Automated Classification of Metamodel Repositories: A Machine Learning Approach

Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, Alfonso Pierantonio
Università degli Studi dell'Aquila, L'Aquila, Italy
{firstname.lastname}@univaq.it

Ludovico Iovino
Gran Sasso Science Institute
L'Aquila, Italy
ludovico.iovino@gssi.infn.it

***Abstract*—Manual classification methods of metamodel repositories require highly trained personnel and the results are usually influenced by subjectivity of human perception. Therefore, automated metamodel classification is very desirable and stringent. In this work, Machine Learning techniques have been employed for metamodel automated classification. In particular, a tool implementing a feed-forward neural network is introduced to classify metamodels. An experimental evaluation over a dataset of 555 metamodels demonstrates that the technique permits to learn from manually classified data and effectively categorize incoming unlabeled data with a considerably high prediction rate: the best performance comprehends 95.40% as success rate, 0.945 as precision, 0.938 as recall, and 0.942 as F_1 score.**

I. Introduction

The sheer complexity of today's software has leveraged the employment of model-driven techniques throughout the development process [49]. Models are pervasively used to enhance the comprehension, productivity, quality, and ease of keeping pace with technology advance [59]. Moreover, the emerging trend of globalization strongly characterizes the need for modeling repositories (e.g., see [26], [30], [32], [37] to mention a few). Repository-based environments typically provide software developers with useful support to locate software artifacts for reuse [27]. However, locating relevant information in a repository presents interesting human and technical challenges because it requires the artifacts in the repository to be precisely classified. Often this task is performed by manually annotating the artifacts with fine-grained metadata, an intrinsically time consuming and error-prone operation that might lead to accuracy problems [11].

In a recent work [5], an automated technique has been proposed to categorize metamodel repositories. Based on an agglomerative hierarchical clustering algorithm [28], the technique can detect *similar* metamodels according to given similarity metrics (some of them specifically devised for metamodels). Interestingly, the experimental validation demonstrated how structural and lexical similarities can help classify metamodels according to the application domain they formalize. To date, many existing studies exploit clustering techniques to group similar metamodels, e.g., [3], [5]. Nevertheless, the performance of clustering techniques is a critical aspect that strictly depends on the chosen similarity measure. Structural similarities, which generally offer better accuracy, have poorer performance when compared with traditional lexical techniques, including cosine similarity [7] and dice coefficient [20] that in contrast have limited accuracy [5].

Machine Learning (ML) algorithms attempt to simulate humans' learning activities, aiming to acquire real-world knowledge autonomously [43]. In this way, ML systems are capable of generalizing from concrete examples, without needing to be manually programmed [22], [60]. Thanks to this characteristic, ML algorithms find their applications in various domains, including web search by learning from a user's long-term search history [50], recommender systems [43], and self-driving vehicles [52]. Despite the high potential, the application of ML algorithms in MDE has gained a moderate growth.

In this paper, to overcome the limitations and ease the burden of manual categorization of metamodels, we propose AURORA, a tool for the AUtomated classification of metamodel RepOsitories using a neuRAl network. Based on a well-established technique, it learns from a training set consisting of labeled metamodels and effectively classifies incoming unlabeled ones. An empirical evaluation of dataset of 555 metamodels using ten-fold cross-validation shows that AURORA predicts a label for an arbitrary metamodel with considerably high accuracy. The main contributions of our paper are *(i)* we build a supervised classifier for categorizing metamodels based on a neural network, and *(ii)* an experimental validation, which has been performed on a dataset of metamodels collected from GitHub and manually categorized, revealed high success rates and accuracy.

Outline of the paper. This paper is organized as follows. Section II introduces background notions and motivates this work. Our supervised classifier for metamodels, AURORA, is introduced in Section III. The evaluation is presented in Section IV and the main results are analyzed in Section V. Section VI discusses the threats to validity. We present some related work and conclude the paper in Section VII and Section VIII, respectively.

II. Background and Motivations

In the context of data mining, classification and clusterization are some of the critical operations that are used to dig deep into available data for gaining knowledge and for identifying repetitive patterns. Classification is a *supervised learning* technique, which relies on the existence of predefined classes of objects and aims at understanding to which class a new item belongs [34]. Clusterization is an *unsupervised learning* technique, which aims at grouping a set of objects in different clusters with respect to some similarity function [28]. Thus, the number of classes is not known ex-ante as in the case of unsupervised learning. Both classification and clusterization have been widely used in different fields such as medicine, biology, physics, geology, and many others.

In software engineering, such techniques have been employed, e.g., in the context of reverse engineering and software maintenance for categorizing software artifacts [36], [40]. In the context of open source software, categorizing available projects by understanding their similarities allows for reusing source code or learning how existing and similar complex systems have been developed [41], [48], thereby improving software quality [51]. In this respect, recommender systems have been conceived to provide developers with suitable recommendations, which can give some guidance to undertake the particular development task at hand [19], [48], [58].

In [17] *Cabot et al.* 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.

In [3], [5], the authors propose the application of clustering techniques to automatically organize reusable metamodels. In [5] modelers are provided with structured overviews of available metamodels, which instead are typically shown as mere lists of stored elements, and which are consequently challenging to browse. Using the approach in [5], metamodels are automatically organized in views consisting of connected graphs. Each graph represents identified clusters, and the thickness of the edges is proportional to the similarity of the connected metamodels represented as nodes in the graph.

The main challenges that in our opinion hamper the adoption in practice of approaches based on hierarchical clustering to categorize metamodels (like [3], [5]) are the following:

- *Timing performance*: the approach in [5] has been evaluated by considering a dataset consisting of ≈300 metamodels retrieved from the Ecore Zoo[1]. The clusterization procedure of the whole dataset takes ≈4 hours. This can be limiting especially when the addition of new metamodels is frequent, and thus the clustering procedure needs to be applied again on the new dataset;
- *The number of identified clusters can be too high for enabling effective search*: even though the approach in [5] is able to automatically catogorize metamodels according to their content and structure, the unsupervised learning technique produces ≈200 clusters out of the considered metamodels in the Ecore Zoo. Thus, modelers can still have some difficulties in grasping the application domains formalized by the available metamodels and getting a bird's eye view of them.

[1]ATLAS Ecore Zoo: http://www.emn.fr/z-info/atlanmod/index.php/Zoos

To confront such issues, in the next section we propose an efficient supervised learning approach based on a neural network, which can categorize metamodels with respect to already defined categories, e.g., metamodels for data modeling, for specifying system requirements, and to support the specification of system behaviors. By considering a learning set consisting of already classified metamodels with respect to predefined labels, the approach can automatically classify unlabeled metamodels.

III. AURORA: A Supervised Classifier for Metamodel Repositories

As already discussed, in supervised classification metamodels can be assigned to specific categories to facilitate the search process. The labeling is performed manually, e.g., when developers create or upload a project, they specify one or more categories to the contained metamodel. Later on, these categories serve as a means to help other developers narrow down the search scope and efficiently approach the project. Conventional wisdom suggests that the prescribed information related to repositories and their categories is meaningful: it reflects the perception of humans towards the relationship between metamodels and categories. We hypothesize that metamodels' features such as classes, attributes and their labels can be exploited to automatically group metamodels into categories. In other words, it is reasonable to categorize metamodels by simulating humans' cognition towards the metamodels-categories relationship, using the available data.

To this end, we come up with the adoption of a neural network. The ability of learning from labeled data underpins the main strength of neural networks, making them a well-founded technique in Machine Learning [35]. To name just a few, pattern recognitions [10], forecasting [61], and classification [2], [45] are their main application domains. In this work, we exploit a neural network to build AURORA. As a base for further presentations, Section III-A recalls the key concepts and notations related to feed-forward neural networks, which mainly come from [42] and [56]. Afterwards, Section III-B introduces our proposed architecture.

A. Feed-forward Neural Networks

The atomic element of a neural network is called *perceptron*. Each perceptron receives a set of inputs and produces an output. For each input x_i, there is a weight ω_i associated with it. A bias b is attached to allow for more flexibility in adjusting the output. An activation function f is used to compute the output, given an input. Fig. 1 depicts a simplified perceptron with three inputs, and the corresponding output is: $f(\sum_{j=1}^{3} \omega_j x_j + b)$.

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, with an exception of the output layer. The edges of the network convey information in a unique direction [46], e.g., from left to right. Depending on the purpose, the number of neurons in a hidden layer as well as the number

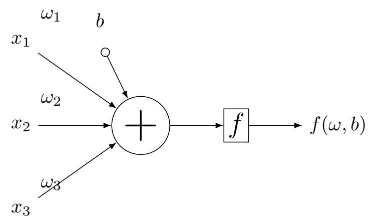


Fig. 1. A perceptron.

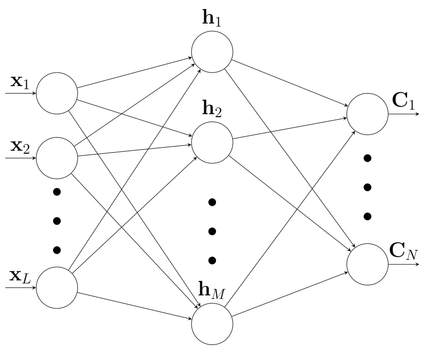


Fig. 2. A three-layer neural network.

of hidden layers may vary. We consider a concrete example as depicted in Fig. 2. For a clear presentation, the constituent weights, biases as well as the activation functions are omitted. The neural network is made of three layers: the first layer is called input and it consists of L neurons, corresponding to the number of input features, i.e., $X = (x_1, x_2, ..., x_L)$ [61]. The second one is the hidden layer with M perceptrons, i.e., $H = (h_1, h_2, ..., h_M)$. The output layer consists of N perceptrons, corresponding to N output categories, i.e., $C = (C_1, C_2, .., C_N)$. In this work, the *sigmoid function* is used as the activation function as it has been widely exploited in different studies [42].

Algorithm 1 The learning process

```
1: procedure LEARNING($training\_data$,$epochs$)
2:     $e = 0$;
3:     $model = initialization()$;
4:     while e < epochs do
5:         $training\_data = shuffle(training\_data)$;
6:         $X, y = split(training\_data)$;
7:         $\hat{y} = predict(X, model)$;
8:         $error = calculate\_error(y, \hat{y})$;
9:         $model = refine\_model(model, error)$;
10:        $e++$;
11:    end while
12:    return $model$
13: end procedure
```

The learning process is illustrated in the pseudo code in Listing 1. In this listing, *epoch* is one round of learning performed on the training data, i.e., introducing all the input vectors [6]. In the beginning of each epoch, the training set is shuffled (Line 5) with the aim of randomizing the input data, thus avoiding the problem of *being stuck in local minima* [42]. For each epoch, only some mini-batches of the training inputs are selected, and the training is done only on these samples. The network is fed with input data X and labels y obtained after being split (Line 6), which is going to be detailed in Section IV-B.

The predicted values $\hat{y}$ (Line 7) are the results of running on the training data and they are computed using the following formula.

$$\hat{y} = f(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+exp(-\sum_j \omega_j x_j - b)} \quad (1)$$

The difference between the real category and the predicted value is called error (Line 8) and computed as given below.

$$E(w,b) = \frac{1}{2L}\sum_x \|y(x) - (\omega.x + b)\|^2 \quad (2)$$

The error converges to zero when $\hat{y} \approx y$, i.e., the predictions match with the real labels. The final aim of the learning process is to find a function that best maps the input data with the output data. In other words, we minimize the error function $E(w,b)$ by choosing a suitable set of weights and biases [10], and this is done by applying Stochastic Gradient Descent (SGD) as follows [14], [18], [42]. The model performs prediction on the training data, then the error between the actual outcome and the predicted results is used to adjust the model to minimize errors (Line 9). The outcome of the training phase is `model` with weights and biases (Line 12) that can be used to approximately produce the outputs from the input data.

Learning is essentially the process of refining the constituent weights and biases so as to produce the corresponding output y, given a specific input X.

Listing 2 depicts the testing process. In contrast to learning, this phase is much simpler 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 final outcome is the predicted labels for the input data.

Algorithm 2 The testing process

```
1: procedure TESTING($model$,$testing\_data$)
2:     $X = testing\_data$;
3:     $results = predict(X, model)$;
4: end procedure
```

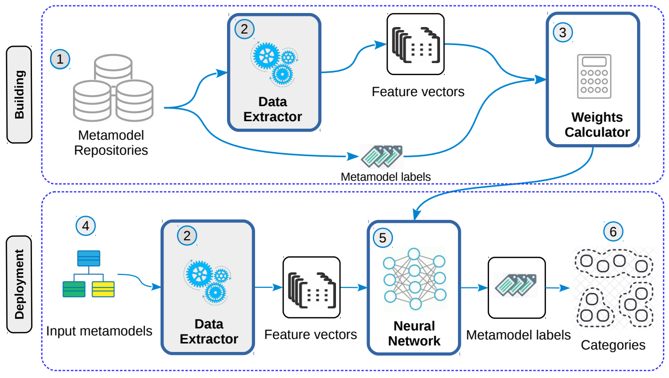


Fig. 3. The AURORA architecture.

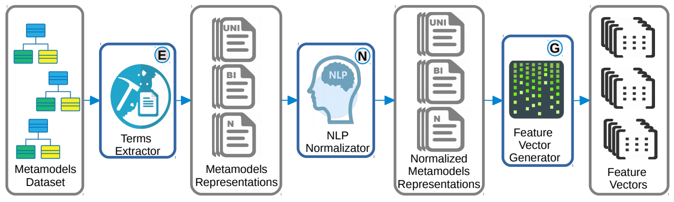


Fig. 4. The data extraction process.

B. System Architecture

The architecture of AURORA is illustrated in Fig. 3 and consists of the building and deployment phases. The former processes the labeled metamodels (1) as follows: each metamodel is serialized into a feature vector, a format that AURORA can process through the `Data Extractor` (2); subsequently, the feature vectors and the corresponding labels are used to train AURORA using the `Weights Calculator` (3). 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 (4) is entered in AURORA, it is parsed employing the `Data Extractor` (2); the generated feature vector is then fed to the neural network (5) 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 (6).

Pre-processing tasks are performed to transform a metamodel into a feature vector [38], i.e., $X = (x_1, x_2, ..., x_L)$, L is the number of input neurons (see Fig. 2). We illustrate how the `Data Extractor` (2) works by means of Fig. 4. The `Term Extractor` (E) parses terms from the input metamodel. Then, the extracted raw terms are normalized by the `NLP Normalizator` (N) that performs the following Natural Language Processing (NLP) steps: *(i)* stemming *(ii)* lemmatization, and *(iii)* stop words removal [23], [55].

Three *encoding schemes* are employed to represent different information granularities concerning the terms extracted from all named instances. In particular, we make use of the following encodings:

– *uni-gram* is a simple collection of terms that does not reflect any metamodel structure;
– *bi-gram* partly represents the structure of the metamodels by considering the containment relations between named 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, etc.).

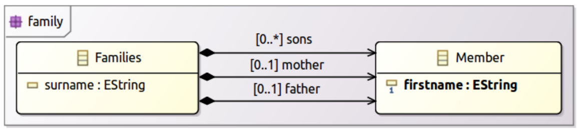


Fig. 5. The *Family* metamodel

As an example, let us take into consideration the *Family* metamodel in Fig. 5. Then according to the extraction process in Fig. 4, the lexical terms in the metamodel are extracted and encoded as prescribed by the above encoding schemes. Table I summarizes the outcome by presenting all lexical terms in the metamodel, and the corresponding encodings for both the raw and normalized feature vectors. For the sake of presentation, each metamodel element name is given with an identifier and its metamodel type. For instance, the package name `family` is *(i)* individually encoded in uni-grams; *(ii)* in bi-grams as part of the terms of the metaclasses `#2` and `#7`; and *(iii)* in n-grams as part of the terms of the structural features `#3`, `#4`, `#5`, `#6`, and `#8`. In addition, the encodings related to normalized feature vectors require the term to be subject to stemming, lemmatization, and stop words removal. Thus, the term `family` is translated into `famili` and comprises both the package `family` and the metaclass `Families`. Moreover, additional information about cardinalities (e.g., 0.1) and whether the node is subject to containment (e.g., TRUE) is appended.

In order to filter out those terms that do not occur frequently, *cut-off* values are given: they define the least number of term occurrences in all the metamodels. As illustrated in Section IV, less frequent terms represent singularities that must be avoided. On the other hand, as demonstrated in Table V higher cut-off values give place to accuracy problems (see for instance the "uni-gram" column). Finally, the `Feature Vector Generator` (G) yields *filtered* feature vectors whose terms have a frequency higher than the cut-off values. The resulting feature vectors, also called Term Document Matrix (TDM) [38], is a two-dimensional matrix where each row represents a metamodel in the repository and columns are the extracted terms, so each entry (j, i) represents the frequency of term i in metamodel j.

IV. Evaluation

In this section, we analyze AURORA's performance employing a thorough evaluation based on a dataset and various quality indicators. The section is organized as follows: Section IV-A presents the dataset used in the evaluation; Section IV-B explains the applied methodology, and Section IV-C recalls the evaluation metrics. Finally, some research questions are introduced in Section IV-D.

A. Dataset

The experimental evaluation is based on a dataset consisting of 555 metamodels retrieved from GitHub [62]. Each metamodel has been manually inspected and classified according to 9 different categories. The results of the classification process are summarized in Table II: for each category, the number of metamodels (**# of mm**), the average number of metaclasses (**avg MC**), and the average number of structural features (**avg Fe.**) of each metamodel are reported. Furthermore, for the sake of presentation, each category has been assigned a unique identifier (**Label**).

To discard those terms that are considered sparse with respect to the whole corpus (and as such not too representative), the following cut-off values are considered $c = \{2, 4, 6, 8\}$. This is

TABLE I
THE *Family* METAMODEL DATA EXTRACTION.

Metamodel element			Raw feature vector			Normalized feature vector		
element type	**id**	**element name**	**uni-gram**	**bi-gram**	**n-gram**	**uni-gram**	**bi-gram**	**n-gram**
package	#1	family	family	–	–	famili	–	–
metaclass	#2	Families	Families	family.Families	–		famili.famili	–
attribute	#3	surname	surname	Families.surname	Families.surname.Int.0.1	surnam	famili.surnam	famili.surnam.Int.0.1
reference	#4	mother	mother	Families.mother	Families.mother.0.1.TRUE	moth	famili.moth	famili.moth.0.1.TRUE
	#5	father	father	Families.father	Families.father.0.1.TRUE	fath	famili.fath	famili.fath.0.1.TRUE
	#6	sons	sons	Families.sons	Famili.sons.0.*.TRUE	son	famili.son	famili.son.0.*.TRUE
metaclass	#7	Member	Member	family.Member	–	memb	famili.memb	–
attribute	#8	firstname	firstname	Member.firstname	Member.firstname.0.1.Int	firstnam	memb.firstnam	memb.firstnam.0.1.Int

TABLE II
THE CATEGORIES OF METAMODELS USED IN THE EVALUATION.

Label	Category Name	# of mm	avg MC	avg Fe.
A	Bibliography DSLs	58	19.79	42.59
B	Issue Tracker DSLs	7	8.57	42.14
C	Pro. build and man. DSLs	38	38.00	90.42
D	Review system DSLs	24	213.58	186.13
E	Database DSLs	101	30.41	69.46
F	Office tools DSLs	54	37.93	96.22
G	Petrinet DSLs	76	13.88	31.95
H	State machine DSLs	159	20.09	30.58
I	Req. spec. DSLs	38	47.78	73.69
	Total	**555**		

inspired by one of the pre-processing phases done in the domain of Document Representation to reduce the dimensionality of a set of documents [57]. Larger values of c would miss the intrinsic features of the dataset by excluding too many terms. The number of input features for the configurations is listed in Table III. It also corresponds to the number of perceptrons in the first layer of the neural network depicted in Fig. 2. It is worth noting that using n-gram as encoding scheme leads to the largest number of input perceptrons, i.e., $10,837$ neurons when $c = 2$, while uni-gram needs much fewer neurons to represent a metamodel, e.g., $1,240$ neurons when $c = 8$. In essence, the computational complexity fluctuates according to the chosen encoding techniques.

TABLE III
THE NUMBER OF INPUT FEATURES FOR EACH CONFIGURATION (**L**).

Encoding	Cut-off value (**c**)			
	2	4	6	8
uni-gram	4,356	2,412	1,715	1,240
bi-gram	6,895	2,665	1,747	1,174
n-gram	10,837	4,166	2,749	1,733

B. Methodology

The performance of AURORA has been assessed by applying the ten-fold cross-validation technique [1], which is considered among the best methods for model selection in Machine Learning [33]. 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 Sec. III, 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 Sec. IV-A. 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.

In this sense, the validation process simulates a real deployment as follows. The training data corresponds to the repositories being available for any mining purposes; they are collected from different sources 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 a real deployment, e.g., being suitable for classifying metamodel crawled from GitHub.

For the sake of reproducibility, we published in GitHub the AURORA tool together with all metadata parsed from the original dataset.[2]

C. Evaluation Metrics

Given a test set, from the manually labeled data we know exactly which category each metamodel belongs to. Thus from the testing data, we create N independent groups of metamodels with labels, i.e., $G = (G_1, G_2, .., G_N)$, which are called ground-truth data. After running AURORA on the test set, we obtain N classes i.e., $C = (C_1, C_2, .., C_N)$, and each contains a set of metamodels. We are interested in how well the produced categories match with the ground-truth data. Thus, to measure the performance of AURORA, *success rate*, *precision*,

[2]https://github.com/models2019-aurora/AURORA

recall, and F_1 *score* are utilized [41]. If $match_i = |G_i \cap C_i|$ is the number of items that appear both in the results and ground-truth data of class i, then the metrics are explained as follows.

Success rate: It is defined as the ratio of correctly classified metamodels to the total number of metamodels in the test set.

$$success\ rate = \frac{\sum_i^N match_i}{\sum_i^N |G_i|} \times 100\% \tag{3}$$

Precision and Recall: These metrics are used to measure how accurate the results are with respect to the ground-truth data. *Precision* is the ratio of the classified items belonging to the ground-truth data:

$$precision_i = \frac{match_i}{|C_i|} \tag{4}$$

and *recall* is the ratio of the ground-truth items being found in the classified items:

$$recall_i = \frac{match_i}{|G_i|} \tag{5}$$

F$_1$ score (F-Measure): The metric is computed as the harmonic average of precision and recall by means of the following formula:

$$F_1 = \frac{2 \cdot precision_i \cdot recall_i}{precision_i + recall_i} \tag{6}$$

Recommendation time: We measure the time needed by AURORA to train and return a prediction on a specific platform. In particular, a laptop with Intel Core i5-7200U CPU @ 2.50GHz $\times$ 4, 8GB RAM, and Ubuntu 16.04 has been exploited for the measurement.

D. Research Questions

By performing a series of experiments on the given dataset using the presented methodology, we attempt to answer the following research questions:

- **RQ**$_1$: *How well can AURORA classify metamodels in terms of success rate, precision, recall, and F_1?* Being accurate is of high importance in the context of classification. Thus, we evaluate if AURORA is capable of categorizing each testing metamodel to its correct class.
- **RQ**$_2$: *Which encoding scheme brings a better classification performance?* We evaluate which technique between uni-gram, bi-gram, and n-gram yields the best trade-off between best prediction accuracy and computational performance.
- **RQ**$_3$: *What is the execution time for training and classifying models?* Finally, it is also important to have limited execution time. We measure the time needed to build the model as well as to perform classification.

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

V. Results and Discussions

We answer the research questions in Section V-A and present some related discussions in Section V-B.

A. Result Analysis

RQ$_1$: *How well can AURORA classify metamodels in terms of success rate, precision, recall, and F_1?* To investigate the effect of different sets of input features, the cut-off values are varied as already reasoned in Section IV-A. We compute success rate, precision, recall, and F$_1$ scores for every test configuration. The average success rates of all ten folds are reported in Table IV.

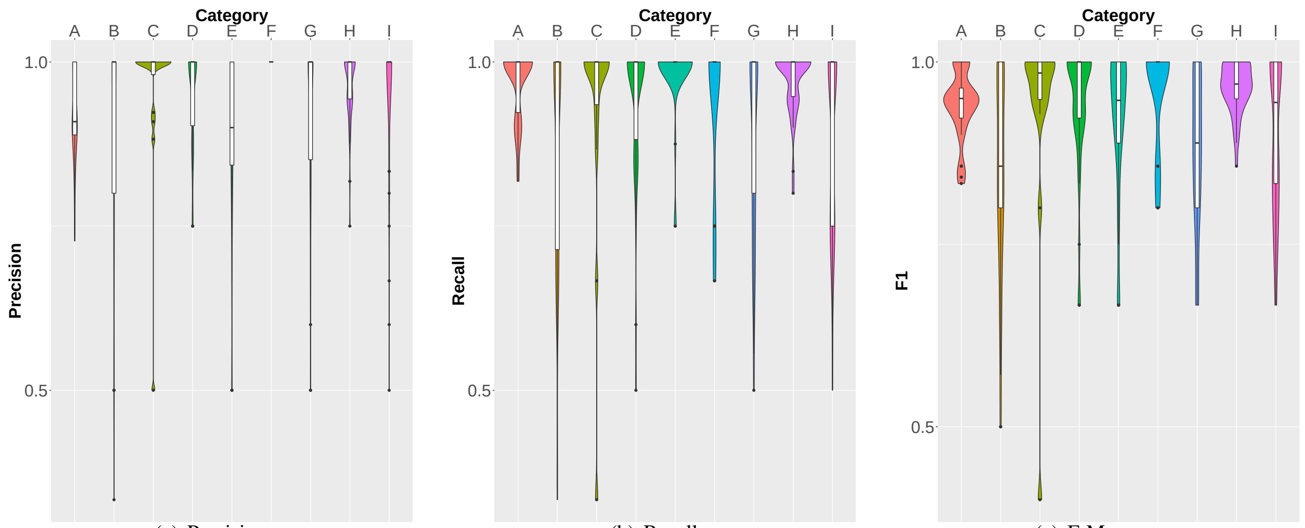
TABLE IV
SUCCESS RATE FOR ALL CONFIGURATIONS.

Encoding	Cut-off value (**c**)			
	2	4	6	8
uni-gram	**95.45**	**94.00**	**94.72**	**94.20**
bi-gram	90.35	86.36	82.17	81.81
n-gram	89.45	84.31	80.35	80.77

In general, AURORA obtains a considerably high success rate that is always larger than 80%. In particular, the minimum success rate is 80.35 and obtained when using n-gram as the encoding scheme and 6 as the cut-off value. Meanwhile, the maximum success rate is 95.45, and it is reached when using uni-gram together with $c = 2$, i.e., more than 95% of the testing metamodels are correctly classified. Furthermore, as reported in the table, the performance generally decreases proportionally to the cut-off value, which implies that incorporating more terms into the feature set boosts the classification accuracy. Nevertheless, this also adds more computational complexity since the number of input neurons needs to be increased accordingly, as depicted in Fig. 2 and Table III.

To further analyze the performance of AURORA, we consider the accuracy for every metamodel category considered in our evaluation. For each quality indicator, we combine the results coming from all cut-off values i.e., $c = \{2, 4, 6, 8\}$ and represent them in a single figure using violin boxplots as shown in Fig. 6(a), 6(b), and 6(c). These figures demonstrate how accurately AURORA assigns a metamodel to each category, regardless of the cut-off values. Each boxplot corresponds to a quality indicator, i.e., precision, recall, and F$_1$ of a category. For instance, Fig. 6(a) depicts the precision scores for the categories. The slim violins imply that this indicator fluctuates dramatically. There, the lowest precision is seen by Category B (**Issue Tracker DSLs**) whose number of items is considerably low, i.e., 7 metamodels (see Table II). Among others, by Category F (**Office tools DSLs**), AURORA achieves the maximum precision compared to other categories, i.e., $precision = 1$ for all cases. That means all metamodels being classified receive the correct category. Category H has the largest number of items, i.e., 159 metamodels.

We consider the recall scores for all categories as depicted in Fig. 6(b). Like for precision, Category B (**Issue Tracker DSLs**) has the lowest recall. AURORA gains a better recall by Category A (**Bibliography DSLs**) and H (**State machine DSLs**). Finally, we analyze the performance of AURORA with respect to F$_1$ score in Fig. 6(c). The worst F$_1$ scores are seen by Category B (**Issue Tracker DSLs**) and Category G (**Petrinet



(a) Precision (b) Recall (c) F-Measure

Fig. 6. Violin boxplots representing Precision, Recall, F_1 scores for all metamodel categories.

DSLs). Again, the best F_1 score is obtained for Category A and H.

AURORA is capable of predicting the label of an incoming metamodel with a high success rate and accuracy. The maximum success rate, precision, recall, and F_1 are 95.40%, 0.945, 0.938, and 0.942, respectively.

RQ$_2$: *Which encoding scheme brings a better classification performance?*

We consider various experimental configurations to see which encoding technique between uni-gram, bi-gram, and n-gram facilitates a better prediction performance. It is important in practice since each encoding scheme may have a different impact on the performance and only the one that helps achieve a higher accuracy should be chosen. Considering n-gram and bi-gram together, we see that using bi-gram to encode a metamodel, a better classification success rate is obtained. For instance, as shown in Table IV, when $c = 2$, using n-gram yields 89.45 as the success rate, meanwhile the corresponding value by bi-gram is 90.35. The same trend is witnessed by the other cut-off values, i.e., $c = \{4, 6, 8\}$. Between the three encoding schemes, uni-gram contributes to the best success rate. For instance, when $c = 4$, the success rate is 94.00, compared to 86.36 and 84.31 by n-gram and bi-gram, respectively.

Table V reports the mean precision, recall, and F_1 scores for all configurations. For the cut-off value $c = 2$, among the encoding schemes, bi-gram has the lowest precision, recall, and F_1. In particular, it gets 0.878, 0.815 as precision and recall, and 0.845 as F_1. Meanwhile, n-gram yields a slightly better accuracy compared to bi-gram, i.e., 0.895, 0.834, and 0.864. For $c = \{4, 6\}$, bi-gram outperforms n-gram with regards to all quality indicators. For example, when $c = 4$, bi-gram gets 0.839, 0.788, and 0.813 as precision, recall, and F_1. The corresponding scores obtained by using n-gram are 0.826, 0.771, and 0.797 as precision, recall, and F_1, respectively. However, n-gram overtakes bi-gram again when a larger cut-off value is used, i.e., $c = 8$. By this configuration, all the quality scores obtained by using bi-gram are considerably lower compared to those obtained by using n-gram. Altogether, we see that there is no clear winner between bi-gram and n-gram in terms of precision, recall, and F_1.

Being consistent with Table IV, Table V demonstrates that running AURORA with data parsed by uni-gram achieves the best accuracy. In particular, its precision, recall, and F_1 scores are always superior to 0.90, with 0.945 being the maximum. Furthermore, generally there is a performance gain when using uni-gram with more features, i.e., smaller cut-off values c. For example, the worst F_1 is 0.911 when $c = 8$, whereas the best F_1 is 0.942 when $c = 2$. Such a gain also applies to precision and recall, with only one exception for $c = \{6, 8\}$, i.e., $recall = 0.906$ and 0.918, respectively.

In summary, running AURORA with uni-gram provides the best success rate and accuracy. There is no distinct winner between n-gram and bi-gram when they are used to parse input metamodels. Furthermore, a feature set with more terms is beneficial to the prediction outcome.

RQ$_3$: *What is the execution time for training and classifying models?*

Finally, we also measure the execution time needed to train the neural network as well as to perform testing since performance was one of the motivational drives for this work. In order to achieve a reliable comparison, by this evaluation, we run AURORA with 300 epochs for all testing configurations (see Listing 1). Generally, by this number of epochs, the system reaches the performance presented in Section V-A.

Figure 7 shows the average execution time for training on 500 metamodels (9 folds) and testing on 55 metamodels (1 fold). It is evident that running AURORA with n-gram as the encoding scheme takes the longest time to finish. In particular, the system

TABLE V
PRECISION (P), RECALL (R), AND F-MEASURE (F_1) FOR ALL CONFIGURATIONS.

Encoding	c=2			c=4			c=6			c=8		
	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
uni-gram	**0.945**	**0.938**	**0.942**	**0.919**	**0.918**	**0.918**	**0.917**	**0.906**	**0.912**	**0.905**	**0.918**	**0.911**
bi-gram	0.878	0.815	0.845	0.839	0.788	0.813	0.857	0.746	0.797	0.789	0.675	0.728
n-gram	0.895	0.834	0.864	0.826	0.771	0.797	0.789	0.710	0.747	0.822	0.773	0.797

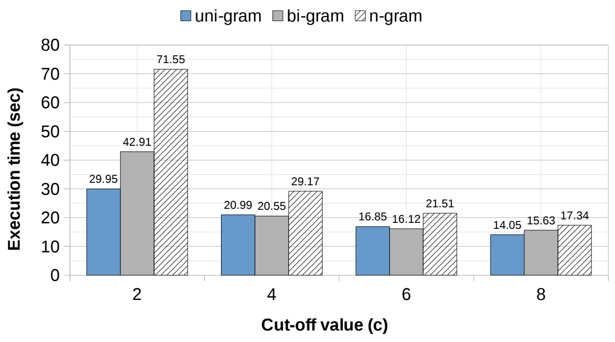


Fig. 7. Timing performance.

spends 71.55 seconds to run the building and deployment phases. Meanwhile, it needs only 29.95 seconds when used together with uni-gram for the same task. This is consistent with Table III where it is shown that n-gram needs more features to represent an input metamodel. With $c = 2$ the difference between the time needed is quite big between uni-gram, bi-gram, and n-gram. Along with a larger value of c, we see that such a difference reduces considerably and becomes almost negligible for a large c. For example, when $c = 8$, the execution time of AURORA with uni-gram, bi-gram, and n-gram is, 14.05, 15.63 and 17.63 seconds, respectively. Together with the outcome of $\mathbf{RQ}_2$, we conclude that performing AURORA with uni-gram brings a superior performance concerning both accuracy and timing.

On the given dataset, AURORA performs the building and deployment phases in a pretty short time when using together with uni-gram as the encoding scheme.

B. Discussions

An interesting result emerging from the experiments is that, among others, 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 a highly informative content, and as seen in Section III-B, uni-gram is not structure preserving. A possible explanation of why uni-gram is instrumental in helping obtain 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. Consequently, it is expected that the outcome might be different if different classification criteria were adopted, which suggests that the encoding performance is not a general property of the method only, but depends also on the specific classification. However, this is a mere assumption (although an interesting one) that needs to be further investigated.

We are also interested in understanding which network topology fosters better success rate and accuracy. In other words, how many hidden layers as well as how many neurons for each layer should be chosen in order to achieve the best prediction. As far as we have seen from current literature, there exist some rules of thumb to set the number of neuron ρ for a hidden layer, e.g., $L \leq \rho \leq N$ [12], or $\rho \leq 2 \times L$ [9] where L, N are the number of input and output neurons, respectively (see Fig. 2). 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 [61]. In this work, we set up a network with only one hidden layer and a certain number of nodes for it (see Fig. 2). We believe that more investigations need to be conducted in order to find a configuration that sustains the overall accuracy.

Through the experiments, we notice that some categories got a higher accuracy, e.g., Category H. Whereas, other categories like B get much lower performance. This may be explained by observing that categories with a higher number of metamodels facilitate a better prediction outcome. However, it is essential to examine whether there is an effective association between the number of items and the prediction accuracy. This remains an open research issue.

Finally, a question that may arise at any time is: How should AURORA be deployed in practice? As shown in Section III-B, initially it is necessary to involve humans, e.g., a group of experienced developers, to manually categorize a *large enough* dataset. Then, the dataset is fed to the system in order 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 developers 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 this way, AURORA is handy since it considerably reduces the effort required to classify metamodels by hands, thus automatizing the categorization process.

VI. Threats to Validity

In the following, we distinguish between internal, construct, and external validity.

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., Category B, D. Such a threat is eased by denser groups, e.g., Category E, H. Furthermore, the selection of the cut-off values may introduce bias, since c is just an even number throughout the evaluation, i.e., $c = \{2, 4, 6, 8\}$. Besides the experiments discussed in this paper, we conducted similar ones with odd numbers, i.e., $c = \{1, 3, 5, 7\}$. However, due to space limits, the final results cannot be introduced here. As far as we can see, there are no subtle differences between the conclusions obtained from the new experiments with those previously presented in the paper.

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 ten-fold cross validation, attempting to simulate a real scenario of classification.

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. We attempt to moderate the threat by considering a set of 555 metamodels that are of different sizes and cover various categories. Nevertheless, such a dataset might not necessarily reflect the entire domain. In this sense, it is essential to evaluate AURORA by incorporating a bigger dataset with more categories, as well as more items for each category. Also considering different classification categories may give more insight about encodings and whether they are (partly or totally) independent from the classification criteria. We consider this task as a future work.

VII. Related Work

This section discusses some of the most important related work about categorization and classification in metamodeling, as well as the adoption of ML in MDE and some prominent applications of Neural Networks.

A. Metamodel categorization

Metamodel classification approaches can be described according to two main categories: those with a classification purpose and those that are agnostic of the purpose.

In [29], different characteristics of UML metamodel extension mechanisms are discussed 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 scope of the paper is to provide modelers with a reliable theoretical base for selecting the right level to extend the metamodel in order to find the right tradeoff of the above aspects. A classification of UML stereotypes is given in [8] where they are analyzed according to their potential to alter the syntax and semantics of the base language in order to be able to control their application in practice. Feature-based criteria for classifying approaches according to the type of spatial relation involved are discussed in [13]. In contrast with 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 in [16] only considers visual programming languages, mainly from the programming 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, and 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, the approach in [39] focuses on requirement diagrams where the various relationships within a diagram have specific meanings that help achieve better classification performance. More recently, generic clustering techniques have been extensively investigated as demonstrated in [53]. 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 with our approach, and the representation is only grouping similar concepts in the same containers.

Two of the very first attempts to cluster metamodels have been presented in [5], [3]. In [5], the authors propose the application of clustering techniques to automatically organize stored metamodels in a structured repository and to provide users with an overview of the application domains covered by the available metamodels. The approach presented in [3] encodes metamodels in a vector space model, and applies hierarchical clustering techniques to compare and visualize them as a tree-based structure. Both the work presented in [5] and [3] share the goal of this paper even though they are based on unsupervised learning techniques, whereas in this work a supervised learning approach is proposed.

B. Machine Learning for MDE

In the context of collaborative modeling, the authors in [4] propose the adoption of an Unsupervised and Reinforcement Learning (RL) approach to repair broken models, which have been corrupted because of conflicting changes. The main intent is to potentially reach model repairing with human-quality without requiring supervision.

The work in [15] reviews the main modeling languages used in ML as well as inference algorithms and corresponding software implementations. The aim of this work is to explore the opportunities of defining a DSML for probabilistic modeling. Similarly, ML is collaterally treated in [47]. 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 are introduced in [24]. A similar tool called OptiML is proposed in [54], aiming to bridge the gap between ML algorithms and heterogeneous hardware to provide a productive programming environment.

The authors in [25] propose interleaving ML with domain modeling. In particular, they decompose machine learning into reusable, chainable, and independently computable small learning units. These micro learning units are modeled together with and at the same level as the domain concepts. The paper in [21] investigates an alternative way to create model transformations, which are deduced from examples of transformed models. This underlying assumption is that examples are easier to write than a transformation program and they are often already available. In particular, the paper introduces a Model Transformation By Example approach, from examples to transformation rules. The underlying machine learning approach is based on Relational Concept Analysis, an extension of the Formal Concept Analysis theory, that classifies a set of objects based on their properties.

C. Applications of Neural Networks

The ability to learn from labeled data allows neural networks to have a wide range of applications. The work in [10] presents an overview of neural networks for pattern recognition. A multi-purpose algorithm based on a single deep convolutional neural network (CNN) for solving various problems, e.g., face detection, face alignment, smile detection is introduced in [44]. The authors in [61] review different applications of neural networks also identifying the characteristics that make them suitable for forecasting tasks. The work also identifies issues related to the selection of number hidden layers as well as the number of neurons.

For classification, neural networks have demonstrated their suitability in various application domains. An approach to classify individual characteristics in behavioural sciences using a neural network is proposed in [45]. The authors in [2] 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. A series of experiments with convolutional neural networks for sentence-level classification tasks is reported in [31].

To the best of our knowledge, AURORA is the first application of neural networks to classify metamodels. Thus, our work distinguishes itself from existing studies in MDE as follows: *(i)* We employ a well-founded ML technique to automatize the classification of metamodels; *(ii)* The tool can automatically learn from labeled metamodels to deal with unlabeled metamodels, thereby being applicable to real metamodel categorization scenarios; *(iii)* Last but not least, our proposed paradigm is expected to pave the way for further adoptions of advanced ML algorithms in MDE, including also Deep Learning [42].

VIII. Conclusions

Starting from the need for classification of metamodels, we came up with a feed-forward neural network that is able to learn from labeled dataset, so as to categorize unlabeled data afterwards. We implemented and evaluated AURORA, a supervised classifier for metamodel repositories. An evaluation on a dataset of 555 manually classified metamodels exploiting ten-fold cross validation demonstrates that the tool is capable of categorizing test data with high success rates and accuracy. With the best configuration, AURORA almost obtains the maximum prediction performance. As a matter of fact, our proposed tool has not been evaluated against any baseline, since we are not aware of any comparable approaches that tackle the same issue. We plan to further study the performance of our proposed approach by making use of more data as well as incorporating additional quality indicators. We also plan to investigate the adoption of neural networks for classifying different kinds of modeling artifacts (e.g., models, model transformations, code generators) with the aim of conceiving recommender systems being able to support different modeling activities.

References

[1] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statist. Surv.*, vol. 4, pp. 40–79, 2010.

[2] M. F. Augusteijn and B. A. Folkert, "Neural network classification and novelty detection," *International Journal of Remote Sensing*, vol. 23, no. 14, pp. 2891–2902, 2002.

[3] O. Babur, L. Cleophas, and M. Brand, "Hierarchical Clustering of Metamodels for Comparative Analysis and Visualization," vol. 9764, 07 2016, pp. 3–18.

[4] A. Barriga, A. Rutle, and R. Heldal, "Automatic model repair using reinforcement learning," in *Proceedings of Workshops co-located with MODELS 2018, Copenhagen, Denmark, October, 14, 2018.*, 2018, pp. 781–786.

[5] F. Basciani, J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Automated clustering of metamodel repositories," in *Advanced Information Systems Engineering*, S. Nurcan, P. Soffer, M. Bajec, and J. Eder, Eds. Cham: Springer International Publishing, 2016, pp. 342–358.

[6] Y. Bengio, *Practical Recommendations for Gradient-Based Training of Deep Architectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 437–478.

[7] P. Berkhin, "A survey of clustering data mining techniques," in *Grouping multidimensional data*. Springer, 2006, pp. 25–71.

[8] S. Berner, M. Glinz, and S. Joos, "A classification of stereotypes for object-oriented modeling languages," in *«UML»'99 — The Unified Modeling Language*, R. France and B. Rumpe, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 249–264.

[9] M. J. Berry and G. Linoff, *Data Mining Techniques: For Marketing, Sales, and Customer Support*. New York, NY, USA: John Wiley & Sons, Inc., 1997.

[10] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.

[11] B. Bislimovska, A. Bozzon, M. Brambilla, and P. Fraternali, "Textual and content-based search in repositories of web application models," *ACM Transactions on the Web (TWEB)*, vol. 8, no. 2, p. 11, 2014.

[12] A. Blum, *Neural Networks in C++: An Object-oriented Framework for Building Connectionist Systems*. New York, NY, USA: John Wiley & Sons, Inc., 1992.

[13] P. Bottoni and A. Grau, "A suite of metamodels as a basis for a classification of visual languages," in *2004 IEEE Symposium on Visual Languages - Human Centric Computing*, Sep. 2004, pp. 83–90.

[14] L. Bottou, “Stochastic gradient learning in neural networks,” in *In Proceedings of Neuro-Nîmes. EC2*, 1991.

[15] D. Breuker, “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*, 2014, pp. 758–767.

[16] M. M. Burnett and M. J. Baker, “A classification system for visual programming languages,” *Journal of Visual Languages and Computing*, vol. 5, no. 3, p. 287, 1994.

[17] J. Cabot, R. Clarisó, M. Brambilla, and S. Gérard, “Cognifying model-driven software engineering,” in *Software Technologies: Applications and Foundations*, M. Seidl and S. Zschaler, Eds. Cham: Springer International Publishing, 2018, pp. 154–160.

[18] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, “Large scale distributed deep networks,” in *Proceedings of the 25th Int. Conf. on Neural Information Processing Systems - Volume 1*, ser. NIPS’12. USA: Curran Associates Inc., 2012, pp. 1223–1231.

[19] T. Di Noia, R. Mirizzi, V. C. Ostuni, D. Romito, and M. Zanker, “Linked Open Data to Support Content-based Recommender Systems,” in *8th International Conference on Semantic Systems*. New York: ACM, 2012, pp. 1–8.

[20] L. R. Dice, “Measures of the amount of ecologic association between species,” *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[21] X. Dolques, M. Huchard, C. Nebut, and P. Reitz, “Learning Transformation Rules from Transformation Examples: An Approach Based on Relational Concept Analysis,” in *2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops*, 2010, pp. 27–32.

[22] P. Domingos, “A few useful things to know about machine learning,” *Commun. ACM*, vol. 55, no. 10, pp. 78–87, Oct. 2012.

[23] W. B. Frakes, “Term conflation for information retrieval,” in *Proceedings of the 7th annual international ACM SIGIR conference on Research and development in information retrieval*. British Computer Society, 1984, pp. 383–389.

[24] V. García-Díaz, J. P. Espada, B. C. P. G. Bustelo, and J. M. C. Lovelle, “Towards a standard-based domain-specific platform to solve machine learning-based problems,” *IJIMAI*, vol. 3, no. 5, pp. 6–12, 2015.

[25] T. Hartmann, A. Moawad, F. Fouquet, and Y. Le Traon, “The Next Evolution of MDE: A Seamless Integration of Machine Learning into Domain Modeling,” in *2017 ACM/IEEE 20th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS)*, Sep. 2017, pp. 180–180.

[26] C. Hein, T. Ritter, and M. Wagner, “Model-driven tool integration with modelbus,” in *Workshop Future Trends of Model-Driven Development*, 2009, pp. 50–52.

[27] T. Isakowitz and R. J. Kauffman, “Supporting search for reusable software objects,” *IEEE Transactions on Software Engineering*, vol. 22, no. 6, pp. 407–423, 1996.

[28] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: a review,” *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[29] W. S. L. Z. Z. M. X. M. Jiang, Yanbing and H. Ma., “On the classification of uml’s meta model extension mechanism,” in *International Conference on the Unified Modeling Language*, 2004, pp. 54–68.

[30] B. Karasneh and M. R. Chaudron, “Online img2uml repository: An online repository for uml models.” in *EESSMOD@ MoDELS*, 2013, pp. 61–66.

[31] Y. Kim, “Convolutional neural networks for sentence classification,” in *Proceedings of the 2014 Conf. on Empirical Methods in NLP, EMNLP 2014, October 25-29, 2014, Doha, Qatar*, 2014, pp. 1746–1751.

[32] M. Koegel and J. Helming, “EMFStore: a model repository for EMF models,” in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 2. IEEE, 2010, pp. 307–308.

[33] R. Kohavi, “A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection,” in *14th Int. Joint Conf. on Artificial Intelligence*. San Francisco: Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143.

[34] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, “Supervised machine learning: A review of classification techniques,” *Emerging artificial intelligence applications in computer engineering*, vol. 160, pp. 3–24, 2007.

[35] A. Krenker, J. Bester, and A. Kos, “Introduction to the artificial neural networks,” in *Artificial Neural Networks*, K. Suzuki, Ed. Rijeka: IntechOpen, 2011, ch. 1.

[36] A. Kuhn, S. Ducasse, and T. Girba, “Enriching reverse engineering with semantic clustering,” in *12th Working Conference on Reverse Engineering (WCRE’05)*. IEEE, 2005, pp. 10–pp.

[37] R. Kutsche, N. Milanovic, G. Bauhoff, T. Baum, M. Cartsburg, D. Kumpe, and J. Widiker, “Bizycle: Model-based interoperability platform for software and data integration,” *Proceedings of the MDTPI at ECMDA*, vol. 430, 2008.

[38] E. Leopold and J. Kindermann, “Text categorization with support vector machines. How to represent texts in input space?” *Machine Learning*, vol. 46, no. 1-3, pp. 423–444, 2002.

[39] O. Lopez, M. A. Laguna, and F. J. García, “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)*, 2002, pp. 71–82.

[40] O. Maqbool and H. Babri, “Hierarchical clustering for software architecture recovery,” *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 759–780, 2007.

[41] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, L. Ochoa, T. Degueule, and M. Di Penta, “FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns,” in *41st ACM/IEEE International Conference on Software Engineering (ICSE 2019)*, 2019.

[42] M. A. Nielsen, “Neural networks and deep learning,” 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[43] I. Portugal, P. S. C. Alencar, and D. D. Cowan, “The use of machine learning algorithms in recommender systems: A systematic review,” *CoRR*, vol. abs/1511.05263, 2015.

[44] R. Ranjan, S. Sankaranarayanan, C. D. Castillo, and R. Chellappa, “An all-in-one convolutional neural network for face analysis,” in *2017 12th IEEE Int. Conf. on Automatic Face Gesture Recognition (FG 2017)*, May 2017, pp. 17–24.

[45] D. Reby, S. Lek, I. Dimopoulos, J. Joachim, J. Lauga, and S. Aulagnier, “Artificial neural networks as a classification method in the behavioural sciences,” *Behavioural Processes*, vol. 40, no. 1, pp. 35 – 43, 1997.

[46] R. Rojas, *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer-Verlag, 1996.

[47] A. M. S. Rössler and S. Günnemann, “ThingML+: Augmenting Model-Driven Software Engineering for the Internet of Things with Machine Learning,” in *Procs. of Workshops co-located with MODELS 2018, Copenhagen, Denmark, October, 14, 2018.*, ser. CEUR Workshop Proceedings, R. Hebig and T. Berger, Eds., vol. 2245. CEUR-WS.org, 2018, pp. 521–523. [Online]. Available: http://ceur-ws.org/Vol-2245

[48] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, “The adaptive web,” P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds. Berlin, Heidelberg: Springer-Verlag, 2007, ch. Collaborative Filtering Recommender Systems, pp. 291–324.

[49] D. C. Schmidt, “Guest editor’s introduction: Model-driven engineering,” *Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[50] D. Sontag, K. Collins-Thompson, P. N. Bennett, R. W. White, S. Dumais, and B. Billerbeck, “Probabilistic models for personalizing web search,” in *Procs. of the Fifth ACM Int. Conf. on Web Search and Data Mining*, ser. WSDM ’12. New York, NY, USA: ACM, 2012, pp. 433–442.

[51] D. Spinellis and C. Szyperski, “How is open source affecting software development?” *IEEE Software*, vol. 21, no. 1, pp. 28–33, Jan 2004.

[52] J. Stilgoe, “Machine learning, social learning and the governance of self-driving cars,” *Social Studies of Science*, vol. 48, no. 1, pp. 25–56, 2018, pMID: 29160165.

[53] D. Strüber, M. Selter, and G. Taentzer, “Tool support for clustering large meta-models,” in *Procs. of the Workshop on Scalability in Model Driven Engineering*, ser. BigMDE ’13. New York, NY, USA: ACM, 2013, pp. 7:1–7:4.

[54] A. Sujeeth, H. Lee, K. Brown, T. Rompf, H. Chafi, M. Wu, A. Atreya, M. Odersky, and K. Olukotun, “OptiML: an implicitly parallel domain-specific language for machine learning,” in *Procs. of the 28th Int. Conf. on Machine Learning (ICML-11)*, 2011, pp. 609–616.

[55] X. Sun, X. Liu, J. Hu, and J. Zhu, “Empirical studies on the nlp techniques for source code data preprocessing,” in *Proceedings of the 2014 3rd Int. Workshop on Evidential Assessment of Soft. Tech.*, ser. EAST 2014. New York, NY, USA: ACM, 2014, pp. 32–39.

[56] D. Svozil, V. Kvasnicka, and J. Pospíchal, “Introduction to multi-layer feed-forward neural networks,” *Chemometrics and Intelligent Laboratory Systems*, vol. 39, pp. 43–62, 11 1997.

[57] B. Tang, R. Spiteri, E. Milios, R. Zhang, S. Wang, J. Tougas, and M. Shafiei, “Document representation and dimension reduction for text clustering,” in *2013 IEEE 29th Int. Conf. on Data Engineering Workshops (ICDEW)*. Los Alamitos, CA, USA: IEEE Computer Society, apr 2007, pp. 770–779.

[58] F. Thung, D. Lo, and J. Lawall, “Automated library recommendation,” in *2013 20th Working Conf. on Reverse Engineering (WCRE)*, Oct 2013, pp. 182–191.

[59] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[60] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, “Machine learning in manufacturing: advantages, challenges, and applications,” *Production & Manufacturing Research*, vol. 4, no. 1, pp. 23–45, 2016.

[61] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, “Forecasting with artificial neural networks:: The state of the art,” *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[62] Önder Babur, “A labeled Ecore metamodel dataset for domain clustering,” Mar. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2585456