Machine learning methods for model classification: A comparative study

José Antonio Hernández López  
Universidad de Murcia  
Spain  
joseantonio.hernandez6@um.es

Jesús Sánchez Cuadrado  
Universidad de Murcia  
Spain  
jesusc@um.es

Riccardo Rubei  
Università degli studi dell’Aquila  
Italy  
riccardo.rubei@graduate.univaq.it

Davide di Ruscio  
Università degli studi dell’Aquila  
Italy  
davide.diruscio@univaq.it

ABSTRACT
In the quest to reuse modeling artifacts, academics and industry have proposed several model repositories over the last decade. Different storage and indexing techniques have been conceived to facilitate searching capabilities to help users find reusable artifacts that might fit the situation at hand. In this respect, machine learning (ML) techniques have been proposed to categorize and group large sets of modeling artifacts automatically. This paper reports the results of a comparative study of different ML classification techniques employed to automatically label models stored in model repositories. We have built a framework to systematically compare different ML models (feed-forward neural networks, graph neural networks, k-nearest neighbors, support version machines, etc.) with varying model encodings (TF-IDF, word embeddings, graphs and paths). We apply this framework to two datasets of about 5,000 Ecore and 5,000 UML models. We show that specific ML models and encodings perform better than others depending on the characteristics of the available datasets (e.g., the presence of duplicates) and on the goals to be achieved.

CCS CONCEPTS
• Software and its engineering → Model-driven software engineering; • Computing methodologies → Machine learning.

KEYWORDS
Model classification, Model-Driven Engineering, Machine learning

1 INTRODUCTION
In recent years, the use of Artificial Intelligence (AI) and Machine Learning (ML) techniques to solve Model-Driven Engineering (MDE) problems has begun to gain traction. This is an active line of work intended to explore the application of ML to enhance several MDE-related scenarios, as testified by the increasing research corpus [9]. For instance, feed-forward neural networks have been used to label metamodels stored in model repositories automatically [29]. An LSTM architecture is used to perform model transformation by example [10]. A graph neural network assesses realistic model generators [25]. Model assistants based on providing recommendations are starting to be built using different techniques [15, 36].

Common obstacles which need to be addressed when applying ML-based projects for dealing with MDE problems are twofold: i) understand the typical steps that need to be performed to employ ML algorithms; ii) determine the best combination of ML model and representation (i.e., the encoding of software models for their consumption by ML algorithms) for the problem at hand. Unfortunately, this topic is not yet well understood and has not been explored in depth. By considering these shortcomings, this paper overviews the typical process that needs to be followed to apply ML projects in MDE. Subsequently, a concrete instance of the presented methodology is presented to systematically compare several ML methods and representations used to support the model classification task. The goal is to determine the pros and cons of each method and derive a set of applicable lessons. To this end, we will consider the model classification problem using ModelSet [21] as the target dataset. Over it, we will run and compare different classification algorithms. Thus, the paper makes the following contributions:

- Overview of a typical workflow that needs to be followed when applying ML models for MDE problems. The paper itself serves as a guide for MDE experts who want to apply ML techniques to understand the different types of models available and how to map software models to the input representations expected by these ML models;
- A methodology instantiating the overviewed process to automatically evaluate the suitability of different ML models to address a particular problem in MDE;
- A framework implementing the proposed methodology for the task of model classification, in which several ML models are systematically compared.
Organization. Section 2 presents the related work and some background about ML and MDE. Section 3 describes our methodology to perform the comparison of ML methods, while Sect. 4 describes in detail the models and encodings used. Section 5 describes the experimental setup and Sect. 6 presents the results of the experiments and a critical discussion. Finally, Section 7 concludes the paper and highlights possible future directions.

2 BACKGROUND AND RELATED WORK

This section presents an overview of recent applications of ML to address MDE problems (Sect. 2.1) and describes a typical workflow to apply ML (Sect. 2.2), which is adapted for the sake of the software model classification problem as presented in Section 3.

2.1 Machine Learning in MDE

Machine learning is a branch of artificial intelligence that encompasses techniques to make computers learn from data. Depending on the shape of the data, ML techniques can be classified as supervised and unsupervised learning. In supervised learning the data includes the labels, and the ML algorithm has to learn a mapping function between the input data and the target labels. If this function is real-valued, then a regression problem is faced. Whereas, if this function is discrete, we have a classification problem. In contrast, in unsupervised learning the data is not labelled, and the system tries to identify patterns by itself. A typical task is clustering, in which the system identifies groups of similar examples according to some criteria and similarity functions.

Machine Learning algorithms have been successfully employed to face different issues in MDE [4, 18]. For example, in [12, 17] ML techniques were used to automatically infer model transformation rules from sets of source and target models. In [13], the authors discussed about the cognification of Model-Driven Software Engineering. The goal is to boost the performance of a process with the utilization of knowledge. The cognification concept is referred not only to artificial intelligence but also includes a combination of past and current human intelligence.

Breuker [8] reviews the main modeling languages used in machine learning to explore model-driven big data analytics. The result is a conceptualization of a DSML. Such a DSML can support code generation from a visual representation of probabilistic models.

ML-Quadrat [27] is a research project conceived to improve ThingML, an open-source modeling tool for IoT/CPS. The authors integrate ML concepts into ThingML at modeling and code generation levels.

In [25], the authors use Graph Neural Networks (GNNs) to characterize realistic model generators by mapping this problem to a binary classification problem. The analysis and classification of model repositories have been addressed also from a clustering perspective. Hierarchical clustering is applied in [6] to organize a collection of metamodels and provide meaningful visualizations of them. Similarly, in [7] the models handled by the MDEForge tool are organized by employing hierarchical clustering. The topic of model encoding for clustering has also been addressed in [5].

AURORA [28, 29] is a tool that exploits a feed-forward neural network to classify metamodels. The authors proved the tool’s capability to classify Ecore models with considerable precision. In a similar vein, a convolutional neural network (CNN) is used to build the MEMOCNN metamodel classifier [30]. The idea is to transform metamodels into special 2D images that the CNN can process. However, both AURORA and MEMOCNN may be biased due to the small size of the dataset (555 Ecore metamodels) and the presence of duplicates. Another issue of the current state-of-the-art in ML applied to MDE is that it typically targets Ecore metamodels. Therefore, we also use UML models in this work, which allows us to contrast the results for the same task applied to Ecore metamodels.

The authors in [14] propose the application of graph kernels to MDE. In this context, Di Rocco et al. [15] proposes MORGAN, a tool based on graph kernels to support the completion of both models and metamodels. The tool is conceived to recommend structural features to complete the models under construction. In particular, MORGAN can recommend classes, fields, and methods within a model class, metaclasses, and structural features like attributes and references.

Burgeño et al. [11] proposed a NLP-based architecture for the completion of software models. The core of their proposal is to use two word embedding models (e.g., GloVe [31], word2vec [26], etc), one trained with general knowledge and the other with contextual knowledge. Both models are interpolated to perform the recommendations.

Altogether, there is a trend to apply ML algorithms to address MDE problems. However, there is currently a gap regarding the knowledge about which ML models and representations are best suited for each type of MDE problem. This work attempts to address this shortcoming for the problem of model classification.

2.2 Common Machine Learning workflow

The application of ML techniques encompasses the execution of a common workflow [2] consisting of typical tasks as shown in Fig. 1 and discussed below.

Data gathering: Identifying data sources relevant to the problem at hand is the first step in a ML project. The size and quality of the considered data are among the peculiar characteristics affecting the final results. Once data sources are identified, data is fetched and typically archived locally for subsequent manipulations.

Data preparation: A key aspect is to ensure that data is in a suitable condition to be used. In this phase, clearing and pre-processing operations are performed to fix issues like missing values, duplication, contradictions, and outlier values. Such a phase is of paramount

Figure 1: Overview of a common workflow for ML.
importance because if the ML application being developed is based on inaccurate data, the corresponding quality will not be as wanted. **Feature engineering:** In this phase, the properties of the subjects under investigation are extracted from raw data. For instance, if we want to predict the population growth in a specific geographical area, we could discard attributes like hair color or height, but select attributes like salary and unemployment rates.

**ML model selection:** Several machine learning algorithms might be suitable for the available data. By relying on previous experiences or similar applications already developed, candidate ML algorithms are selected. It is important to remark that this is an iterative process that encompasses conducting different experiments to evaluate how the chosen algorithms perform with the available data with the final goal of identifying the best configuration.

**Data encoding:** Depending on the previously selected ML algorithm, prepared data must be encoded accordingly. For instance, if the chosen ML algorithm is based on a graph-based similarity, it is necessary to encode data as graphs.

**ML model training:** This is the process of feeding the selected ML algorithm with the prepared and encoded data. In the case of supervised learning, training data consist of input values and one or more target variables (labels). The goal is to train the algorithm so that in case values that are not in the training data are given as input, expected output values are produced. In the case of unsupervised learning, the goal is determining patterns in the data, e.g., to automatically organize input items in clusters (i.e., groups of objects that are similar with respect to some similarity function).

**ML model evaluation:** In this phase, the trained model is evaluated against a testing dataset. Values not in the training data are given as input to assess the model’s accuracy. The produced values are compared with the expected ones, organized in a purposely prepared ground truth. A number of evaluation metrics can be used to assess the accuracy of the model under analysis, including precision, recall, and success rate. If the obtained results are not satisfactory, there are different ways to intervene, including fine-tuning the parameters of the considered model, or moving back to the process, e.g., to improve the encoding of the data, select a different ML algorithm, or even refine or change the data under disposal.

**ML model deployment:** The trained model, which is satisfactory according to the previous evaluation step, is deployed to cloud or on-premises infrastructures to enable its adoption by the final users.

**ML model monitoring:** The deployed model is continuously monitored for possible errors during real-world execution or even to detect potential accuracy degradation that might induce the implementation of additional iterations of the process shown in Figure 1.

### 3 COMPARISON METHODOLOGY

This section presents a specific instance of the workflow shown in the previous section, adapted to automatically evaluate the suitability of different existing ML models to address the problem of model classification. Although this problem has been investigated over the last few years by the MDE community, no systematic comparison of different ML techniques and encodings has been performed yet.

The proposed methodology aims at comparing existing ML/MDE tools as well as new implementations, and consists of a specific instance of the workflow shown in Fig. 1. The methodology has been implemented in terms of a framework able to automatize the execution of the different tools under analysis. The existing approaches that have been analyzed in this paper are MAR [23, 24], a custom implementation of Lucene [34] specifically designed to classify metamodels, AURORA [28, 29] and MEMOCNN [30]. The rationale behind this selection is that these tools are based on solid ML techniques and demonstrated effectiveness in the metamodels classification. MAR is a search engine designed to work with models; nevertheless, the authors tested the capability of MAR to classify metamodels. The implementation of Lucene is based on the well-known Apache search engine and represents an attempt to employ a general-purpose tool to classify metamodels. AURORA is one of the first experiments to classify metamodels by exploiting feed-forward neural networks. Instead of relying on the original AURORA tools, we reimplemented the FFNN of AURORA because we could not integrate its tools that depend on external cloud-based technologies. Finally, MEMOCNN is the first attempt that represents metamodels using matrices and exploits a convolutional neural network to classify them. Additionally, we have implemented new classifiers based on Support Vector Machine, k-Nearest Neighbours, Naive Bayes Models, and Graph Neural Networks.

The framework automatizes and supports the activities shown in Fig. 2 and described below.

**Data gathering.** We have directly used an existing dataset, `ModelSet`, instead of creating a new one to address this step. `ModelSet` provides 5,000 Ecore models and 5,000 UML models, labelled with a category that reflects the application domain of the model (e.g., an Ecore model representing a state machine or a UML model describing the behaviour of an ATM). We use the provided Python library\(^1\) to load and handle the labels of the dataset consistently.

**Data preparation.** To perform fair comparisons, we have pre-processed the original models as follows.

- **Detect duplicates.** We have observed that there are a large number of quasi-duplicate models for some categories of models. This seems to be due to the fact that `ModelSet` was built by gathering models from public repositories (GitHub and GenMyModel) in which users tend to copy-paste models. This may introduce a bias in the evaluation of the methods. Therefore, we have adapted to models the method to detect duplicates in program files devised by Allamanis in [1]. This pre-processing step is optional since we will perform experiments with duplicates and without them to compare the effect of duplication.

- **Filter.** Another issue is that some categories contain few models. Therefore, we filter out those categories with less than ten models as discussed and motivated later in the paper.

- **Prepared dataset.** After detecting duplicates and filtering categories, we obtain the list of models that we use for the evaluation. Finally, we apply this process to both UML and Ecore and save the list in a separate file to ensure that the evaluation of the different methods consistently uses the same models.

\(^1\)http://github.com/modelset/modelset-py
4 COMPARISON DIMENSIONS

The comparison of existing techniques and tools for model classification has been performed by considering two different aspects, i.e., the adopted ML algorithm (see Section 4.1) and the way software models are encoded (see Section 4.2). We developed a set of tools to automate the methodology presented in the previous section and thus simplify the investigation of all the possible encoding and ML algorithm combinations.

The considered combinations of model encodings and ML models are shown in Table 1 and graphically depicted in Fig 3. The choice of these combinations was driven by the compatibility of each encoding with the ML model (e.g., Naive Bayes models cannot handle word embedding) and the techniques used in previous works about the application of ML to MDE (column INSPIRED BY/ADAPTED FROM). Specifically, there are two main families of representations i.e., BoW (Bag-of-Words) and graph. The former encodes the terms of the model (e.g., the names associated with the model elements), and the latter encodes the structure of the model plus some attributes of the model elements.

4.1 ML models

In this paper, we will focus on classification problems. Thus, in the rest of the section, we will present several ML algorithms used to solve this problem. From now on, we will assume that we have a dataset of pairs \( \mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n} \) where \( x_i \) is the input data and \( y_i \) is its associated label.

4.1.1 \( k \)-Nearest Neighbors (KNN)

This one of the simplest classification algorithms. Given new data \( x \), its \( k \) most similar/closest elements of the dataset \( \mathcal{D} \) are computed. Then, the label of the majority is assigned to \( x \). The flexibility of this technique is given by the concept of similarity that has to be defined by the user.

An advantage of using \( k \)-NN is that one can use a well-known measure (e.g., euclidean distance) or create a user-defined one. In
Figure 3: Input features and ML models.

particular, in this paper, we will use search engines to take advantage of their capabilities to compare and rank models according to their similarity to an input query. In our experiments, we consider $k$-NN using the following three different similarity measures (the first one is typically used in $k$-NN, whereas the other two are intended to evaluate the possibility of using off-the-shelf search engines to as a lightweight method to perform model classification):

- The euclidean distance over numeric vectors. In the next section we will show two strategies to encode models using a Bag-of-Words representation. This enables us to map software models to numeric vectors.
- The Lucene search engine. Lucene uses the scoring function Okapi BM25. It is a bag of words retrieval function that ranks a set of documents based on the query terms appearing in each document.
- The MAR search engine. MAR uses an adapted scoring function Okapi BM25 over BoPs. We use an approach similar to Lucene, that is, we use MAR as the similarity measure of $k$-NN by performing a query using the given model and retrieving the top $k$ more similar models.

4.1.2 Bayes Naive Classifiers (BNC). This is a simple ML technique that takes advantage of the independence assumption between the dimensions of the data. Given new data $x$, we want to assign it the label $l$ such that

$$
\text{argmax}_l P(y = y_l | x). 
$$

Using the Bayes theorem, we have the following:

$$
P(y = y_l | x) = \frac{P(y_l)P(x | y_l = l)}{P(x)}. 
$$

If we replace this formula into (2) and taking into account that $P(x)$ does not depend on $l$, we can transform (2) into

$$
\text{argmax}_l P(y_l) \prod_{j=1}^{K} P(x^j | y = l) 
$$

Now, using the indepence assumption of the components, $P(x | y = l) = \prod_{j=1}^{K} P(x^j | y = l)$ and we have the following

$$
\text{argmax}_l P(y_l) \prod_{j=1}^{K} P(x^j | y = l) 
$$

where each $x^j$ is the $j$ component or dimension of $x$. Normally, $P(y_l)$ is estimated by counting the proportion of the different classes of the dataset $D$. Depending on how $P(x^j | y = l)$ is estimated using $D$, BNC are divided in several types. We will to consider the following ones:
- Gaussian Naive Bayes (GNB). The $P(x|y = l)$ distribution is assumed to be Gaussian.
- Multinomial Naive Bayes (MBN). In this case, a multinomial distribution is considered.
- Complement Naive Bayes (CNB). This is an adaptation of the MBN for unbalanced datasets.

4.1.3 Support Vector Machine (SVM). Let us assume that we are facing a binary classification problem i.e., there are only two distinct labels. This ML algorithm tries to find a hyperplane in the data space that divides the group of data points $x_i$ into two groups, one for each label. Given new data $x$, we just have to see in which side of the hyperplane it falls in. It is important to remark that this ML model has two important hyperparameters:
- The regularization term $C$ it is used to control the overfitting, affecting the number of missclassified points in the training set.
- The kernel used: sometimes it is not possible to find a separation hyperplane, and it is necessary to map these points to a higher dimensional space (kernel trick). The kernel is the dot product in this new dimensional space.

Finally, we will use the one-vs-one schema to extend this approach to multiclass classification. First, we train one SVM that separates $n$ classes into two groups, one for each label. Then, in each step (i.e., after the application of each layer), it is updated with the vectors of its neighborhood.

We have applied FFNN using BoW representations. Also, we have experimented with a GNN (which can directly use a graph representation) with two convolutional layers with the GraphSAGE operator [19] and a mean global pooling. Finally, we have also applied a CNN by encoding a model as a special BoW representation mapped to a 2D matrix to emulate an input image. The input image transits three convolutional layers. In each transition, a convolutional filter is applied together with a pooling procedure to decrease the size of the feature. Finally, the output of this pipeline is passed through two fully-connected layers to perform the classification.

4.2 Encoding of software models
A ML model cannot receive a raw software model as input, but it has to be transformed to a set of features which can be processed by the target ML model (e.g., a numeric vector). Therefore, an important task to use ML models with software models is to choose the proper encoding and implement it effectively. In the following, we present different encoding techniques with which we have experimented in this work.

Models as a Bags of Words (BoW). For each model, we extract the string attribute values and we split them taking into account the white space, snake case and camel case conventions. As a result, a multiset (bag) of words is obtained from an input model. For instance, the BoW associated to the Ecore model in Fig. 3 is \{fsm, states, state, target, source, transition, transitions\}. In this way, the vocabulary associated to the dataset is the set of words extracted from it. Finally, each bag is transformed into a real valued vector. For this, we have considered the following alternatives:

- **TF-IDF**. Each bag of words is represented by a high dimensional vector whose number of dimensions is the size of the global vocabulary (i.e., one dimension per different word). More concretely, each bag $b$ is transformed into a vector $v$ and the component associated to the word $i$ of the vocabulary is the following:

$$v_i = \text{tf}(b, i) \times \log \frac{1 + n}{1 + \text{df}(i)} + 1.$$  

Where $\text{tf}(b, i)$ is the number of times $i$ appears in $b$, $n$ is the number of bags and $\text{df}(i)$ is the number of bags that contain the term $i$. Finally, we normalize the vector $v$ by its norm. The final vector is a high-dimensional vector in which a zero means the absence of a term and the higher

---

2e.g., in the case of Ecore, the most notable attribute is $\text{ENamedElement}$.name.
the component associated to a concept is the stronger it is in the model. For instance, in the graphical representation of the TD-IDF encoding of the simple FSM metamodel in Fig. 3, the fsm term is represented with a lighter grey than the transitions.

- **Word embeddings.** A word embedding is a representation of text where each word has an associated vector, and words with similar meanings have similar vectors (i.e., geometrically close vectors). A valuable property of word embeddings is that once learned (from a certain collection of documents), it is possible to reuse them for several projects. Thus, there are several already trained word embeddings ready to be used. In this work, we consider the 300-d GloVe [31] embeddings trained with Wikipedia and Gigaword 5. Since each word already has a vector associated that encodes its meaning, we transform each bag into a vector by computing the average of the vectors associated with the words of the bag.

- **2D TF-IDF.** We follow the encoding technique used in [30] to encode the features. The TF-IDF vectors are split into sequences of consecutive cells and then they are stacked vertically to generate 2D matrices.

**Models as graphs.** Software models typically have a graph-like structure. Hence, this representation is particularly suitable to enable ML models to learn the structure of the models. In particular, we transform each model of the dataset into a graph following this procedure:

- Each object of the model is mapped to a node in the graph.
- Each reference of the model is mapped to a edge in the graph.
- Each object of the model is mapped to a node in the graph.
- Each node is given two attributes: an identifier for the object if available (e.g., a name attribute) and the name of its meta-class. In general, an object may have more than one attribute, but ML graph algorithms typically require a fixed number of attributes. Thus, we have chosen these two pieces of information as the most representative ones for classification. Nevertheless, other tasks may require the selection of different types of attributes.

If the target ML model is SVM (see Sect. 4.1.3), we consider graph kernels [14] and we can run SVM directly over the raw graphs. A graph kernel can be seen as a dot product between graphs in a high-dimensional vector space.

If the target ML model is GNN (see Sect. 4.1.4), the raw graph is introduced into the ML model. However, to adequately handle attribute values (which are strings), we need to map the vocabulary to numeric vectors. This is done in three steps. First of all, each word in the vocabulary is assigned a unique integer value. Then, the node attributes are converted into a numeric vector (with size 2 because we are considering only two attributes). Finally, the GNN requires an embedding layer to map the node attributes into a larger numeric vector which is learned during training and represents the meaning of the node.

Finally, we use MAR as an alternative means to encode the model structure. MAR is a search engine designed to perform query-by-example [23]. In MAR each model is encoded as bag-of-paths (BoPs), where paths are computed between attribute values. For instance, for the metamodel shown in Fig. 3, the path \((FSM, name, EClass)\) has length 2 and simply encodes the name of class. However, \((FSM, name, EClass, transitions, name, EStructuralFeature)\) has length 2 and encodes the existence of a reference named transition associated to the FSM. A configuration parameter of MAR is the maximum length of the paths. The longer the path is, the more model structure is encoded. In this work, we set the path length to 3, which is the standard value used by MAR (as reported in [24]).

5 EXPERIMENTS

In this section, we explain our experimental setup and report the results of our experiments. Through the performed experiments, we aim to answer the following research questions:

- **RQ1:** Which model achieves a greater performance in the task of model classification?
- **RQ2:** How the chosen encoding of software models affects the task of model classification?
- **RQ3:** Which is the effect of data duplication on the performance of the ML models?

The first research question aims at providing insights into which ML model to choose when faced with a classification task. Then, RQ2 complements RQ1 by studying the importance of the selected encoding in the performance of the models. Finally, RQ3 addresses the question of whether the results can be biased due to the presence of duplicates, which is a typical scenario when data is downloaded from public sources. It is important to remark that the experiments have been facilitated by the methodology explained in Sect. 3, instantiated and automated for the task of model classification.

5.1 Replication package

The results of the experiments and detailed instructions to run them from scratch are described in a replication package available at https://figshare.com/s/5904baec3dbd0a48036c.

The replication package is built using standard Python ML libraries like scikit-learn, PyTorch, Pandas, etc. Moreover, we provide a Python execution environment to make sure that the same dependencies are used. Finally, the results of the paper are automatically generated from the outputs of the experiments.

5.2 Data

We use ModelSet [22] as the source dataset to perform our experiments. It contains about 5,000 Ecore models and 5,000 UML models labelled with their categories and gathered from GitHub and GenMyModel respectively. The main reason for choosing it is that, currently, it is the largest dataset of labelled models available. In this work, we are interested in classification which is a supervised task and requires a labelled dataset. Although other datasets have been used in previous works, they are very small [3] which contains only 555 Ecore metamodels, and this hinders the possibility of obtaining reliable conclusions from its use. The Lindholmen dataset, instead, contains more than 90,000 UML models [33], but they are not labelled. Moreover, the dataset is not curated, and actually, many of the models are broken.

An important aspect to consider when using ModelSet is that it contains duplicate and quasi-duplicate models, which is a typical issue in datasets extracted from public sources like GitHub. Therefore, we have built an implementation of the algorithm proposed...
We evaluate how well the presented models are able to classify metamodels. Initially, we considered Accuracy, that is, the correctly classified models over the total models analysed:

\[
\text{Accuracy} = \frac{\text{Correctly classified models}}{\text{Total models}} \tag{3}
\]

To support our experiments we have developed a framework to easily plug-in new classification methods and evaluate them consistently. To this end, we have relied on the Python library sci-kit learn. To integrate a ML method in our framework, there are three main scenarios:

- Using an ML algorithm already supported by sci-kit. This is straightforward.
- Using an ML algorithm which can be implemented in Python. In this case, a scikit classifier needs to be created. This requires essentially implementing a training procedure (a `fit` method) and a prediction procedure (`predict` method). In this case, integration is straightforward as well, although additional work maybe need in the implementation. This is the case of our implementation of GNN for models which rely on Pytorch and PyG.
- Integrate a third-party tool, typically not implemented as a Python library. This is the case with the MAR and Lucene search engines and the MEMOCNN classifier. In this case, the strategy is similar to before. A scikit classifier is created, but it is much more complex since it needs to wrap the third-party tool by calling external programs and processing their results.

The column IMPL. in Table 1 shows which tool we have used to implement each ML method and encoding. We have implemented a TF-IDF encoding using the scikit facilities, word embeddings using...
Table 4: Results for Ecore, with duplicate models

<table>
<thead>
<tr>
<th>Model</th>
<th>Encoding</th>
<th>B. Accuracy</th>
<th>Best hyper.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFNN</td>
<td>TF-IDF</td>
<td>0.824972</td>
<td>hidden size = 150</td>
</tr>
<tr>
<td>SVM</td>
<td>TF-IDF</td>
<td>0.815609</td>
<td>kernel = linear, C = 10</td>
</tr>
<tr>
<td>GNN</td>
<td>Raw graph</td>
<td>0.807656</td>
<td>–</td>
</tr>
<tr>
<td>SVM</td>
<td>WordE</td>
<td>0.786898</td>
<td>kernel = linear, C = 100</td>
</tr>
<tr>
<td>k–NN</td>
<td>Lucene (BoW)</td>
<td>0.786793</td>
<td>k = 1</td>
</tr>
<tr>
<td>2D</td>
<td>TF-IDF</td>
<td>0.778440</td>
<td>–</td>
</tr>
<tr>
<td>FFNN</td>
<td>WordE</td>
<td>0.777899</td>
<td>hidden size = 150</td>
</tr>
<tr>
<td>k–NN</td>
<td>MAR (BoP)</td>
<td>0.775339</td>
<td>k = 3</td>
</tr>
<tr>
<td>CNN</td>
<td>2D TF-IDF</td>
<td>0.764505</td>
<td>k = 1</td>
</tr>
<tr>
<td>FFNN</td>
<td>WordE</td>
<td>0.777899</td>
<td>hidden size = 150</td>
</tr>
<tr>
<td>k–NN</td>
<td>MAR (BoP)</td>
<td>0.775339</td>
<td>k = 3</td>
</tr>
<tr>
<td>CNB</td>
<td>TF-IDF</td>
<td>0.723883</td>
<td>k = 1</td>
</tr>
<tr>
<td>MNB</td>
<td>TF-IDF</td>
<td>0.716409</td>
<td>α = 0.1</td>
</tr>
<tr>
<td>SVM</td>
<td>Graph kernel</td>
<td>0.758644</td>
<td>α = 0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.754884</td>
<td>α = 0.1</td>
</tr>
</tbody>
</table>

Table 5: Results for Ecore, removing duplicate models

<table>
<thead>
<tr>
<th>Model</th>
<th>Encoding</th>
<th>B. Accuracy</th>
<th>Best hyper.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFNN</td>
<td>TF-IDF</td>
<td>0.828472</td>
<td>hidden size = 150</td>
</tr>
<tr>
<td>SVM</td>
<td>TF-IDF</td>
<td>0.815609</td>
<td>kernel = linear, C = 10</td>
</tr>
<tr>
<td>GNN</td>
<td>Raw graph</td>
<td>0.807656</td>
<td>–</td>
</tr>
<tr>
<td>SVM</td>
<td>WordE</td>
<td>0.786898</td>
<td>kernel = linear, C = 100</td>
</tr>
<tr>
<td>k–NN</td>
<td>Lucene (BoW)</td>
<td>0.786793</td>
<td>k = 1</td>
</tr>
<tr>
<td>2D</td>
<td>TF-IDF</td>
<td>0.778440</td>
<td>–</td>
</tr>
<tr>
<td>FFNN</td>
<td>WordE</td>
<td>0.777899</td>
<td>hidden size = 150</td>
</tr>
<tr>
<td>k–NN</td>
<td>MAR (BoP)</td>
<td>0.775339</td>
<td>k = 3</td>
</tr>
<tr>
<td>CNN</td>
<td>2D TF-IDF</td>
<td>0.764505</td>
<td>k = 1</td>
</tr>
<tr>
<td>FFNN</td>
<td>WordE</td>
<td>0.777899</td>
<td>hidden size = 150</td>
</tr>
<tr>
<td>k–NN</td>
<td>MAR (BoP)</td>
<td>0.775339</td>
<td>k = 3</td>
</tr>
<tr>
<td>CNB</td>
<td>TF-IDF</td>
<td>0.723883</td>
<td>k = 1</td>
</tr>
<tr>
<td>MNB</td>
<td>TF-IDF</td>
<td>0.716409</td>
<td>α = 0.1</td>
</tr>
<tr>
<td>SVM</td>
<td>Graph kernel</td>
<td>0.758644</td>
<td>α = 0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.754884</td>
<td>α = 0.1</td>
</tr>
</tbody>
</table>

Table 6: Results for UML, with duplicate models

<table>
<thead>
<tr>
<th>Model</th>
<th>Encoding</th>
<th>B. Accuracy</th>
<th>Best hyper.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFNN</td>
<td>WordE</td>
<td>0.873906</td>
<td>kernel = linear, C = 10</td>
</tr>
<tr>
<td>SVM</td>
<td>TF-IDF</td>
<td>0.872578</td>
<td>hidden size = 150</td>
</tr>
<tr>
<td>SVM</td>
<td>TF-IDF</td>
<td>0.870490</td>
<td>kernel = linear, C = 100</td>
</tr>
<tr>
<td>FFNN</td>
<td>TF-IDF</td>
<td>0.861412</td>
<td>hidden size = 100</td>
</tr>
<tr>
<td>k–NN</td>
<td>MAR (BoP)</td>
<td>0.859487</td>
<td>k = 1</td>
</tr>
<tr>
<td>k–NN</td>
<td>WordE</td>
<td>0.849309</td>
<td>k = 1</td>
</tr>
<tr>
<td>k–NN</td>
<td>TF-IDF</td>
<td>0.847727</td>
<td>k = 1</td>
</tr>
<tr>
<td>GNN</td>
<td>Raw graph</td>
<td>0.843559</td>
<td>–</td>
</tr>
<tr>
<td>GNB</td>
<td>TF-IDF</td>
<td>0.798754</td>
<td>–</td>
</tr>
<tr>
<td>SVM</td>
<td>Graph kernel</td>
<td>0.769627</td>
<td>C = 0.1</td>
</tr>
<tr>
<td>CNB</td>
<td>TF-IDF</td>
<td>0.760579</td>
<td>α = 0.1</td>
</tr>
<tr>
<td>MNB</td>
<td>TF-IDF</td>
<td>0.745817</td>
<td>α = 0.1</td>
</tr>
</tbody>
</table>

Table 7: Results for UML, removing duplicate models

<table>
<thead>
<tr>
<th>Model</th>
<th>Encoding</th>
<th>B. Accuracy</th>
<th>Best hyper.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFNN</td>
<td>WordE</td>
<td>0.775893</td>
<td>hidden size = 150</td>
</tr>
<tr>
<td>SVM</td>
<td>TF-IDF</td>
<td>0.758389</td>
<td>hidden size = 50</td>
</tr>
<tr>
<td>SVM</td>
<td>WordE</td>
<td>0.756125</td>
<td>kernel = rbf, C = 100</td>
</tr>
<tr>
<td>SVM</td>
<td>TF-IDF</td>
<td>0.744753</td>
<td>kernel = linear, C = 10</td>
</tr>
<tr>
<td>k–NN</td>
<td>MAR (BoP)</td>
<td>0.716452</td>
<td>k = 3</td>
</tr>
<tr>
<td>GNN</td>
<td>Raw graph</td>
<td>0.713418</td>
<td>–</td>
</tr>
<tr>
<td>CNB</td>
<td>TF-IDF</td>
<td>0.703389</td>
<td>α = 0.1</td>
</tr>
<tr>
<td>k–NN</td>
<td>WordE</td>
<td>0.694383</td>
<td>k = 1</td>
</tr>
<tr>
<td>k–NN</td>
<td>TF-IDF</td>
<td>0.686937</td>
<td>k = 1</td>
</tr>
<tr>
<td>GNB</td>
<td>TF-IDF</td>
<td>0.630084</td>
<td>–</td>
</tr>
<tr>
<td>MNB</td>
<td>TF-IDF</td>
<td>0.628251</td>
<td>α = 0.1</td>
</tr>
<tr>
<td>SVM</td>
<td>Graph kernel</td>
<td>0.530019</td>
<td>C = 0.1</td>
</tr>
</tbody>
</table>

6.1 RQ1: Best model

The best models are feed-forward neural network (FFNN) and SVM in the four scenarios (Ecore and UML with and without duplicates). Moreover, FFNN outperforms the SVM for 3 out of the 4 scenarios. Using a “lightweight method” like k–NN plus MAR (or Lucene) obtains consistently worse results than more advanced ML methods, particularly neural networks and SVMs. However, the loss of accuracy when using MAR or Lucene is not very high, which means that these models are still competitive. This suggests that the classification tasks could be effectively addressed with simple, off-the-shelf machinery. We foresee two scenarios in which lightweight methods can be a good fit: a) for teams with little experience with ML and the additional complexity that it brings for testing and deploying the ML models and b) when there already exist search engines with a large number of indexed models, and thus, it is possible to profit from them with little effort.

Finally, it is worth mentioning that FFNN and SVM, which can be considered simple ML models, outperform more complex models like CNN and GNN which can be considered deep learning models since they have much more learnable parameters. One reason could be that the dataset size is not large enough for the usage of complex models.

6.2 RQ2: Encoding schemes

In the case of Ecore, the best encoding scheme is TF-IDF. In Tables 4 and 5 the first two models use as encoding technique TF-IDF. Interestingly, TF-IDF outperforms word embedding, which is a more sophisticated encoding. We believe that this can be caused by the...
fact that we use a word embedding model trained on general texts, which may not be a good fit for software models. Thus, a potential future direction is to train word embeddings based on modelling texts.

On the other hand, for UML, the best encoding is word embedding. This confirms our previous argument. The rationale, in this case, is that UML models are often used to describe non-technical domains: e.g., a shopping center, a bank, etc. Therefore, the pre-trained embeddings capture the meaning of the model elements much better.

In the same line, the performance of the GNN drops for UML (in relative terms with respect to FFNN and SVM). This can be caused by the fact that the embedding layer is simply initialized with the vocabulary. A potential solution to improve the GNN performance for UML could be to initialize the embedding layer directly with word embeddings.

An interesting observation is related to the choice between simple or more complex encodings. One could expect that structured encoding schemes based on graphs (as used by MAR and models like GNN and SVM with kernel) should be superior. The rationale is that they are a good match for the graph-based nature of software models. However, our experiments reject this idea for the classification task. The main reason that we have observed is that the domain of application of a software model is mainly associated with the names given to its model elements. For example, most meta-models about Petri nets have words like petri, net, place, transition, arc, etc and the main difference with e.g., a state machine model is not the structure but the naming.

Thus, for ML tasks similar to the one in our experiments, possibly a simple encoding scheme based on extracting the strings is enough.

6.3 RQ3: The effect of (near) duplication

There is an important reduction in the accuracy of all the methods when duplicates are discarded. As expected, the type of model whose performance decays the most is the $k$−NN. This model is considered a naive memorization method [1]. If a model of the training set is duplicated in the test set, likely, this model appears first in the list of nearest neighbors.

Previous works [29] have obtained similar results using a feed-forward neural network, with a similar TF-IDF encoding (i.e., precision above 90%). However, our results show that these experiments could be biased due to the existence of duplicates. These results suggest that future works applying ML for MDE tasks should carefully consider the presence of duplicates.

Arguably, having duplicates in the dataset may reflect well scenarios in which the data has been obtained from public repositories (i.e., ModelSet was built from GitHub and GenMyModel). Nevertheless, our results show that there is still room to improve classification methods applied to software models.

6.4 Assessment

Our experiments have shown that the performance of the tested ML models (which are standard models) for the task of model classification is good (around 80% when duplicates are removed) but not excellent. This contrasts with previous works [16, 22, 28, 30], the reason can be due to a variety of factors that makes the classification harder for the ML models, namely: the fact that we have used a larger dataset (with more categories), the removal of duplicates, and the use of UML models, which may be more complex than Ecore models. In practice, this means that model classification is not yet a solved problem. Instead, more research is needed to propose specific ML models to obtain better performance.

The internal threats to our approach are essentially related to the dataset we adopted. We identified some specific threats: ModelSet is composed of ~5000 Ecore and UML models, which could not be enough to train some ML models properly, but its size is enough to draw initial results. Moreover, to the best of our knowledge, it is the largest curated and labeled dataset publicly available. Another possible threat is related to the quality of ModelSet, the existence of duplicates is a relevant problem which affects different datasets. We alleviate this problem by considering two versions of the original dataset, with and without duplicates. Some of the employed tools are not able to classify UML models. We plan to extend these tools to analyze their performance in this task. An external threat that affects the generalization of the results is that we have only considered one kind of ML task. To partially mitigate this, we have performed the task with two with both Ecore and UML. Nevertheless, the proposed framework can boost the experimentation with other relevant ML tasks in the context of MDE.

7 CONCLUSION AND FUTURE WORK

In the last years, the application of ML techniques to enhance MDE-related scenarios has been increasingly researched. However, the current body of knowledge lacks a good understanding of which ML models and encodings are best suited. This paper aims to fill this gap, focussing on the task of model classification. To this end, we have considered six ML models and six possible encodings, and we have systematically combined and evaluated them. Finally, we applied these two datasets of about 5,000 Ecore models and 5,000 UML models. The results show that a feed-forward neural network using a TF-IDF encoding is generally the best approach. However, simpler models like $k$-NN combined with a search engine provide almost comparable results. In general, for model classification, the model structure is irrelevant (i.e., it is not needed to use more complex models like GNNs). Moreover, we have shown that the performance of all ML models is reduced when (quasi-)duplicate models are discarded.

In future works, we plan to apply this framework to other ML tasks in the MDE domain to generalise our results. Regarding extensions for the task of model classification, we plan to consider multi-classification as done in [22] (e.g., multiple labels per model) as well as using other performance metrics. We also plan to investigate other ML models which can generalize better and improve the accuracy of the tested models.

Acknowledgments. Work partially supported by the EMELIOT national research project, which has been funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY). Jesús Sánchez Cuadrado enjoys a Ramón y Cajal 2017 grant (RYC-2017-237) funded by MINECO (Spain) and co-funded by the European Social Fund. José Antonio Hernández López enjoys a FPU grant funded by the Universidad de Murcia.
REFERENCES


