and n-gram, we also tried to embed word2vec [52] and BERT [24] as two additional encodings. Nevertheless, by means of an empirical evaluation, we realized that using AURORA with data parsed by word2vec yields a mediocre performance. This happens due to the fact that word2vec uses a context window with an arbitrary size to represent the terms of a document. Unlike the n-gram encoding that has been specifically tailored to represent metamodels, the technique involves a term within a context that does not reflect metamodels' structure. Meanwhile, BERT cannot be used to parse metamodels as it generates a completely different output, which is an incompatible format to AURORA. We anticipate that BERT can be exploited as an independent classifier by training with the bert-based model and the uni-gram representation of each training metamodel. Nevertheless, this is out of the scope of the current work. Thus, we consider the issue as future work.

Answer to RQ₁. Uni-gram is the most effective encoding scheme with respect to all metrics; bi-gram helps obtain a better outcome than n-gram does. The differences in performance among the encodings are statistically significant.

5.2 RQ₂: How does the cutoff value affect the classification's outcome?

Given that the quadratic loss function is used, we study how the cutoff value influences the performance by all the encoding schemes in Table 3. First, by uni-gram, the accuracy fluctuates among the cutoff values; however, there is no distinct difference in the accuracy. For instance, the maximum score obtained when $k = 2$ is 93.44%, which is slightly better than 93.00%, the accuracy when $k = 6$. When the cutoff value increases, the accuracy slightly changes. Take as an example; the accuracy decreases to 91.70% when $k = 8$. Concerning precision, we see a similar pattern. Though the scores vary through the cutoff values, there is no significant improvement in precision when we incorporate more features as the input data. The best precision corresponding to 0.941 is obtained when $k = 2$, while the worst precision 0.903 is achieved when $k = 8$. By recall and F₁, the best performance is seen when $k = 4$, while by other cutoff values, there is no clear distinction between the outcomes.

Considering bi-gram, it is evident that increasing the cutoff value induces a reduction in performance concerning all evaluation metrics. For example, when $k = 2$, AURORA achieves an accuracy of 85.90%, while it gets 80.60% correctly classified items when $k = 4$. The precision, recall, and F₁ scores gained by using bi-gram also decrease considerably along with the cutoff values.

The results demonstrate a clear difference in performance when using n-gram as the encoding scheme, i.e., all the evaluation metrics considerably decrease when c increases. Take as an example, the accuracy changes from 85.60% to 80.30% when we change k from 2 to 4. The accuracy declines further down to 75.10% when $k = 6$. By other metrics, i.e., precision, recall, and F₁ scores, the same trend can be observed: they substantially fall when we use fewer features to feed AURORA.

Answer to RQ₂. Using uni-gram, there is no distinct difference in performance when the cutoff value changes. However, by bi-gram and n-gram, a better prediction performance is achieved for a more detailed feature set.

5.3 RQ₃: Does the cross-entropy cost function account for a gain in performance?

By changing the loss function, we expect to get a gain in performance. To answer this research question, we consider the scores presented in Table 3 and Figs. 11a, b, c and 12a, b, c.

From Table 3, it is clear that when the cross-entropy (CE) cost function is used, better accuracy is obtained by all encoding schemes as well as cutoff values. For instance, when uni-gram is used to parse the metamodel, and $k = 4$ we get 95.60% as accuracy, which is also the maximum

value among all the configurations. Similarly, by other cutoff values for uni-gram, the cross-entropy error function facilitates a superior performance in comparison with the quadratic one. By bi-gram, we witness the same trend: The cross-entropy function contributes to better performance. In particular, we get 86.80%, 81.40%, 79.90%, and 78.80% as accuracy for $k = \{2, 4, 6, 8\}$, respectively. Meanwhile, the corresponding scores by employing the quadratic function are 85.90%, 80.60%, 77.12%, and 77.40%. The same effect can also be seen with n-gram: By all cutoff values, the cross-entropy function helps achieve a better accuracy compared to the quadratic cost function.

Similarly, by other metrics, i.e., precision, recall, and F1, it can be seen that using cross-entropy as the loss function brings in a performance gain. Concerning precision, as shown in Table 3, only by bi-gram there is a slight increment in precision with cross-entropy compared to quadratic, but by the other encoding schemes, the cross-entropy function fosters a better classification performance. For example, with n-gram and $k = 2$, 0.874 is the precision compared to 0.841 by the quadratic function using the same setting. Similarly, the recall and F1 scores yielded by the cross-entropy function are always superior to those yielded by the quadratic cost function.

We further compare the two loss functions by reporting the ROC curves for all the categories in Figs. 11a, b, c and 12a, b, c. By observing Figs. 11a and 12a together to compare the two functions with respect to using uni-gram as the encoding scheme, it is evident that the ROC curves corresponding to the cross-entropy function converge to the upper left corner of the diagram, implying a high true positive rate as well as a low false positive rate. Moreover, the AUC values for the categories range from 0.97 to 1.00. Meanwhile, the quadratic function suffers a low prediction accuracy for a category with a small number of metamodels. In particular, in Fig. 11a by Class 2 (*Issue Tracker DSLs*) with only seven metamodels, the corresponding ROC curve is close to the diagonal line, which corresponds to a mediocre performance. Similarly, by considering pairwise the figures, either Figs. 11b and 12b, or 11c and 12c, the same trend can be witnessed: In comparison with the quadratic function, the cross-entropy one contributes to a more effective classification of the metamodels into their correct categories.

Altogether, we conclude that the application of cross-entropy as the error function brings about an improvement in the performance, compared to the original AURORA design [58]. Moreover, it is worth noting that also for this configuration, uni-gram is the most suitable encoding scheme as it contributes to a gain in performance in comparison with bi-gram and n-gram.

The gain in performance by using cross-entropy can be explained as follows. The quadratic function is based on mean squared error (MSE), which is indeed more suitable

Table 5 Accuracy (%) for various numbers of perceptrons

M	10	20	30	40	50	100
Acc.	95.00	94.50	94.82	93.62	94.84	93.20

for linear regression by its definition. Meanwhile, the cross-entropy function is more suitable for classification as it works with particular sets of output values, i.e., multi-nominal classification. Altogether, we see that this is important in practice since we can substantially improve the prediction performance by choosing a suitable loss function, depending on the requirements.

Answer to RQ₃. In comparison with the quadratic loss function, the cross-entropy loss function fosters a superior prediction performance.

5.4 RQ₄: Is an increase in the network's width or depth beneficial to the performance?

We investigate whether an increase in the network's width or depth boosts up the prediction accuracy. Figure 6b depicts a simple neural network where there are one input, one hidden and one output layer. In practice, the number of perceptrons, i.e., the network's width as well as the number of hidden layers, i.e., the network's depth can be flexibly changed to aim for better accuracy. However, increasing either depth or width contribute toward computational complexity. To this end, we are interested in understanding which network topology sustains the overall performance, i.e., how many hidden layers as well as how many neurons for each layer should be chosen to maintain a trade-off between accuracy and efficiency. To the best of our knowledge, there exist some rules of thumb to set the number of neuron M for a hidden layer, e.g., $L \leq M \leq N$ [15], or $M \leq 2 \times L$ where L, N are the number of input and output neurons, respectively (cf. Fig. 6b). Still, selecting the right topology for neural networks, i.e., the number of layers, as well as the number of nodes for each hidden layer remains an empirical matter [80].

First, we examine how an increase in the network's width affects performance. For this experiment, we chose the best configuration with the following settings: cross-entropy error function, cutoff value $k = 2$, and uni-gram as the encoding scheme. The number of perceptrons has been varied as follows: $M = \{10, 20, 30, 40, 50, 100\}$, and the corresponding accuracies are depicted in Table 5.

The table shows that generally, there is no radical improvement in accuracy when M is increased. There is only a tiny growth in the accuracy when M is changed from 20 to 30, i.e., the corresponding accuracies are 94.50% and 94.82%. Interestingly, when $M = 40$, there is a decrease in the accuracy compared to when $M = 30$. Similarly, when M rises to

Fig. 11 ROC curves obtained by using the quadratic loss function

Fig. 12 ROC curves obtained by using the cross-entropy loss function

100 from 50, an apparent reduction in the accuracy occurs, i.e., 94.84% compared to 93.20%. Overall, an increase in the number of perceptrons does not help enhance prediction accuracy. Moreover, the execution time increases linearly with the number of perceptrons. To further investigate the effect of changing M , we also sketched the ROC curves for these experiments. We noticed that there are no significant differences between the performance obtained by changing M . Thus, the corresponding images are omitted from the paper for the sake of clarity. Similarly, we have seen that there is no evident change in the AUC values obtained for various categories. In this sense, we conclude that on the given dataset, a growth in the number of perceptrons in the hidden layer is not beneficial to performance, since it not only hampers accuracy gain but also adds up to the computational complexity.

Second, to see the effect of changing the network's depth, we fixed 10 as the number of perceptrons for each layer, i.e., width=10, and varied the number of hidden layers, i.e., $\rho = \{2, 3, 4, 5, 10\}$. We obtained similar results: There is no definite improvement in performance when we add more hidden layers, while the computational complexity goes up very quickly. For the sake of presentation and clarity, the corresponding results are omitted from the paper.

Answer to RQ₄. On the given dataset, an increase in the network's depth or width does not help AURORA enhance its classification performance.

5.5 RQ₅: How does AURORA compare with GBDT and SVM?

Besides neural networks, GBDT and SVM are among the most well-founded machine learning algorithms [31,74]. They have been widely applied in various domains, such as pattern classification [22], face recognition [65], image segmentation [53], or recommender systems [40]. In this research question, we perform additional experiments to see how well AURORA works compared to GBDT and SVM in terms of prediction performance. To simplify the presentation, we make use of uni-gram to encode the data for this evaluation. Moreover, aiming for a fair comparison, we run ten-fold cross-validation using exactly the same input data for all the three classifiers, i.e., GBDT, SVM,¹³ and AURORA. The experimental results are shown in Table 6.

Concerning accuracy, it is evident that AURORA always obtains a superior performance compared to both GBDT and SVM. For instance, with $c=2$, AURORA gets 94.30% as

¹³ We made use of the GBDT and SVM implementations embedded in the **scikit-learn** library (www.scikit-learn.org).

Table 6 Comparison of GDT and SVM with AURORA (AUR.)

	c=2			c=4			c=6			c=8		
	GBDT	SVM	AUR.	GBDT	SVM	AUR.	GBDT	SVM	AUR.	GBDT	SVM	AUR.
Accuracy (%)	92.20	75.41	94.30	91.72	79.20	92.70	92.41	81.43	93.10	92.92	82.84	92.98
Precision	0.880	0.662	0.945	0.874	0.699	0.958	0.883	0.703	0.952	0.895	0.734	0.941
Recall	0.915	0.795	0.874	0.894	0.830	0.907	0.903	0.824	0.892	0.920	0.829	0.885
F ₁ -score	0.886	0.692	0.910	0.870	0.718	0.932	0.884	0.718	0.921	0.896	0.747	0.912

Bold values represent the better evaluation results

accuracy, while the corresponding value by GBDT and SVM is 92.20% and 75.41%, respectively. Similarly by other cutoff values, AURORA also outperforms GBDT and SVM with respect to accuracy.

With precision, the same trend can be witnessed, i.e., AURORA gets the best precision compared to the baselines by all the cutoff values. Only by recall, GBDT is the classifier that gets the best performance by almost all configurations, i.e., $c=\{2,6,8\}$. This means that GBDT is good at retrieving the metamodels that belong to the ground-truth data. However, when we consider F₁-score, it is clear that AURORA is the best classifier since it always achieves the largest F₁ value. Altogether, by taking all the evaluation metrics into consideration, we conclude that AURORA brings the best classification results, among others.

Answer to RQ₅. AURORA outperforms both GBDT and SVM by most of the quality metrics.

6 Discussions

This section discusses the lessons learned as well as the envision for future developments.

- Encoding** The experimental results indicate that the techniques used to parse and represent metamodel play an essential role in obtaining an excellent performance. We came across an exciting outcome: Running AURORA on a feature set parsed by uni-gram leads to the best classification results. This appears to be counterintuitive at first sight because the structure in metamodels has highly informative content, and as given in Sect. 3.2, uni-gram is not structure-preserving. A possible explanation of why uni-gram fosters the best classification performance might be related to the nature of the manual classification of the dataset involved in the experiment. In essence, the assigned labels reflect categories that are unrelated to the structural information of the metamodels, i.e., the structure does not offer any useful information for distinguishing the categories.
- Applicability** In the scope of this paper, we deal with categorization in the metamodel application domains. In particular, we considered a set of metamodels for domain language specifications classified according to their functionalities. Consequently, it is expected that the outcome might be different if other classification criteria rather than application domain were adopted. For instance, if we classify metamodels according to their complexity using metrics, such as number of classes, attributes, and hierarchies, we may obtain completely different outcomes. This suggests that the encoding performance is not a general property of the method only but also depends on the specific classification. However, this is a mere assumption, although an interesting one, which needs to be further investigated.
- Model classification** Also models could be automatically classified with respect to some characteristics derived from the language they conform to. For this reason, it is important to first classify metamodels and then models. In this respect, we plan to apply the proposed approach to classification of models. This aspect is linked to the metamodel concepts they conform, in fact it would be particularly useful to apply it to GPL, e.g., UML models. For instance, UML models could be classified according to the domain they represent: statecharts for IoT systems or statecharts for Web applications could be classified using our approach.
- Deeper and wider networks** Recently, several attempts have been made to boost up a neural network's performance by increasing various dimensions. However, what we can see from the experiments is that the extension of the given network does not bring any gain. As a consequence, we conclude that for the given dataset, it is reasonable to choose the most compact network configuration to maximize both accuracy and efficiency. We attempt to ascertain the rationale behind the decline in performance when the network is deeper or wider as follows. Deepening or widening allows a network to approximate the target function with increased non-linearity, thereby getting better feature representations. Nevertheless, it also adds up to the complexity of the network, which makes the network more difficult to optimize

and susceptible to overfitting. Adding more layers may lead to overfitting, for example, given a training metamodel, the model memorizes well the target class, and thus, it fails to generalize from data. Therefore, depending on the domain and the amount of input data, extending a network may probably bring benefits. For instance, our experiments give evidence that AURORA performs poorly on categories with a small number of metamodels; however, it substantially improves its performance on more populous categories. In this respect, the results suggest that data plays an essential role in obtaining a high prediction performance: To be concrete, a denser dataset is likely to be beneficial to the final classification.

- **Deployment** A question that may arise at any time is: “How can AURORA be deployed in practice?” As shown in Sect. 3.2, initially it is necessary to involve humans, e.g., a group of experienced modelers, to manually categorize a *large enough* dataset. Then, the dataset is fed to the system to train it, and the building phase might be done offline, e.g., overnight. Afterward, the deployment stage can be conducted to classify an arbitrary metamodel automatically. We can incrementally train the neural network, i.e., when more labeled metamodels are available over time, for example, by including more modelers for the manual labeling process; we continue refining the network without needing to train it from scratch. This is one of the main advantages of a neural network: In the first place, it is related to efficiency since there is no need to run the training again and again when there is a newly added metamodel, as it is the case with *Clustering* [10]. Notably, the system earns the *benefit of hindsight*: It can improve the prediction outcome once it has learned from additional training data. Furthermore, once the training data has been used, it can be discarded to give place for additional data, i.e., it is not necessary to store the *already-trained* data. Our proposed tool is a data-driven approach, and thus, its prediction performance is greatly influenced by the quality of data. To be concrete, if the data is not adequately labeled, i.e., there is label noise, it may cause a reduction in accuracy [30]. We consider the motivating examples in Sect. 2.2, where there are very similar metamodels; however, they have been labeled with entirely different names by humans. When such data is fed to train AURORA, then the final prediction would be negatively affected since it is challenging to teach a machine learning tool to name two metamodels similar in structure but with different labels [30,59]. This can be considered as a pitfall that modelers should avoid when deploying AURORA. In summary, we conclude that the labeled data quality accounts for variation in the classification outcome, which confirms the findings of an existing work [73].

- **Future developments** Despite the fact the various ML algorithms have been deployed in MDE in recent years, according to our careful observation, we see that the application of these techniques in the MDE domain is still in its infancy. In this sense, there is a need for a roadmap for the use of machine learning techniques in the MDE domain where different issues in MDE and the possible machine learning solutions are discussed. The recent achievement obtained for deep neural networks, which are based on a convolutional neural network (CNN), has unleashed many applications in different domains. We anticipate that deep neural networks such as recurrent neural networks (RNNs), long short-term memory neural networks (LSTM) can solve different tasks in MDE, such as recommendation of model repair or recommendation of metamodel specification. Among others, we envisage the usage of CNNs to classify metamodels. It is our hypothesis that the use of CNNs allows us to exploit a well-founded background related to many technical advantages, which have been proven through a series of work in image classification. We suppose that the topic is worth being investigated, and we consider this as future work.

7 Threats to validity

We distinguish between internal, construct, external, and conclusion validity as follows.

- **Internal validity** Such threats are the internal factors that could have influenced the final outcomes. One possible threat could be seen through the results obtained for the categories with a considerably low number of items (e.g., class *Tracker DSLs* or *Review sys. DSLs* in Fig. 9a). Such a threat is eased by denser groups (e.g., class *Database DSLs* and *State mac. DSLs* in Fig. 9b). Furthermore, the selection of the cutoff values may introduce bias, since c is just an even number throughout the evaluation, i.e., $k = \{2, 4, 6, 8\}$. Besides the experiments discussed in this paper, we conducted similar ones with odd numbers, i.e., $k = \{1, 3, 5, 7\}$. As far as we can see, there are no significant differences between the conclusions obtained from the new experiments; therefore, the results are omitted from the paper, for the sake of clarity.
- **External validity** The main threat to *external validity* concerns the generalizability of our findings, i.e., whether they would still be valid outside the scope of this study. The types of categorization have a profound impact on the performance of any machine learning algorithms. In this respect, we suppose that the generalizability of our approach only applies to the metamodel application domain level. For the other types of categorizations, it

may not hold any more, and thus, this needs further investigation. The external validity can also be affected by categories with a small number of metamodels. Thus, we attempt to moderate the threat by considering a set of 555 metamodels that are of different sizes and cover various categories. We suppose that the threat is partially moderated by the categories with a significant number of metamodels. However, it can be fully mitigated only if there is a considerably large as well as balanced dataset. In this respect, it is necessary to consider a bigger dataset with more metamodels for each category, and this is deferred as our future work.

- **Construct validity** They are related to the experimental settings presented in the paper, concerning the simulated setting used to evaluate the tool. The threat has been mitigated by applying tenfold cross-validation, attempting to simulate a real scenario of classification.
- **Conclusion validity** This is related to the exploited experiment methodology, i.e., if it is intrinsically related to the obtained outcome, or there are also other contributing factors. The quality metrics, i.e., accuracy, precision, recall, F_1 score, ROC, and AUC, may possibly cause a threat to conclusion validity. To deal with the issue, we employed the same metrics for evaluating all the experimental configurations.

8 Related work

Our main contributions are related to the deployment of ML techniques in MDE. Thus, in this section, we review some of the most notable related work on the adoption of ML algorithms in the context of MDE, as well as the categorization of models. Afterward, we list some significant applications of neural networks and associate our work with them.

8.1 Machine learning in MDE

The MDE community has made considerable progress in recent years as regards adopting various ML algorithms to solve different issues [5,19]. Kusmenko et al. [48] implement *MontiAnna*, a holistic deep learning modeling framework where deep neural networks can be designed, trained, and integrated into Component & Connector systems. Afterward, *MontiAnna* has been extended using reinforcement learning (RL) [34]. In the context of collaborative modeling, Barriga et al. [9] propose the adoption of an unsupervised and reinforcement learning (RL) approach to repairing broken models, which have been corrupted because of conflicting changes. The primary intent is to reach model repairing with human quality without requiring supervision potentially. Various studies [20,25] use ML techniques to automatically infer model transformation rules from sets of source and target

models. Metalearning is a technique that aims at using ML itself to automatically learn the most appropriate algorithms and parameters for an ML problem. Hartmann et al. [36] propose interleaving ML with domain modeling. In particular, they decompose machine learning into reusable, chainable, and independently computable small learning units. These microlearning units are modeled together with and at the same level as the domain concepts.

Breuker [18] reviews the main modeling languages used in ML as well as inference algorithms and corresponding software implementations. This work aims to explore the opportunities of defining a DSML for probabilistic modeling. Similarly, ML has been collaterally treated [70], where the authors present an extension of the modeling language and tool support of ThingML—an open-source modeling tool for IoT/CPS—to address ML needs for IoT applications. To allow developers to design solutions that solve machine learning-based problems by automatically generating code for other technologies as a transparent bridge, a language, and a technology-independent development environment were introduced [32]. A similar tool called OptiML has been proposed [76], attempting to bridge the gap between ML algorithms and heterogeneous hardware to provide a productive working environment.

Cabot et al. [23] discuss how Model-driven software engineering can benefit from the adoption of machine learning techniques and in general of cognification, which is the “*application of knowledge to boost the performance and impact of a process.*” Modeling bots, model inferences, and real-time model reviewers are only some of the examples of new functionalities that can be introduced thanks to cognification.

Based on the experience we have acquired by developing recommender systems in software engineering [56,57], we see a great potential of applying similar recommendation techniques for the MDE domain. We already developed a system for providing developers with suitable packages and classes when they create metamodels. As future work, we plan to further improve the proposed approach by employing long short-term memory (LSTM) neural networks.

In summary, it can be seen that learning algorithms have been successfully applied to tackle different issues in the MDE context. Nevertheless, the adoption of cognitive techniques in this domain seems not to have received proper attention being commensurate with its potential. In this respect, we believe that the introduction of AURORA is expected to advance the existing approaches in the classification of metamodels.

8.2 Categorization of metamodels

Classification and clustering techniques have been employed to categorize metamodels for a diversity of purposes, not

excluded reuse [6]. In particular, an agglomerative hierarchical clustering technique has been employed to organize metamodels in repositories automatically. Hierarchical clustering techniques are the main engine to perform metamodel comparison, analysis, and visualization [6]. In these approaches, metamodel named elements are converted into lexical terms and represented as a vector space model. This encoding enables the usage of hierarchical clustering algorithms to group similar objects and serve it as a dendrogram. The approaches mentioned above share the same intent of this paper even though they are based on unsupervised learning techniques, whereas our work follows the supervised learning strategy.

Lopez et al. [51] present a domain-oriented approach for software requirements reuse. First, requirements captured from semiformal diagrams are injected into models, which are then analyzed to check for any probable quality issues. Finally, a mechanism to cluster the requirements is applied to allow the domain requirements patterns to be identified and reused. Reuser [67] is a tool used to automatically retrieve related UML artifacts and propose them to the modeler. A search algorithm is applied to find similar artifacts by classifying them into a concept lattice. Then it compares the input artifact query against the concept lattice, to retrieve the closest set of matching artifacts.

Jiang et al. [41] discuss different characteristics of UML metamodel extension mechanisms according to a four-level classification. Each level of metamodel extension has different features in (contrasting) aspects such as readability, expression capability, use scope, and tool support. The paper aims at providing modelers with a reliable theoretical base for selecting the right level to extend the metamodel to find the right trade-off of the above aspects. A classification model for UML stereotypes has been developed in [13], where the artifacts are analyzed according to their potential to alter the syntax and semantics of the base language to be able to control their application in practice. Feature-based criteria for classifying approaches according to the type of spatial relation involved have been widely discussed [16]. In contrast to our work, this is a taxonomic approach that manually identifies categories for visual languages based on their structure (and neglecting concrete syntax and semantics). The taxonomy promoted by Burnett et al. [21] only considers visual programming languages, mainly from the paradigm they express. This is a brief (and non-exhaustive) survey of purpose-oriented classification approaches, and further secondary studies are available. However, it goes beyond the scope of our paper to discuss the *why* classifications, as well as their enabling techniques, are necessary.

Most of the approaches that are agnostic of the classification purpose are based on lexical and structural information encoded in the metamodels. Such knowledge can be leveraged in a more semantic-aware analysis when the artifacts are

constrained to specific categories. For instance, an approach [51] focuses on requirement diagrams where the various relationships within a diagram have particular meanings that help achieve better classification performance. More recently, generic clustering techniques have been extensively investigated [75]. The work presents a tool that supports the decomposition of a metamodel into clusters of model elements. The main goal is to improve model maintainability by facilitating model comprehension. In particular, modularity is achieved by decomposing the input model into a set of sub-models. The main feature that makes this study different from others is that it works by splitting a single model into clusters, and a post-processing phase is introduced: Users are allowed to add more clusters or to rename, remove, as well as to regroup existing ones. Indeed, the identified clusters result unnamed in contrast to our approach, and the representation is only grouping similar concepts in the same containers.

8.3 Applications of neural networks

The ability to learn from labeled data underpins the main strength of neural networks, making them a well-defined technique in machine learning. To name just a few, pattern recognitions [14], forecasting [80], and classification [4,26] are the main application domains of neural networks. For classification, neural networks have demonstrated their suitability in various applications. Augusteijn *et al.* [4] investigate two different types of neural networks for classification, i.e., back-propagation and probabilistic neural network, and find out that only the latter is suitable for the detection of novel patterns. Convolutional neural networks (CNNs) were originally designed to work with images, and thus, they have been widely used to solve various recognition tasks. To name but a few, CNNs have been applied to detect olive fruits [33], to classify images [46]. The success of CNNs in computer vision is a source of inspiration for applications in other domains. For instance, CNNs have been exploited to classify DNA sequences [3], or to process natural language [12]. We envisage the adoption of CNNs to solve different issues in MDE, such as metamodel specification or classification [60].

Through a careful observation, we realized that AURORA is the first application of neural networks to classify metamodels. Furthermore, it is among the initial attempts to deploy neural networks into a completely new domain, i.e., MDE [71]. Our work distinguishes itself from existing studies in MDE as follows: (i) We employed a well-founded ML technique to automate the classification of metamodels; (ii) the tool can automatically learn from labeled metamodels to deal with unlabeled metamodels, thereby applying to real metamodel categorization scenarios; and (iii) last but not least, our proposed paradigm is expected to pave the way for further adoptions of advanced machine learn-

ing algorithms, not excluding deep learning [35,62] in MDE.

9 Conclusions

We proposed AURORA as the first attempt to automatize the classification of metamodels, exploiting a well-founded machine learning technique. AURORA is built on top of a feed-forward neural network utilizing two different loss functions. Thus, it can learn from labeled data and classify unknown metamodels. The prototype is handy since it considerably reduces the effort required to classify metamodels by hands, thus automatizing the categorization process.

AURORA has been evaluated on a manually classified dataset using the tenfold cross-validation technique. The experimental results demonstrate that the tool is capable of categorizing the test data with high accuracy. We also see that AURORA outperforms gradient boosted decision tree and support vector machines, two well-defined machine learning algorithms.

As future work, we plan to investigate the adoption of different types of networks, e.g., convolutional neural networks, or recurrent neural networks to classify various modeling artifacts, e.g., models, code generators, or to recommend model transformation, to conceive recommender systems being able to support modelers with a wide range of modeling activities.

Acknowledgements The research described in this paper has been carried out as part of the CROSSMINER Project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant 732223. We thank the anonymous reviewers for their valuable comments and suggestions that helped us improve the paper.

Funding Open access funding provided by Università degli Studi dell'Aquila within the CRUI-CARE Agreement.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Agarap, A.F.: Deep learning using rectified linear units (relu). *arxiv:1803.08375Comment*: 7 pages, 11 figures, 9 tables (2018)
2. Aly, M.: Survey on multiclass classification methods. *Neural Netw.* **19**, 1–9 (2005)
3. Aoki, G., Sakakibara, Y.: Convolutional neural networks for classification of alignments of non-coding RNA sequences. *Bioinformatics* **34**(13), i237–i244 (2018)
4. Augusteijn, M.F., Folkert, B.A.: Neural network classification and novelty detection. *Int. J. Remote Sens.* **23**(14), 2891–2902 (2002)
5. Babur, Ö., Chaudron, M.R., Cleophas, L., Di Ruscio, D., Kolovos, D.: Ammore 2018: first international workshop on analytics and mining of model repositories. *CEUR Workshop Proc.* **2245**, 778–779 (2018)
6. Babur, O., Cleophas, Z.: Using n-grams for the automated clustering of structural models. In: Steffen, B., Baier, C., van den Brand, M., Eder, J., Hinchey, M., Margaria, T. (eds.) *SOFSEM 2017: Theory and Practice of Computer Science*, pp. 510–524. Springer International Publishing, Cham (2017)
7. Babur, Ö., Cleophas, L., van den Brand, M.: Hierarchical clustering of metamodels for comparative analysis and visualization. In: *Modelling Foundations and Applications. Lecture Notes in Computer Science*, pp. 3–18. Springer, Germany (2016)
8. Babur, Ö., Cleophas, L., van den Brand, M., Tekinerdogan, B., Aksit, M.: Models, more models, and then a lot more. In: *Federation of International Conferences on Software Technologies: Applications and Foundations*, pp. 129–135. Springer, New York (2017)
9. Barriga, A., Rutle, A., Heldal, R.: Automatic model repair using reinforcement learning. In: *Proceedings of Workshops co-located with MODELS 2018. Copenhagen, Denmark*, p 781–786, (2018)
10. Basciani, F., Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Automated clustering of metamodel repositories. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) *Advanced Information Systems Engineering*, pp. 342–358. Springer International Publishing, Cham (2016)
11. Basciani, F., Rocco, J. D., Ruscio, D. D., Salle, A. D., Iovino, L., Pierantonio, A.: MDEForge: an extensible web-based modeling platform. In: *Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud, Cloud-MDE@MoDELS 2014, Valencia, Spain, September 30, 2014.*, pp 66–75, (2014)
12. Belinkov, Y., Glass, J.: Analysis methods in neural language processing: a survey. *Trans. Assoc. Comput. Linguist.* **7**, 49–72 (2019)
13. Berner, S., Glinz, M., Joos, S.: A classification of stereotypes for object-oriented modeling languages. In: France, R., Rumpe, B. (eds.) *UML'99- The Unified Modeling Language*, pp. 249–264. Springer, Berlin, Heidelberg (1999)
14. Bishop, C.M.: *Neural Networks for Pattern Recognition*. Oxford University Press Inc, New York, NY, USA (1995)
15. Blum, A.: *Neural Networks in C++: An Object-oriented Framework for Building Connectionist Systems*. Wiley, New York (1992)
16. Bottoni, P., Grau, A.: A suite of metamodels as a basis for a classification of visual languages. In: *2004 IEEE Symposium on Visual Languages—Human Centric Computing*, 83–90, Sep. (2004)
17. Bottou, L.: Stochastic gradient learning in neural networks. In: *Proceedings of Neuro-Nîmes. EC2*, (1991)
18. Breuker, D.: Towards model-driven engineering for big data analytics—an exploratory analysis of domain-specific languages for machine learning. In: *2014 47th Hawaii International Conference on System Sciences*, pp 758–767, (2014)
19. Burgueño, L., Burdusel, A., Gérard, S., Wimmer, M.: *MDE Intelligence19: First international workshop on Artificial Intelligence and Model-driven Engineering*. IEEE, (2019)
20. Burgueño, L., Cabot, J., Gérard, S.: An lstm-based neural network architecture for model transformations. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp 294–299, (2019)

21. Burnett, M.M., Baker, M.J.: A classification system for visual programming languages. *J. Vis. Lang. Comput.* **5**(3), 287–300 (1994)
22. Byun, H., Lee, S.-W.: Applications of support vector machines for pattern recognition: a survey. In: Lee, S.-W., Verri, A. (eds.) *Pattern Recognition with Support Vector Machines*, pp. 213–236. Springer, Berlin, Heidelberg (2002)
23. Cabot, J., Clarisó, R., Brambilla, M., Gérard, S.: Cognifying model-driven software engineering. In: Seidl, M., Zschaler, S. (eds.) *Software Technologies: Applications and Foundations*, pp. 154–160. Springer International Publishing, Cham (2018)
24. Devlin, J., Chang, J., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics (2019)
25. Dolques, X., Huchard, M., Nebut, C., Reitz, P.: Learning transformation rules from transformation examples: an approach based on relational concept analysis. In: *2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops*, pp 27–32, (2010)
26. Duong, L.T., Nguyen, P.T., Di Sipio, C., Di Ruscio, D.: Automated fruit recognition using efficientnet and mixnet. *Comput. Electron. Agric.* **171**, 105326 (2020)
27. Fawcett, T.: An introduction to roc analysis. *Pattern Recogn. Lett.* **27**(8), 861–874 (2006)
28. France, R., Bieman, J., Cheng, B. H.: Repository for model driven development (remodd). In: *International Conference on Model Driven Engineering Languages and Systems*. Springer, pp 311–317 (2006)
29. France, R., Bieman, J., Cheng, J.: Repository for model driven development (remodd). In: Kühne, T. (ed.) *Models in Software Engineering*, pp. 311–317. Springer, Berlin, Heidelberg (2007)
30. Frenay, B., Verleysen, M.: Classification in the presence of label noise: a survey. *IEEE Trans. Neural Netw. Learn. Syst.* **25**(5), 845–869 (2014)
31. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. *Ann. Stat.* **29**, 1189–1232 (2000)
32. García-Díaz, V., Espada, J.P., Bustelo, B.C.P.G., Lovelle, J.M.C.: Towards a standard-based domain-specific platform to solve machine learning-based problems. *IJIMAI* **3**(5), 6–12 (2015)
33. Gatica, G., Best, G., Ceroni, J., Lefranc, G.: Olive fruits recognition using neural networks. *Procedia Computer Science*. In: *First International Conference on Information Technology and Quantitative Management* **17**, 412–419, (2013)
34. Gatto, N., Kusmenko, E., Rumpe, B.: Modeling deep reinforcement learning based architectures for cyber-physical systems. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pp 196–202. IEEE, (2019)
35. Goodfellow, I., Bengio, Y., Courville, A.: *Deep Learning*. The MIT Press, Cambridge (2016)
36. Hartmann, T., Moawad, A., Fouquet, F., Le Traon, Y.: The next evolution of mde: a seamless integration of machine learning into domain modeling. *Softw. Syst. Model.* **18**(2), 1285–1304 (2019)
37. Hastie, T., Tibshirani, R., Friedman, J.: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics, New York (2009)
38. Hein, C., Ritter, T., Wagner, M.: Model-driven tool integration with modelbus. In: *Workshop Future Trends of Model-Driven Development*, pages 50–52, (2009)
39. Isakowitz, T., Kauffman, R.J.: Supporting search for reusable software objects. *IEEE Trans. Softw. Eng.* **22**(6), 407–423 (1996)
40. Jahrer, M., Tösch, A., Legenstein, R. A.: Combining predictions for accurate recommender systems. In: B. Rao, B. Krishnapuram, A. Tomkins, and Q. Yang, (eds), *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, USA, July 25–28, 2010, pp 693–702. ACM, (2010)
41. Jiang, Y., Shao, W., Zhang, L., Ma, Z., Meng, X., Ma, H.: On the classification of uml’s meta model extension mechanism. In: Baar, T., Strohmeier, A., Moreira, A., Mellor, S.J. (eds.) *The Unified Modeling Language Modeling Languages and Applications*, pp. 54–68. Springer, Berlin, Heidelberg (2004)
42. Karasneh, B., Chaudron, M. R.: Online img2uml repository: an online repository for uml models. In: *EESSMOD@ MODELS*, 61–66, (2013)
43. Kogel, M., Helming, J.: EMFStore: a model repository for EMF models. In: *2010 ACM/IEEE 32nd International Conference on Software Engineering*, **2**, 307–308. IEEE, (2010)
44. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *14th International Joint Conference on Artificial Intelligence*, pp 1137–1143 San Francisco. Morgan Kaufmann Publishers Inc (1995)
45. Kotsiantis, S.B., Zaharakis, I., Pintelas, P.: Supervised machine learning: a review of classification techniques. *Emerg. Artif. Intell. Appl. Comput. Eng.* **160**(1), 3–24 (2007)
46. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) *Advances in Neural Information Processing Systems*, pp. 1097–1105. Curran Associates Inc, New York (2012)
47. Kuhn, G. E., Ducasse, S., Girba, T.: Enriching reverse engineering with semantic clustering. In: *12th Working Conference on Reverse Engineering (WCRE’05)*, pages 10 pp.–142, (2005)
48. Kusmenko, E., Nickels, S., Pavlitskaya, S., Rumpe, B., Timmermanns, T.: Modeling and training of neural processing systems. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp 283–293. IEEE, (2019)
49. Latinne, P., Debeir, O., Decaestecker, C.: Combining different methods and numbers of weak decision trees. *Pattern Anal. Appl.* **5**(2), 201–209 (2002)
50. Lopes, C. V., Maj, P., Martins, P., Saini, V., Yang, D., Zitny, J., Sajjani, H., Vitek, J.: Déjàvu: a map of code duplicates on github. In: *Proceedings of the ACM on Programming Languages*, **1**(OOPSLA):1–28, (2017)
51. Lopez, O., Laguna, M. A., García, F. J.: Reuse based analysis and clustering of requirements diagrams. In: *Pre-Proceedings of the Eighth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ’02)*, pp. 71–82, (2002)
52. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q. (eds.) *Advances in Neural Information Processing Systems*, pp. 3111–3119. Curran Associates Inc, New York (2013)
53. Mitra, P., Uma Shankar, B., Pal, S.K.: Segmentation of multispectral remote sensing images using active support vector machines. *Pattern Recognit. Lett.* **25**(9), 1067–1074 (2004)
54. Mokaddem, C. e., Sahraoui, H., Syriani, E.: Recommending model refactoring rules from refactoring examples. In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS ’18*, pp 257–266, New York, NY, USA, ACM (2018)
55. Nasr, G. E., Badr, E. A., Joun, C.: Cross entropy error function in neural networks: forecasting gasoline demand. In: *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*, May 14–16, 2002, Pensacola Beach, Florida, USA, pp 381–384, (2002)

56. Nguyen, P.T., Di Rocco, J., Di Ruscio, D., Di Penta, M.: Cross-Rec: supporting software developers by recommending third-party libraries. *J. Syst. Softw.* **161**, 110460 (2019)
57. Nguyen, P. T., Di Rocco, J., Di Ruscio, D., Ochoa, L., Degueule, T., Di Penta, M.: FOCUS: a Recommender System for Mining API Function Calls and Usage Patterns. In: *Proceedings of the 41st International Conference on Software Engineering, ICSE '19*, pp 1050–1060, Piscataway, NJ, USA. IEEE Press (2019)
58. Nguyen, P. T., Di Rocco, J., Di Ruscio, D., Pierantonio, A., Iovino, L.: Automated classification of metamodel repositories: a machine learning approach. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp 272–282, (2019)
59. Nguyen, L., Di Rocco, J., Di Sipio, C., Di Ruscio, D., Di Penta, M.: Recommending api function calls and code snippets to support software development. *IEEE Trans. Softw. Eng.* **1**, (2021)
60. Nguyen, P.T., Di Ruscio, D., Pierantonio, A., Di Rocco, J., Iovino, L.: Convolutional neural networks for enhanced classification mechanisms of metamodels. *J. Syst. Softw.* **172**, 110860 (2020)
61. Nguyen, P.T., Eckert, K., Ragone, A., Di Noia, T.: Modification to K-Medoids and CLARA for effective document clustering. In: *Foundations of Intelligent Systems*, pp. 481–491. Springer International Publishing, Cham (2017)
62. Nielsen, M. A.: *Neural networks and deep learning*, (2018)
63. Nielsen, T. D., Rouillard, T., Makarov, N.: A semantic search capability for a grid model repository. In: *2019 IEEE Power Energy Society General Meeting (PESGM)*, pp 1–5, (2019)
64. Önder Babur. A labeled Ecore metamodel dataset for domain clustering, Mar. (2019)
65. Phillips, P.: Support vector machines applied to face recognition. In: *Kearns, M., Solla, S., Cohn, D. (eds.) Advances in Neural Information Processing Systems*, pp. 803–809. MIT Press, Cambridge (1999)
66. Portugal, I., Alencar, P. S. C., Cowan, D. D.: The use of machine learning algorithms in recommender systems: a systematic review. *CoRR*, abs/1511.05263, (2015)
67. Robinson, W.N., Woo, H.G.: Finding reusable uml sequence diagrams automatically. *IEEE Softw.* **21**(5), 60–67 (2004)
68. Rojas, R.: *Neural Networks: A Systematic Introduction*. Springer-Verlag, Berlin, Heidelberg (1996)
69. Rokach, L., Maimon, O.: Data mining for improving the quality of manufacturing: a feature set decomposition approach. *J. Intell. Manuf.* **17**(3), 285–299 (2006)
70. Moin, A., Rössler, S., Günnemann, S.: Thingml+: Augmenting model-driven software engineering for the internet of things with machine learning. In: *R. Hebig and T. Berger (eds.) Proceedings of Workshops Co-Located with MODELS 2018, Copenhagen, Denmark, October, 14, 2018, volume 2245 of CEUR Workshop Proceedings*, pp 521–523. CEUR-WS.org (2018)
71. Schmidt, D.C.: Guest editor's introduction: model-driven engineering. *Computer* **39**(2), 25–31 (2006)
72. Schmidt, D.C.: Guest editor's introduction: model-driven engineering. *Computer* **39**(2), 25–31 (2006)
73. Sessions, V., Valtorta, M.: The effects of data quality on machine learning algorithms. In: *J. R. Talburt, E. M. Pierce, N. Wu, and T. Campbell (eds.), Proceedings of the 11th International Conference on Information Quality, MIT, Cambridge, MA, USA, November 10–12, 2006*, pages 485–498. MIT, (2006)
74. Steinwart, I., Christmann, A.: *Support Vector Machines*, 1st edn. Springer Publishing Company, Incorporated, New York (2008)
75. Strüder, D., Selter, M., Taentzer, G.: Tool support for clustering large meta-models. In: *Proceedings of the Workshop on Scalability in Model Driven Engineering, BigMDE '13*, pages 7:1–7:4, New York, NY, USA. ACM. (2013)
76. Sujeeth, A., Lee, H., Brown, K., Rompf, T., Chafi, H., Wu, M., Atreya, A., Odersky, M., Olukotun, K.: OptiML: an implicitly parallel domain-specific language for machine learning. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 609–616, (2011)
77. Tang, B., Spiteri, R., Milios, E., Zhang, R., Wang, S., Tougas, J., Shafiei, M.: Document representation and dimension reduction for text clustering. In: *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*, pages 770–779, Los Alamitos, CA, USA. IEEE Computer Society (2007)
78. Vapnik, V.: *The nature of statistical learning theory*. Springer Science & Business Media, New York (2013)
79. Whittle, J., Hutchinson, J., Rouncefield, M.: The state of practice in model-driven engineering. *IEEE Softw.* **31**(3), 79–85 (2013)
80. Zhang, G., Eddy Patuwo, B., Hu, M.Y.: Forecasting with artificial neural networks: the state of the art. *Int. J. Forecast.* **14**(1), 35–62 (1998)
81. Zhang, M., Zhou, Z.: Multi-label neural networks with applications to functional genomics and text categorization. *IEEE Trans. Knowl. Data Eng.* **18**(10), 1338–1351 (2006)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Phuong T. Nguyen obtained a PhD in Computer Science from the University of Jena, Germany. Since the graduation, he has worked as a university teaching and research assistant in Vietnam and Italy. Phuong is now with Università degli Studi dell'Aquila, Italy, as a postdoctoral researcher. His research interests include computer networks, semantic web, recommender systems, and machine learning. Recently, he has been working to develop recommender systems in software engineering

for mining open source code repositories.

Juri Di Rocco is a postdoctoral researcher at the Department of Information Engineering Computer Science and Mathematics, Università degli Studi dell'Aquila, Italy. Previously, he obtained the PhD degree from Università degli Studi dell'Aquila in the MDE Group research team with Alfonso Pierantonio and Davide Di Ruscio. He is interested in all aspects of software language engineering. Main research interests are related to several aspects of model-driven engineering (MDE) including

domain-specific modeling languages, model transformation, model differencing, modeling repositories, and mining techniques.

Ludovico Iovino is currently Assistant Professor at the GSSI-Gran Sasso Science Institute, L'Aquila—in the Computer Science scientific area. His interests include model-driven engineering (MDE), model transformations, metamodel evolution, code generation, and software quality evaluation. Currently, he is working on model-based artifacts and issues related to the metamodel evolution problem. He has been included in program committees of numerous conferences and in

the local organization of the STAF conference 2015 and models and evolution workshop from 2 years. He is a part of different academic projects related to model repositories, model migration tools, and Eclipse Plugins.

Davide Di Ruscio is an Associate Professor at the DISIM—University of L'Aquila. His main research interests are related to several aspects of software engineering, open-source software, and model-driven engineering (MDE) including domain-specific modeling languages, model transformation, model differencing, coupled evolution, and recommendation systems. He has published more than 160 papers in various journals, conferences, and workshops on such topics. He is a

member of the steering committee of the International Conference on Model Transformation (ICMT), the Software Language Engineering (SLE) conference, the Seminar Series on Advanced Techniques & Tools for Software Evolution (SATTOSE), the Workshop on Modelling in Software Engineering at ICSE (MiSE), and the International Workshop on Robotics Software Engineering (RoSE). He is in the editorial board of the International Journal on Software and Systems Modeling (SoSyM), IEEE Software, the Journal of Object Technology, and the IET Software journal. More information is available at <http://people.disim.univaq.it/diruscio>.

Alfonso Pierantonio is Professor in Computer Science at Università degli Studi dell'Aquila, Italy. His current research interests include model-driven engineering with a specific emphasis on model management analytics and co-evolution. Alfonso is involved in several scientific projects, including EU H2020 Lowcomote and EU H2020 Typhon; moreover, he is the principal co-investigator of the Italian Railways funded project ERMES. He has been general chair of STAF 2015, PC chair of

ECMFA 2018, and designated general chair of MoDELS 2022 besides being on the organizing, steering, and program committees of many international conferences. Alfonso is the Editor-in-Chief of the Journal of Object Technology (JOT) and a member of the editorial board of the Journal on Software and Systems Modeling (SoSyM). He has published more than 140 papers in international journals, conferences, and workshops.