Model Selection Using Graph Neural Networks

Model Selection Using Graph Neural Networks

Gonzalo Nápoles[1(✉)], Isel Grau[2,3], Çiçek Güven[1], and Yamisleydi Salgueiro[4]

[1] Department of Cognitive Science and Artificial Intelligence, Tilburg University, Tilburg, The Netherlands
g.r.napoles@uvt.nl

[2] Information Systems Group, Eindhoven University of Technology, Eindhoven, The Netherlands

[3] Eindhoven Artificial Intelligence Systems Institute, Eindhoven University of Technology, Eindhoven, The Netherlands

[4] Department of Industrial Engineering-Faculty of Engineering, Universidad de Talca, Campus, Curicó, Chile

Abstract. This paper tackles the problem of selecting the optimal models (algorithms and their hyperparameters) for a structured classification problem using Graph Neural Networks (GNNs). Recent efforts in this direction associate statistical meta-features describing the problem with the performance of predefined models. However, the predictive power of these meta-features is insufficient while being expensive to compute. The approach presented in this paper encodes each problem as a granular knowledge graph where nodes denote prototypes, while edges capture their distance. Moreover, nodes are labeled with the most popular class in their neighborhood, and their quality is quantified with a purity score. The adjacency-based representations of these knowledge graphs establish positive arrows between close prototypes that belong to different decision classes. Therefore, solving the multilabel model selection problem consists of predicting the set of optimal models for a given dataset represented by its adjacency-based matrix knowledge graph. The results indicate that the proposed GNN-based meta-classifier can predict an optimal model for 92% of the datasets, suppressing the need to extract low-level features.

Keywords: Multilabel model selection problem · Graphical neural networks · Meta-classifier · Meta-features

1 Introduction

The algorithm selection problem was first introduced in Ref. [19] and continues to be relevant in machine learning. This task involves creating a model that can predict the best-performing algorithm based on the properties of the data. Algorithm selection techniques [9,13] usually rely on supervised meta-models that link meta-features to candidate algorithms. These meta-features can range from

K. Arai (Ed.): IntelliSys 2024, LNNS 1066, pp. 332–347, 2024.
https://doi.org/10.1007/978-3-031-66428-1_20

simple statistics, such as the number of features and instances, to information-theoretic measures, like mutual information and landmark features.

Algorithm selection is a crucial component of Automated Machine Learning (AutoML), which encompasses various tasks like data cleaning, feature selection, hyperparameter optimization, and result post-processing. For hyperparameter tuning, for example, constrained Bayesian optimization, as described in Ref. [10], is the most commonly used approach due to its effectiveness in handling memory and time constraints. A variety of AutoML frameworks, such as AutoWEKA [14], Auto-sklearn [6], Auto-PyTorch [31], and AutoKeras [11], play a significant role in automating these tasks.

Although several approaches concerning algorithm selection and hyperparameter tuning are reported in the literature, some major challenges persist. In general, AutoML frameworks are criticized as environmentally unfriendly due to their high consumption of resources [23]. Regarding the dataset representation, existing meta-feature methods are generally expensive to compute with a limited budget [6]. In addition, meta-features often hide relevant information as they aggregate negative and positive values that might cancel each other or are not defined for certain types of features, such as categorical ones. On the other hand, non-brute force hyperparameter optimization continues to be a computationally expensive process involving optimizing either a fitness function, an acquisition function, or building a surrogate model. Furthermore, for a given problem, there could be several optimized models reporting the highest performance, yet correctly recognizing one of them would solve the model selection problem.

In this paper, we propose to extend the algorithm selection problem with a set of hyperparameter values and approach this as a *multilabel model selection problem*. Let $\Omega = \{\Omega_1, \ldots, \Omega_o\}$ be a set of classification algorithms with hyperparameters $\Delta = \{\Delta_1, \ldots, \Delta_o\}$ such that Δ_l is the set of hyperparameters for the l-th classifier. Moreover, let us define $\delta_l^q \in \Delta_l$ as the q-th hyperparameter and $\mathcal{D}(\delta_l^q)$ as its domain. The set of all settings for the l-th classifier is $\Pi_l = \times_q \mathcal{D}(\delta_l^q)$ while the set of all models is given by $\Pi = \bigcup_l \Omega_l \times \Pi_l$. Observe that a model can be understood as a classifier using a specific hyperparameter setting. Therefore, the multilabel model selection problem consists of predicting which models will yield the best performance possible w.r.t. a given metric for an unseen classification dataset. Afterward, the model selection problem consists of selecting $\pi_l \in \Pi$ with the highest likelihood of having the highest performance among all explored models.

As the main contribution of this paper, we introduce a novel representation of the datasets as a granular knowledge graph. The decision space of datasets is represented as graphs where nodes denote prototypes of data points while edges capture their distance. Nodes also contain information about the purity of the cluster represented by the prototype. This representation captures information about the decision space, such as which regions are easier or difficult to separate, based on the similarity of instances. Extracting an adjacency matrix from the graph representation allows us to leverage the predicting power of graph neural networks (GNN) as a meta-classifier. Therefore, we propose a GNN-based approach for addressing the multilabel model selection problem. The experimental evaluation reveals that combining the graph representation with the GNN-based meta-classifier outperforms state-of-the-art meta-classifiers using meta-features for several performance measures. In particular, the GNN-based metaclassifier can predict an optimal model for 92% of the 9889 datasets studied.

The remainder of this paper is organized as follows. Section 2 revises the state-of-the-art concerning meta-learning approaches, with emphasis on alternative representations. Section 3 presents the proposed knowledge graph representation, the multilabel data generation, and the GNN-based metaclassifier. Section 4 conducts numerical simulations involving synthetically generated datasets and compares our approach against meta-classifiers using meta-features. Finally, Sect. 5 summarizes our method's results and limitations while discussing recommendations to be addressed in future studies.

2 Related Work

Several approaches have been proposed to address the algorithm selection problem, with the most common involving mapping statistical meta-features to candidate algorithms. Recent survey studies on meta-features [20,21,24] classify them as general, statistical, information-theoretic, model-based, landmarking, and others. To standardize meta-feature selection, Ref. [18] propose a systematic framework for generation. Furthermore, a new development is the meta-feature extractor package introduced by Ref. [1]. The characterization and standardization of these measures facilitate reproducible comparisons in meta-learning experiments [1,21]. Nevertheless, existing meta-feature methods are generally computationally expensive to obtain [6], hampering the already costly AutoML solutions.

Another characteristic that differentiates meta-learning solutions is the algorithm used as the meta-learner. Several works attempt to leverage the similarities of a given problem with the ones encoded in the meta-datasets by using lazy learners such as kNN [4,12,22,25]. In these approaches, the similarity is computed based on meta-features over a meta-dataset of numerous problems. For example, the authors in Ref. [22] use 84 datasets and 17 algorithms to provide a ranked list of the best-performing algorithms for similar meta-instances. This approach maps the top nearest neighbors to the algorithm's space to obtain the ranked list based on a performance measure. The main advantage of a lazy meta-learner such as kNN is its extensibility, as new meta-instances can be incorporated without retraining the meta-learning model. However, the results of the meta-learning method are sensitive to the choice of the number of neighbors [30]. Other approaches, such as decision trees, have been explored as meta-learners [2,3], although they normally underperform compared to lazy approaches. A notable observation is that in these approaches the solution often focuses solely on algorithm selection, overlooking the consideration of hyperparameters, which are addressed separately.

Studies using alternative representations to meta-features are more scarce and mostly concentrated on learning the entire machine learning pipeline rather than algorithm selection. For example, in Ref. [5] the authors use natural language models to obtain embeddings from the textual descriptions of the datasets and the machine learning pipelines. Similarly, Ref. [7] represents the entire dataset with column-level and table-level embeddings from a pre-trained language model. Interestingly, they model pipeline creation as a graph generation problem. This means that the graph neural network is used for generating the optimal pipelines for a given dataset embedding, which contrasts with our approach, where the graph is used for representation.

A final drawback identified in the reviewed literature on meta-learning experiments is the limited number of datasets used. Currently, there is no clear guideline regarding the optimal number of datasets to be considered [13]. However, we argue that synthetic data generation can help augment the number of training instances, thus leading to improved results. To this end, we identified several synthetic data generation approaches available in the literature for generating tabular data [8,15,26,27].

3 Proposed Method

The proposed AutoML method consists of creating a GNN-based meta-classifier to solve the multilabel model selection problem defined above. That means the GNN will receive a structured pattern classification dataset and output a binary vector denoting the models that will perform well for that problem. Moreover, the GNN-based meta-classifier will return the confidence values of this classification process, allowing us to select the most likely optimal model among all those regarded as optimal.

3.1 Granular Knowledge Graphs

Let's assume that we have a set of classification problems $P = \{p_1, \ldots, p_k, \ldots, p_m\}$ to be used to build the meta-classifier, where, the k-th problem is described by a set of features F_k and associated with a set of decision classes D_k. The primary novelty of our proposal is that it encodes each problem $p_k \in P$ as a granular knowledge graph $G_k = (V_k, E_k)$ where V_k is the set of n nodes and E_k is the set of $n \cdot (n-1)$ edges. Each node $v_i \in V_k$ denotes an information granule (or prototype), while the edge $e_{ij} \in E_k \subseteq V_k \times V_k$ gives the distance between the i-th and j-th prototypes.

The first step towards creating such a granular knowledge graph is to gather the available data into n clusters, which produces a hard partition of the universe of discourse $X = \{C_1, \ldots, C_i, \ldots, C_n\}$ where C_i is the i-th cluster. It holds that $\cup_i C_i = X$ and $\cap_i C_i = \emptyset$. Afterward, we reduce each cluster to a prototype as follows:

$$c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x \tag{1}$$

where c_i is the i-th prototype for cluster C_i and $|\cdot|$ stands for the set cardinality operator. Therefore, $e_{ij} = e_{ji}$ can be computed as $||c_i - c_j||_2$. Distances can be normalized by dividing by the maximum edge value once the whole graph is built to improve the graph's interpretability.

Nodes in the graph are associated with the attribute functions that give further information about the clusters and prototypes. The label function $g : V_k \rightarrow D_k$ gives the most popular decision class observed in a cluster, where D_k is the set of decision classes for the k-th pattern classification problem being encoded. For example, $g(v_1) = c_2$ means that the first cluster is dominated by the second decision class, which will then be used to label that node in the granular knowledge graph. The purity function $h : V_k \rightarrow [0, 1]$ gives the purity score of a node, which is computed as the ratio between the frequency of the most popular class within a cluster and its cardinality. For example, if a cluster is supported by 10 instances and 8 of them belong to the same decision class, then that node will have a purity score of 0.8.

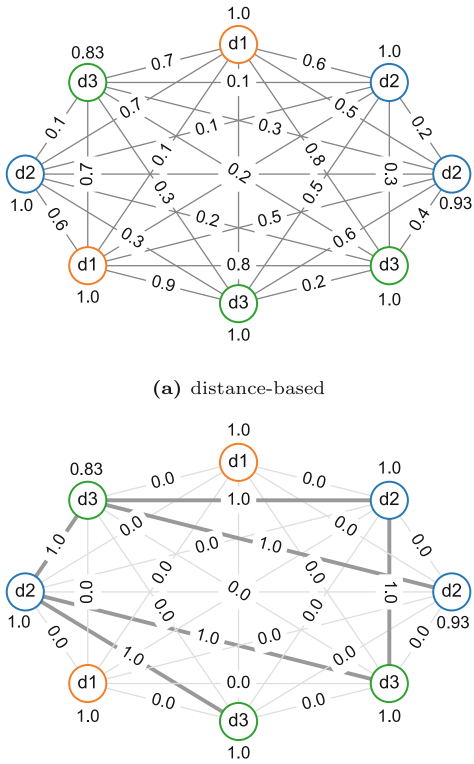
As mentioned, the purpose of transforming the pattern classification problems into granular knowledge graphs is to build a GNN as the meta-classifier. These neural systems usually expect the graph data to be described with adjacency matrices. Aiming to obtain the adjacency matrices, we can rely on the label function $g(\cdot)$ and the edge information. If two prototypes are close to each other and have different decision classes associated with them according to the label function, then such an edge will be transformed into a positive arrow. Relationships that connect nodes with the same labels or that are far away are transformed into negative arrows. Equation (2) shows how to compute the edge values in the adjacency-based granular knowledge graph,

$$e_{ij} = \begin{cases} 0 \; g(v_i) = g(v_j) \vee ||c_i - c_j||_2 > \lambda \\ 1 \; g(v_i) \neq g(v_j) \wedge ||c_i - c_j||_2 \leq \lambda \end{cases} \tag{2}$$

where λ represents a user-specified parameter. The larger the value of this parameter, the sparser the adjacency matrix that defines the relationships between prototypes. For convenience, such a parameter can be defined in terms of the q-th percentile (e.g., the threshold value is the edge value corresponding with the 75th percentile).

Figure 1 portrays the distance-based and adjacency-based knowledge graphs describing the Iris dataset. In this example, we used 8 prototypes to keep the graphs readable, while the distance threshold to determine that two prototypes are close is given by the 50th percentile.

These structures include valuable insights into the problem's decision space. For example, the fact that the far-left node is located close to two other pure prototypes labeled with a different decision class indicates that these nodes partially define the decision space in that region. Hence, a classifier might struggle to separate the instances contained in these clusters. In contrast, the fact that the top node is pure and clearly separated from the remaining ones indicates that a classifier could easily recognize the instances it represents. In summary, while the node labels and purity scores give information about the problem's



(a) distance-based

(b) adjacency-based

Fig. 1. Distance-based and adjacency-based granular knowledge graphs for the Iris dataset. In the second graph, thick arrows denote relationships that connect prototypes dominated by different decision classes

instance-level decision space, the node labels and edge values give information about its prototype-level decision space.

The number of prototypes is another parameter to be defined by the user. The more prototypes we have, the larger the knowledge graphs describing the pattern classification problems. While this might increase the power of the GNN-based meta-classifier, it also increases the risk of overfitting. Ultimately, we might not need an extremely detailed representation of the problem's decision space but educated insights about its complexity. Figure 2 plots the decision space of a Naïve Bayes classifier for different numbers of prototypes extracted from the Iris dataset. This example shows that the decision space remains similar across all configurations, even when having fewer prototypes might lead to knowledge representations that hide misclassifications. However, this information will be captured by the purity scores associated with graph nodes.

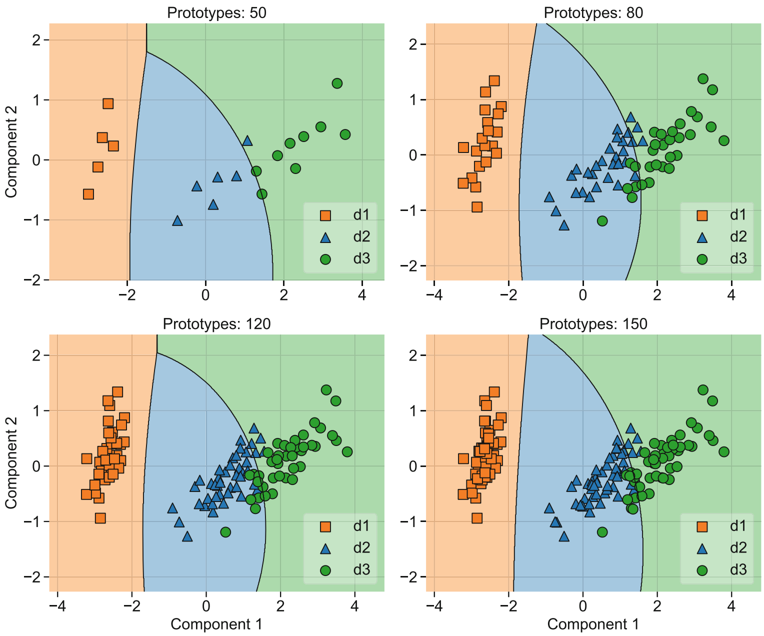


Fig. 2. Decision space of a Naïve Bayes classifier for different numbers of prototypes extracted from the Iris dataset

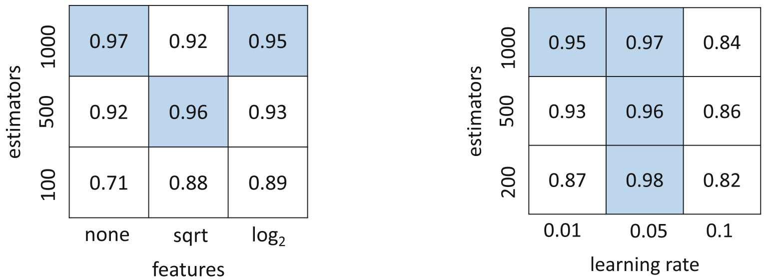
It is worth mentioning that GNNs can handle graph instances with different shapes (as expressed by their number of nodes and edges). That means we can have a problem described by 100 prototypes and another by 50, thus equipping our proposal with further flexibility.

3.2 Creating the Multilabel Targets

After building the graphs, we need to compute the labels encoding models' performance on each problem. In other words, the graph $G_k = (V_k, E_k)$ describing the k-th problem will be associated with a binary vector L_k with the target labels. These vectors have as many dimensions as models (given by $|\Pi|$), so positive labels mark models performing well for that problem. To do that, each problem p_k is tested on a set of models and hyperparameters using a grid search cross-validation. The best-performing models (determined using a performance threshold) are positively labeled, while those that do not reach the optimal performance are negatively labeled.

Figure 3 exemplifies this procedure for two hyperparameter grids concerning Random Forest (RF) and Light Gradient Boosting Machine (LGBM). Each algorithm varies two hyperparameters over three values, leading to 18 models that are evaluated using Cohen's kappa score as the performance metric. Those highlighted in blue are positively labeled as their kappa scores are within a 1%

difference from the best score value in both grids. The remaining models are negatively labeled since their performance falls below the specified threshold.



(a) Random Forest **(b)** LGBM

Fig. 3. Hyperparameter performance grid for (a) Random Forest and (b) Light Gradient Boosting Machine. In this example, each algorithm varies two hyperparameters over three possible values, totaling 18 models

4 Numerical Simulations

In this section, we cover the aspects concerning the generation of synthetic classification problems, the hyperparameter tuning aspects, and the comparison against other algorithms that use traditional meta-features.

4.1 Generating Classification Problems

Naturally, training a GNN requires a significant amount of data that might not be available. To generate synthetic pattern classification problems with varying complexity, we will use the `make_classification` function from the `Sklearn` library. This function generates normally distributed data points around the vertices of a hypercube where the length of its sides is controlled by the `class_sep` parameter. The function parameters are randomly selected from arbitrarily-defined intervals as follows. The number of instances $\texttt{n\_samples} \sim U_{\mathbb{N}}(500, 5000)$ and the number of decision classes $\texttt{n\_classes} \sim U_{\mathbb{N}}(2, 20)$[1]. The number of problem features $\texttt{n\_features} = r_f \times \texttt{n\_classes}$, where $r_f \sim U_{\mathbb{N}}(5, 10)$, whereas the number of informative features $\texttt{n\_informative} = \lfloor r_{fi} \times \texttt{n\_features} \rfloor$ where $r_{fi} \sim U_{\mathbb{R}}(0.2, 0.4)$. The number of redundant features $\texttt{n\_redundant} = \lfloor r_{fe} \times (\texttt{n\_features} - \texttt{n\_informative}) \rfloor$ where $r_{fe} \sim U_{\mathbb{R}}(0.2, 0.4)$. Similarly, the number of repeated features $\texttt{n\_repeated} = \lfloor r_{fr} \times (\texttt{n\_features} - \texttt{n\_informative} -$

[1] In this paper, the operation of randomly generating real and integer values in the (a, b) interval we will be denoted as $x \sim U_{\mathbb{R}}(a, b)$ and $y \sim U_{\mathbb{N}}(a, b)$, respectively.

n_redundant)$\rfloor$ where $r_{fr} \sim U_{\mathbb{R}}(0.2, 0.4)$. It shall be mentioned that parameters r_{fi}, r_{fe}, and r_{fr} represent ratios and $\lfloor . \rfloor$ represents the floor operation.

Concerning the remaining parameters, the number of clusters per decision class n_clusters_per_class $\sim U_{\mathbb{N}}(1, 5)$ and the proportions of samples assigned to each decision class weights $\sim U_{\mathbb{R}}(0.2, 0.8)$, which creates problems with different imbalance ratios. It must hold that n_clusters_per_class $<$ $\lfloor 2^{fi}/$n_classes$\rfloor$ such that $fi =$ n_informative. The flip_y parameter takes values in $[0.01, 0.05]$, denoting the fraction of samples with randomly assigned decision classes. Larger values introduce noise in the labels and make the classification task harder. The hypercube parameter determines whether the clusters are allocated at the vertices of a hypercube or a random polytope. Finally, class_sep $\sim U_{\mathbb{N}}(0.5, 1.5)$ is a factor multiplying the hypercube size where larger values spread out the clusters, thus making the classification problem easier on average.

This function allowed us to generate 9889 classification problems evaluated on 24 different models, as explained in the previous subsection. These models concern two classifiers devoted to structured pattern classification problems: Random Forest (RF) and Light Gradient Boosting Machine (LGBM). In the case of RF, the hyperparameters to be considered are the information criterion (gini or entropy), the strategy for selecting the maximum number of features (sqrt, log2 or None) and the number of estimators (500 or 1000). In the case of LGBM, the hyperparameters to be considered are boosting type (gbdt or dart), the learning rate (0.1, 0.05, 0.01), and the number of estimators (500 or 1000). That means that 237,336 models were built. Finally, we further enlarged the training data by generating another 49,445 meta-instances by resampling the generated data and introducing some noise. More specifically, each edge in the adjacency-based graphs and the label vectors had a probability of 0.1 of being flipped. Moreover, Gaussian noise was added to the purity vectors with a standard deviation of 0.1, thus leading to considerably dissimilar meta-instances used during the GNN's training process.

4.2 Meta-classifiers and Hyperparameter Tuning

We will compare our approach against four meta-classifiers that use traditional meta-features to describe the pattern classification problems. The first meta-classifier concerns Binary Relevance (BIREL) [28], which builds a binary classification problem per label. BIREL will use a decision tree as a base classifier. The hyperparameters to be tuned include the information criterion (gini or entropy) and the strategy for selecting the maximum number of features (sqrt, log2). The second meta-classifier is RAndom k-labELsets (RAkEL) [17], which randomly partitions the labels into k subsets, each treated as a single-label classification problem. The hyperparameter tuning strategy used for this algorithm is the same as for BIREL. The third meta-classifier is Multi-label kNN (MLkNN) [29], which is an extension of the k-Nearest Neighbors (kNN) to handle multi-label data. The number of neighbors (3, 5, 7, or 10) is the hyperparameter to be tuned. The last baseline meta-classifier concerns a Convolutional Neural Network (CNN) described in Ref. [16]. This CNN architecture is fairly complex and involves fully connected, normalization, convolutional, pooling, and dropout layers. Regarding the hyperparameters, we will fine-tune the learning rate (0.001, 0.005, 0.01), the dropout rate (from 0.0 to 0.9 with step equal to 0.1), and the neuron's activation function (`relu`, `elu`, `selu`). Moreover, we use the Adam optimizer with 200 epochs and the squared hinge function as the activation function in the last layer. None of these models support the granular knowledge graphs introduced in this paper to describe the classification datasets. Hence, we will use traditional meta-features extracted using `pymfe` package [1] and an additional set of meta-features described in Ref. [16]. The first set involves 350 meta-features, while the second set is smaller and comprises 62 meta-features.

The architecture of the GNN is much simpler than the CNN mentioned above involving two graph convolutional layers, each using 32 channels and the rectified linear unit (ReLU) as the activation function. These graph convolutional layers are followed by a fully-connected layer of 1024 neural ReLU units and a dropout layer. The last layer uses the *softsign* activation function $f(x) = \frac{x}{|x|+1}$. Aiming to use this function consistently, we need to encode the binary multilabel vectors with $\{-1, 1\}$. Similarly to the CNN model, we will employ the Adam optimizer and the mean squared error as the loss function. The hyperparameters of the GNN to be optimized are the dropout rate (0.2, 0.5, 0.8) and the batch size (64, 128, and 256). Fixed parameters that are not optimized include the number of prototypes (50 for all datasets) and the distance threshold (distance values corresponding with the 25th percentile).

It is worth mentioning that all simulations were performed on a high performance computing environment that uses two Intel Xeon Gold 6152 CPUs at 2.10 GHz, each with 22 cores and 187 GB of memory.

4.3 Numerical Results and Discussion

To evaluate the predictions of the meta-classifiers, we will employ multiple evaluation metrics. For the multilabel metrics, we will compute the Hamming loss and micro versions of precision, recall, specificity, and F1-score. The process involves calculating these metrics individually for each label and then reporting their unweighted mean. Another crucial measure in our study is the "hit rate" due to its practical significance. It quantifies the proportion of instances for which the most likely positive model predicted by the meta-classifier is an optimal (positive) model. Equation (3) formalizes this performance measure,

$$hit(X) = \frac{1}{|X|} \sum_{x \in X} \begin{cases} 0, & if\ \hat{L}^{+}_{(x)} \not\subseteq L^{+}_{(x)} \\ 1, & if\ \hat{L}^{+}_{(x)} \subseteq L^{+}_{(x)} \end{cases} \tag{3}$$

where X denotes the set of instances, $\hat{L}^{+}_{(x)}$ is the most probable positive model for a given instance as predicted by the meta-classifier, whereas $L^{+}_{(x)}$ is the set of all known positive models associated to that instance.

Tables 1 and 2 show the performance metrics associated with the meta-classifiers using the meta-features described in Refs. [1,16], respectively. It is worth recalling that our GNN does not use these meta-features but the granular knowledge graphs describing the problems' decision space. The best-performing model is highlighted in black, which turns out to be the GNN model in all cases. When contrasting our model with the CNN meta-classifier in particular, we notice a sharp increase in the precision, specificity, and F1-score values. We also observe a moderate increase in the hit rate and recall values. The Hamming loss also decreases, which is the desired behavior since that performance metric should be minimized. These numerical results confirm that we can tackle the model selection problem by using the proposed granular knowledge graphs to describe the classification problems, releasing us from the burden of computing low-level meta-features.

Table 1. Performance measures for each meta-classifier using the meta-features in Ref. [16]

	CNN	BIREL	RAKEL	MLKNN	GNN
Hit rate	0.91	0.49	0.50	0.72	**0.92**
Precision	0.80	0.68	0.67	0.57	**0.86**
Recall	0.78	0.68	0.67	0.67	**0.81**
Specificity	0.87	0.79	0.79	0.67	**0.91**
F1-score	0.79	0.68	0.67	0.62	**0.84**
Hamming	0.15	0.24	0.24	0.30	**0.12**

The best-performing algorithm according to each metric is highlighted in bold

Table 2. Performance measures for each meta-classifier using the meta-features in Ref. [1]

	CNN	BIREL	RAKEL	MLKNN	GNN
Hit rate	0.90	0.47	0.49	0.80	**0.92**
Precision	0.79	0.67	0.66	0.63	**0.86**
Recall	0.78	0.67	0.67	0.71	**0.81**
Specificity	0.86	0.79	0.79	0.73	**0.91**
F1-score	0.78	0.67	0.66	0.67	**0.84**
Hamming	0.16	0.24	0.25	0.26	**0.12**

The best-performing algorithm according to each metric is highlighted in bold

Next, let us analyze the predictions for all LGBM models using different numbers of estimators, booster strategies, and learning rates. These models respectively

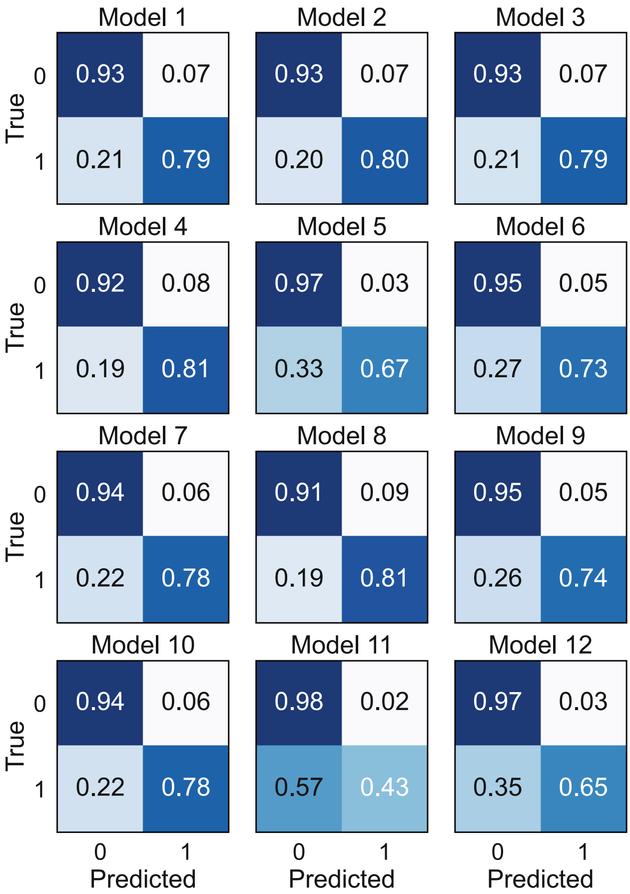


Fig. 4. Normalized confusion matrices concerning the LGBM models, each using a different number of estimators, booster strategy, and learning rate

employ the following settings: `500/gbdt/0.1`, `1000/gbdt/0.1`, `500/gbdt/0.05`, `1000/gbdt/0.05`, `500/gbdt/ 0.01`, `1000/gbdt/0.01`, `500/dart/0.1`, `1000/dart/0.1`, `500/dart/0.05`, `1000/dart/0.05`, `500/dart/0.01`, and `1000/dart/0.01`. Figure 4 shows the confusion matrix for each model, indicating that the GNN induces low false positive rates. This behavior is desired since the meta-classifier should not suggest models that lead to sub-optimal results. In contrast, the GNN might fail to retrieve positive models, as the false negative rates indicate. This behavior is not problematic since retrieving a single positive model would be enough to solve the model selection problem. The confusion matrices for the RF models exhibit similar patterns.

Another aspect to be explored is the loss function's role in the GNN's performance. To do that, we will study the Hamming loss reported by the GNN on a validation set when using the following loss functions:

– Mean Squared Error (MSE):

$$f_1(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

– Squared Hinge Loss:

$$f_2(y, \hat{y}_i) = \max(0, 1 - y_i \cdot \hat{y}_i)^2$$

– Huber Loss:

$$f_3(y_i, \hat{y}_i) = \begin{cases} 0.5 \cdot (y_i - \hat{y}_i)^2, & if\ |y_i - \hat{y}_i| \leq \delta \\ \delta \cdot |y_i - \hat{y}_i| - 0.5 \cdot \delta^2, & if\ |y_i - \hat{y}_i| > \delta \end{cases}$$

where y_i and $\hat{y}_i$ are the target (ground truth) and the predicted labels for the i-th dimension of the label vector, respectively. In the Huber loss function (also known as Smooth Mean Absolute Error), the δ parameter determines the threshold between the quadratic and linear loss.

Figure 5 shows the simulation results concerning the loss functions such that the number of epochs was set to 200 and the batch size to 256. Surprisingly, the Squared Hinge function reports the worst results, while the Huber function stands as the best-performing alternative, closely followed by the MSE function. It is worth clarifying that we do not visualize the first 25 epochs in order to highlight the performance differences between these functions towards the end.

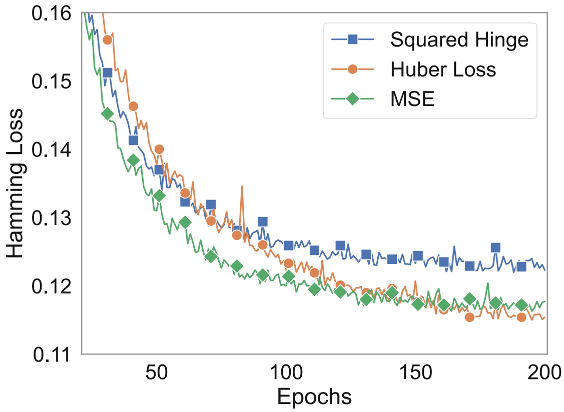


Fig. 5. Hamming loss on a validation set reported by the GNN when using different loss functions

Overall, the results indicate that the granular knowledge graphs can effectively describe the decision space of structured pattern classification problems. Moreover, the fact that we need fewer prototypes than instances indicates that this knowledge representation approach is much more efficient than the traditional approach of using meta-features. Note that fine-tuning the number of prototypes for each problem could improve prediction rates.

5 Concluding Remarks

This paper presented a GNN-based approach for tackling the model selection problem in machine learning. Contrary to other approaches reported in the literature that focus on selecting the algorithm for a given problem and optimizing its hyperparameters, our approach attempts to solve both problems in a single shot. In that way, selecting the most suitable model and its optimal hyperparameters will be a matter of processing an instance with the GNN meta-classifier. However, the main novelty of our paper concerns the granular knowledge graphs that describe each pattern classification problem. Such knowledge graphs enclose a granular representation of the problem's decision space, freeing us from computing endless meta-features. It is worth highlighting that computing such meta-features can be computationally expansive in some cases, which conflicts with the intuition behind the model selection problem.

The numerical simulations show that our proposal outperforms the previous attempts at solving the multilabel model selection problem while barely performing hyperparameter tuning. More importantly, the GNN model reports relatively low false positive rates, a key behavior when selecting the optimal models for a given dataset. The hit rate is another important measure since it provides insights into the meta-classifier's performance when predicting the most likely model for a dataset. After all, confidently selecting a model that produces optimal results might be enough in practical situations. In the case of high-stake application domains where misclassifications come at a high cost, we can use the GNN model to filter out negative models and perform a grid search using the positive ones, thus reducing the computational complexity of this procedure.

The promising results of our approach are accompanied by research paths to be explored in future research endeavors. First, the GNN meta-classifier might suffer from scalability issues since the number of output neurons depends on the number of algorithms and hyperparameters to be explored. Moreover, the set of hyperparameters and their values must be defined in advance. Both limitations can be tackled by re-designing the network such that each output neuron represents a hyperparameter instead of a hyperparameter-value combination. Second, such a powerful model requires larger training sets that better resemble real-world datasets. Finally, it would be interesting to fine-tune the number of prototypes (nodes in the granular knowledge graph) when building the GNN-based meta-classifier.

References

1. Alcobaça, E., Siqueira, F., Rivolli, A., Garcia, L.P.F., Oliva, J.T., de Carvalho, A.C.P.L.F.: MFE: towards reproducible meta-feature extraction. J. Mach. Learn. Res. **21**(111), 1–5 (2020)

2. Ali, S., Smith, K.A.: On learning algorithm selection for classification. Appl. Soft Comput. **6**(2), 119–138 (2006)
3. Brazdil, P., Gama, J., Henery, B.: Characterizing the applicability of classification algorithms using meta-level learning. In: Bergadano, F., De Raedt, L. (eds.) Proceedings of the European Conference on Machine Learning (ECML-94), pp. 83–102. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
4. Brazdil, P.B., Soares, C., Da Costa, J.P.: Ranking learning algorithms: using IBL and meta-learning on accuracy and time results. Mach. Learn. **50**(3), 251–277 (2003)
5. Iddo Drori, L., Liu, Y.N., Koorathota, S.C., Li, J.S., Freire, J., Udell, M.: Automl using metadata language embeddings, Antonio Khalil Moretti (2019)
6. Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., Hutter, F.: Auto-Sklearn 2.0: hands-free automl via meta-learning. J. Mach. Learn. Res. **23**(261), 1–61 (2022)
7. Helali, M., Mansour, E., Abdelaziz, I., Dolby, J., Srinivas, K.: A scalable automl approach based on graph neural networks. Proc. VLDB Endow. **15**(11), 2428–2436 (2022)
8. Hernandez, M., Epelde, G., Alberdi, A., Cilla, R., Rankin, D.: Synthetic data generation for tabular health records: a systematic review. Neurocomputing **493**, 28–45 (2022)
9. Hospedales, T.M., Antoniou, A., Micaelli, P., Storkey, A.J.: Meta-learning in neural networks: a survey. IEEE Trans. Pattern Analy. Mach. Intell. **44**(9), 5149–5169 (2021)
10. Hutter, F., Kotthoff, L., Vanschoren, J.: Automated Machine Learning: Methods, Systems, Challenges, 1st edn. Springer Publishing Company, Incorporated (2019)
11. Jin, H., Chollet, F., Song, Q., Xia, H.: AutoKeras: an automl library for deep learning. J. Mach. Learn. Res. **24**(6), 1–6 (2023)
12. Kalousis, A., Theoharis, T.: NOEMON: an intelligent assistant for classier selection. In: Proceedings of the ICML-99 Workshop on Recent Advances in Meta-Level Learning and Future Work, pp. 28–37 (1999)
13. Khan, I., Zhang, X., Rehman, M., Ali, R.: A literature survey and empirical study of meta-learning for classifier selection. IEEE Access **8**, 10262–10281 (2020)
14. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA: automatic model selection and hyperparameter optimization in weka. In: Automated Machine Learning: Methods, Systems, Challenges, pp. 81–95 (2019)
15. Mannino, M., Abouzied, A.: Is this real? generating synthetic data that looks real. In: Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, pp. 549–561 (2019)
16. Nápoles, G., Grau, I., Güven, Ç., Özdemir, O., Salgueiro, Y.: Which is the best model for my data? (2023)
17. Padmashani, R., Nivaashini, M., Vidhyapriya, R.: RAkEL algorithm and mahalanobis distance-based intrusion detection system against network intrusions. In: International Conference on Artificial Intelligence, Smart Grid and Smart City Applications, pp. 689–696. Springer (2019)
18. Pinto, F., Soares, C., Mendes-Moreira, J.: Towards automatic generation of metafeatures. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 215–226. Springer (2016)
19. Rice, J.R.: The algorithm selection problem. In: Advances in Computers, vol. 15, pp. 65–118. Elsevier (1976)

20. Rivolli, A., Garcia, L.P.F., Soares, C., Vanschoren, J., de Carvalho, A.C.P.L.F.: Characterizing classification datasets: a study of meta-features for meta-learning (2018)
21. Rivolli, A., Garcia, L.P.F., Soares, C., Vanschoren, J., de Carvalho, A.C.P.L.F.: Meta-features for meta-learning. Knowl.-Based Syst. **240**, 108101 (2022)
22. Song, Q., Wang, G., Wang, C.: Automatic recommendation of classification algorithms based on data set characteristics. Pattern Recogn. **45**(7), 2672–2689 (2012)
23. Tornede, T., Tornede, A., Hanselle, J., Mohr, F., Wever, M., Hüllermeier, E.: Towards green automated machine learning: Status quo and future directions. J. Artif. Intell. Res. **77**, 427–457 (2023)
24. Vanschoren, J.: Meta-learning: A Survey (2018)
25. Wang, G., Song, Q., Zhang, X., Zhang, K.: A generic multilabel learning-based classification algorithm recommendation method. ACM Trans. Knowl. Discov. Data **9**(1) (2014)
26. Xu, L., Skoularidou, M., Cuesta-Infante, A., Veeramachaneni, K.: Modeling tabular data using conditional GAN. Adv. Neural Inf. Proc. Syst. **32** (2019)
27. Yale, A., Dash, S., Dutta, R., Guyon, I., Pavao, A., Bennett, K.P.: Generation and evaluation of privacy preserving synthetic health data. Neurocomputing **416**, 244–255 (2020)
28. Zhang, M.-L., Li, Y.-K., Liu, X.-Y., Geng, X.: Binary relevance for multi-label learning: an overview. Front. Comp. Sci. **12**(2), 191–202 (2018)
29. Zhu, D., Zhu, H., Liu, X., Li, H., Wang, F., Li, H., Feng, D.: CREDO: Efficient and privacy-preserving multi-level medical pre-diagnosis based on ml-knn. Inf. Sci. **514**, 244–262 (2020)
30. Zhu, X., Yang, X., Ying, C., Wang, G.: A new classification algorithm recommendation method based on link prediction. Knowl.-Based Syst. **159**, 171–185 (2018)
31. Zimmer, L., Lindauer, M., Hutter, F.: Auto-PyTorch: Multi-fidelity metalearning for efficient and robust AutoDL. IEEE Trans. Pattern Anal. Mach. Intell. **43**(9), 3079–3090 (2021)