AutoGRD: Model Recommendation Through Graphical Dataset Representation

Conference Paper · November 2019

DOI: 10.1145/3357384.3357896

AutoGRD: Model Recommendation Through Graphical Dataset Representation

Noy Cohen-Shapira
Ben-Gurion University of the Negev
Beer-Sheva, Israel
noycohe@post.bgu.ac.il

Lior Rokach
Ben-Gurion University of the Negev
Beer-Sheva, Israel
liorrk@post.bgu.ac.il

Bracha Shapira
Ben-Gurion University of the Negev
Beer-Sheva
bshapira@post.bgu.ac.il

Gilad Katz
Ben-Gurion University of the Negev
Beer-Sheva, Israel
giladkz@post.bgu.ac.il

Roman Vainshtein
Ben-Gurion University of the Negev
Beer-Sheva, Israel
romanva@post.bgu.ac.il

ABSTRACT

The widespread use of machine learning algorithms and the high level of expertise required to utilize them have fuelled the demand for solutions that can be used by non-experts. One of the main challenges non-experts face in applying machine learning to new problems is algorithm selection – the identification of the algorithm(s) that will deliver top performance for a given dataset, task, and evaluation measure. We present AutoGRD, a novel meta-learning approach for algorithm recommendation. AutoGRD first represents datasets as graphs and then extracts their latent representation that is used to train a ranking meta-model capable of accurately recommending top-performing algorithms for previously unseen datasets. We evaluate our approach on 250 datasets and demonstrate its effectiveness both for classification and regression tasks. AutoGRD outperforms state-of-the-art meta-learning and Bayesian methods.

CCS CONCEPTS

• **Computing methodologies → Machine learning approaches**; *Information extraction*; • **Theory of computation** → Machine learning theory;

KEYWORDS

Meta-learning; Algorithm selection; AutoML; Dataset representation; Classification; Regression; Graph embedding

ACM Reference format:
Noy Cohen-Shapira, Lior Rokach, Bracha Shapira, Gilad Katz, and Roman Vainshtein. 2019. AutoGRD: Model Recommendation Through Graphical Dataset Representation. In *Proceedings of The 28th ACM International Conference on Information and Knowledge Management, Beijing, China, November 3–7, 2019 (CIKM ’19),* 10 pages.
https://doi.org/10.1145/3357384.3357896

1 INTRODUCTION

The explosive growth of digital information has led to the widespread adoption of machine learning (ML) solutions. While ML can assist in decision-making and data analysis, human expertise is often still required. Human involvement is needed for two reasons. First, domain experts may provide unique characteristics of the domain which may drastically affect the performance of the algorithms; and second, the large number of algorithms and hyperparameter configurations make brute force search infeasible. Therefore, expert data-scientists are needed.

One of the main challenges in applying ML to a new problem is *algorithm selection* – the identification of algorithms (or algorithm families) that are likely to perform well on a given combination of dataset, task, and evaluation measure [13]. Algorithm selection is difficult, since the performance of an algorithm is primarily a "black box" affected by multiple characteristics of the dataset, including size, the number of features and their composition, the number of classes, instance distribution etc. The complexity of algorithm selection, as well as the time-consuming nature of testing multiple configurations, underscore the need to automate this process.

The term automated machine learning (AutoML) is often used to describe systems that aim to automate part or all of the ML "pipeline". While several AutoML techniques addressed in prior studies were aimed at automating the process of algorithm selection (Auto-Weka [31] and Auto-sklearn [13]), they require a great deal of computational resources due to the need to run candidate algorithms on the data. For large datasets, even a few runs may take several hours.

An alternative approach for addressing the algorithm selection challenge is *meta-learning*, as proposed by several studies [1, 24, 26]. We argue that the existing approaches suffer from an inherent limitation due to the fact that they only model the general characteristics of the dataset (e.g., number of instances/features, imbalance, etc.) and not the dependencies and interactions of the instances that make up the dataset. By modeling these interactions in a graph and creating embeddings that capture their essence, we show that we can better infer datasets’ similarity and hence improve the selection of learning algorithms. We present AutoGRD, a novel meta-learning-based method for algorithm recommendation. AutoGRD applies a novel approach to model the co-occurrences of the dataset’s instances, and then converts these co-occurrences into a graph. Next, the GCD algorithm [36] is applied in order to extract a latent representation of the graph. By combining the latent representation with a vector denoting the various classifiers, AutoGRD produces a meta-model capable of recommending top-performing algorithms for previously unseen datasets.

We evaluate AutoGRD on 250 datasets, testing our approach on both classification and regression tasks. Our experiments show that AutoGRD outperforms both other meta-learning based approaches and the popular Auto-Weka framework. The latter result is particularly noteworthy, since Auto-Weka iteratively tests multiple algorithms on the dataset at hand to select the best algorithm while our approach does not perform any direct test. The code of the AutoGRD algorithm will be publicly available[1].

Our contributions in this study are as follows:

1) We propose a method for creating novel fixed-size graph-based embedding representations for datasets. These representations offer new perspectives on the analyzed data .

2) We present an efficient and highly accurate meta-learning approach for algorithm selection for any data type.

3) We empirically demonstrate the merits of our approach on a large group of datasets and algorithms. Our results show that AutoGRD is highly effective for both classification and regression tasks.

2 RELATED WORK

2.1 Automatic algorithm selection

Algorithm selection is one of the main challenges in applying ML to a new dataset, with the need to take the dataset characteristics, type of task, and evaluation measure into account. The algorithm selection problem can be formally defined as follows: given a set of learning algorithms L, a dataset D, a task T (e.g., classification or regression), and an evaluation measure M, the goal of model selection is to find an algorithm $l^* \in L$ that minimizes or maximizes M [22, 31].

While it is usually necessary to rely on human expertise (i.e., data scientists) in order to find a solution to the algorithm selection problem, there has been increased interest in recent years in AutoML solutions. These solutions perform an iterative selection process to find suitable ML algorithms for a given dataset. Existing AutoML solutions for algorithm selection are typically based on Bayesian optimization [14], evolutionary algorithms [23] deep reinforcement learning [10], and budget-based evaluation [21].

Auto-Weka [31] was one of the first frameworks developed for automating ML. Based on the Bayesian optimization framework SMAC [18], Auto-Weka simultaneously tunes hyperparameters for all learning algorithm models implemented in the Weka [17] open-source software. Auto-sklearn [13] is a similar automated solution which is based on scikit-learn library in Python. Auto-sklearn stacks multiple models to achieve high predictive performance. TPOT [23] is a Python-based framework that uses an evolutionary algorithm to generate ML pipelines while optimizing their hyperparameters. AlphaD3M [10] is a newer method which uses a reinforcement learning approach and deep learning mechanisms for creating a model that can predict pipeline performance.

While all of the above mentioned approaches are effective, they are also computationally expensive (both in terms of running time and resources), because they require an iterative search of different model configurations. In addition, when given a new dataset, most of the above solutions have to start the search for a suitable ML algorithm "from scratch". This limitation is particularly problematic when tackling large datasets which require long running times.

Meta-learning ("learning about learning") is an alternative approach for dealing with algorithm selection. Meta-learning algorithms aim to learn the behavior of learning algorithms and which features of the dataset contribute to the improved performance of one algorithm over others [7]. This knowledge can then be used to better identify high-performing algorithms for solving a task on previously unseen datasets [28].

As described by [26], a meta-learning framework begins with a collection of learning algorithms and datasets. The framework extracts meta-features that capture (or aspire to capture) the "essence" of a given dataset. Each learning algorithm is then applied on each dataset and estimates the performance. The framework collects the meta-features and the performance of the evaluated learning algorithms as meta-data. Then, a learning algorithm, a meta-learner, is trained on the meta-data and produces a meta-model that matches the values of the meta-features with the most suitable algorithm for each dataset. Finally, for each new dataset, the meta-learning approach extracts the meta-features and uses the meta-model to recommend algorithm(s) for that dataset.

The meta-features used in existing work are commonly divided into three categories [2, 34]:

1) **Statistical and information-theoretic meta-features** directly describe the training set. These meta-features can be simple (e.g., number of training instances), statistical (e.g., mean, variance), or information theory-based. An example of a comprehensive list of meta-features can be found in [19].

2) **Model-based meta-features** describe the learning model to be applied on a given dataset, e.g., the number of leaf nodes of a decision tree [24].

3) **Landmarker meta-features** are generated by using the estimated performance of a simple learning algorithm on a given dataset for a quick performance assessment [1].

While the design of meta-features is an important process, it is also considered a major challenge for meta-learning [2, 3, 33]; given this difficulty, meta-feature extraction has recently been the subject of research aimed at its systematization. The systematic framework for meta-feature extraction proposed in [26] is based on three components: meta-function (e.g., entropy), object (e.g., set of independent variables), and post-processing function (e.g., average value). By applying the meta-function to all possible objects and processing the results with all possible post-processing functions, the method enables the systematic generation of sets of meta-features.

Existing meta-learning-based approaches for algorithm selection largely focus on modeling the relationships between certain data characteristics and the performance of ML algorithms [7, 20]. Recently, a study by Vainshtein et al. [32] proposed AutoDi, a framework for combining two types of meta-features: dataset-based and embedding-based. With the latter, features were extracted from a large corpus of academic publications; these features were used to

[1] in https://github.com/noycohen100/AutoGRD.git

train a ranking model using XGBoost. Given a new dataset, AutoDi produces a list of algorithms, sorted by their predicted efficacy on the data. The dataset-based features of AutoDi are based on the work by Katz et al. [19], who used this type of features for the purpose of automatic feature engineering. In contrast to the above-mentioned studies, our approach utilizes a different type of meta-features which is extracted based on instance co-occurrence.

2.2 Graph representation by embedding

Graphs have been used for modeling entity interactions in various fields,including social networks and biological networks [15]. The ever-increasing amount of data has fuelled the search for more efficient forms of representation.

Graph embedding is currently the state-of-the-art graph representation approach. Its aim is to project a graph into a low-dimensional space (of size *d*), while preserving some of the connections between the vertices in the original graph [5]. Goyal et al. [15] define graph embedding as: "given a graph $G = (V, E)$, graph embedding is a mapping $f : v_i \rightarrow y_i \in R^d, \forall i \in [n]$ such that $n = |V|$, $d << |V|$, and the goal of f is to preserve some proximity measure defined on graph G".

While earlier graph embedding approaches employed factorization methods [27], these methods often failed to capture the global structure of the graph. This shortcoming led to the development of current embedding-based methods, which are both more adept at representing global structures while also being more scalable. One such solution is LINE [30], a proximity-based method which utilizes both first and second-order methods for generating the embedding.

Another approach for graph embedding is based on random walks. Two frameworks which utilize this approach are DeepWalk [25] and node2vec [16], although the latter employs bias-based random walks. Both approaches strive to preserve the high-order proximity among nodes by maximizing the probability of a subsequent node occurring in fixed length random walks [15]. In general, node2vec is considered a better performer than DeepWalk due to its ability to iterate between breadth-first and depth-first searches.

While all of the above-mentioned methods produce a vector representation for each vertex in a d-dimensional space, they do not provide a fixed size matrix of the graph representation which is required for many ML tasks including the one presented in this study. One approach used to address this shortcoming was presented in [36], where the authors propose the *graphlet correlation distance* (GCD) method.

Graphlets are small, induced subgraphs that appear at any frequency, whice are generated by selecting a small group of nodes in a graph along with all of their inter-connecting edges (no outer edges are selected). This approach utilizes automorphism orbits, which are used to capture topologically symmetrical nodes in a graphlet. For further reading we refer the reader to [35]. By using graphlets, the authors produce a representation for each vertex that contains the exposed and latent properties of the network topology. Based on the vertices' representation, GCD enables the creation of an *mXm* embedding representation matrix of the graph referred to as the *graphlet correlation matrix* (GCM), where *m* is a configurable parameter.

3 MOTIVATION

The ever-increasing size of datasets, as well as the growing number of learning algorithms, makes the identification of top-performing algorithms for a given dataset an extremely challenging task. Since the computational cost of training multiple algorithms is often prohibitive, there is a clear need for an efficient approach capable of producing a short (and preferably ranked) list of promising candidates. This list could then be used "as is" or serve as the basis of more focused exploration..

Our work in this study is based on the hypothesis that the performance of a given learning algorithm l_i on dataset d_j – given by f(d_j,l_i) – is dependent, to a large extent, on the latent connections among the instances of the dataset. We further hypothesize that by modeling these latent connections (i.e., embeddings) we will be able to infer dataset similarity with respect to algorithm performance. In other words, we hypothesize that given a function g which generates dataset embeddings, g(d_1) ≈g(d_2)), will result in f(g(d_1),l_i) ≈ $f(g((d_2), l_i))$.

In this study we use the classification models generated by multiple decision trees (i.e., decision forests) in order to generate the dataset embeddings. While decision forests may not be an intuitive choice, a recent study by Z. Zhou and J. Feng [37] has shown that the probability distributions produced by decision forests, when stacked, can successfully model latent factors and achieve performance competitive with that of deep neural networks in various tasks. Another study by the same researchers [11] shows that this technique could be similarly applied to the creation of autoencoders.

Our approach builds on the above-mentioned studies but with some significant differences. While we also use decision forests to analyze datasets, we don't use the classification distribution or stacking. Instead, we analyze the co-occurrences of the dataset's samples in the leaves of the various decision trees and use them to construct a graph representation. We then use the GCD algorithm (see Subsection 2.2) to create a fixed-size representation of our graph. This fixed size representation can then be easily trained to produce a meta-model capable of recommending high-performing algorithms for a given dataset.

4 THE PROPOSED METHOD

4.1 Overview

AutoGRD is a meta-learning framework for ranking the performance of learning algorithms. The framework consists of two phases: *train* and *test*. During the train phase, we analyze multiple datasets and obtain the meta-data needed to train our meta-model. This phase is comprised of three main steps: *representation*, *extraction* and *modeling*. During the test phase we utilize the generated meta-model to predict algorithm performance. The test phase consists of three main steps: *representation*, *extraction* and *prediction*. The representation and extraction steps are identical for both phases. Next we review each phase in detail.

4.2 The train phase

During the train phase we evaluate multiple classification or regession algorithms on a large group of diverse datasets in order to obtain a large corpus that is then used to train our meta-model. The complete training process is described in Figure 1; this is followed by a description of each step.

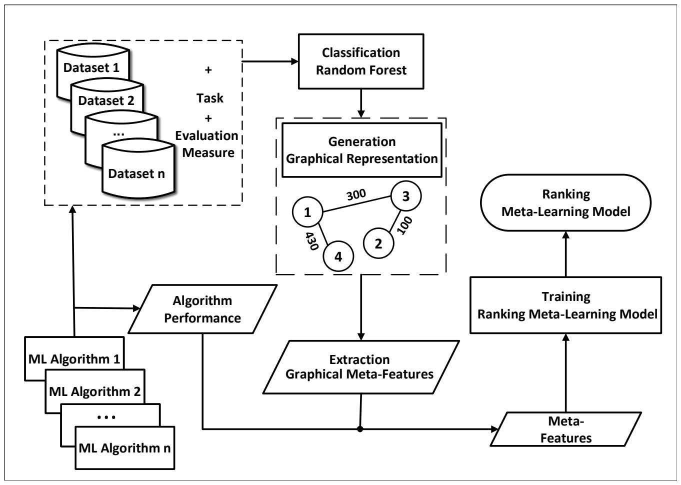


Figure 1: AutoGRD training flow

4.2.1 The representation step: generating graphical representations of datasets. The intuition behind our representation is that datasets which have a large degree of similarity are likely to induce similar performance from learning algorithms so that

by analyzing dataset similarity we will be able to infer algorithm performance for previously unseen datasets. Furthermore, we suggest that dataset similarity can be expressed through instance distribution and correlation. Therefore, by creating hierarchical representations of the instances of multiple datasets', we are able to extract latent graphical features that can quantify datasets similarity. Finally, we hypothesize that these latent graphical features which model the interactions among the dataset's instances, are more informative than dataset-level meta-features, such as number of instances, number of features, value distributions, etc., and will therefore result in more accurate recommendation of algorithms

In order to obtain the dataset instance distribution representation for a given dataset D, we apply a two-step process of *classification* and *generation*:

Classification. We apply the Random Forest algorithm, a popular ensemble method, to create a hierarchical representation of the datasets. The algorithm produces multiple decision trees, which are trained on random subsets of features and instances of the original dataset. The trees' output is then combined to produce the final output (for a comprehensive review of the algorithm we refer the reader to [29]). Next, we use the leaves of all generated decision trees for representing the training instances. Our eventual goal is to represent the dataset as a graph where the vertices represent the dataset's instances and the edges indicate the existence of a sufficiently high co-occurrence score among them. More specifically, for each possible pair of instances we count the number of leaves in which they co-appear. We denote this value as the *co-occurrence score.*

Random Forest was chosen for calculating the relationship between instances for two reasons. First, this algorithm is able to achieve high predictive performance for classification tasks [12] with relative low computational cost and without the need to tune the hyper-parameters. Second, we are interested in calculating the relationship between a pair of instances with regard to the target attribute. Standard similarity measures, such as Euclidean distance or cosine similarity, are based on all of the features in the dataset, regardless of the target variable; thus, irrelevant features can affect the similarity values. In contrast, Random Forest consists of decision trees that select the most relevant features, and two instances are considered similar according to a certain tree if they share the same path from the root to the leaf.

The Random Forest model can be geometrically expressed as a partition of the covariates space into many disjoint regions. Two instances obtain the highest co-occurrence score when they reside in the same region even if they are not necessarily identical. Instances that are located in adjacent regions share all but one leaf and therefore still have a high co-occurrence score, although not the highest score. Figure 2 illustrates this idea.

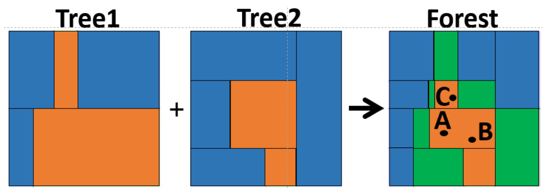


Figure 2: Geometric illustration of Random Forest

A and B fall in the same region in the forest, and therefore their distance is equal to zero, i.e., high similarity; since A and C fall in adjacent regions in the forest, their distance is equal to one. This scoring is unlike other similarity measures (e.g., Euclidean distance or cosine similarity), which would assign a value larger than zero and smaller than one for these two cases, respectively.

To remove weak and coincidental co-occurrences, we remove the weakest connections from our co-occurrence matrix. We set the percentage of removed connections to 10%, based on the recommendations found in [35]. Following this pruning, we refer to the remaining co-occurrence scores as the *filtered co-occurrence score set.* Like [35], we found that this process improves the quality of the embeddings produced by the GCD algorithm and consequently the performance of our algorithm.

Generation. We generate an unweighted and undirected graph $G = (V, E)$. Vertices represent instances and edge represent the existence of a filtered co-occurrence score between two instances. It should be noted that since the GCD algorithm does not apply to weighted graphs, the actual co-occurrence value is only used for the initial filtering of the edges. The representation process is presented in Figure 3 and Algorithm 1.

4.2.2 The extraction step: meta-feature generation. In this step we use the GCD method to generate the embedding representation of the graph described above. One advantage of GCD is that the size of its output can be predefined. This trait enables us to create fixed-size representations for graphs of varying sizes, thus simplifying this step.

Algorithm 1 Generating graphical representations of dataset

```
1:  procedure GenerateGraph(dataset D)
2:      $decisionTrees \leftarrow$ RandomForest(D)
3:      $leaves \leftarrow decisionTrees$.GetLeaves()
4:      $leaves \leftarrow$ RemoveLowCoOccurrences(leaves)
5:      $E \leftarrow \emptyset$
6:      $V \leftarrow D$.GetInstances()
7:      for each $(A, B)$ in V do
8:          $CoScore \leftarrow leaves$.GetCoOcurrence(A, B)
9:          $E \leftarrow E \cup$(A, B, CoScore)
10:     return $(V, E)$
```

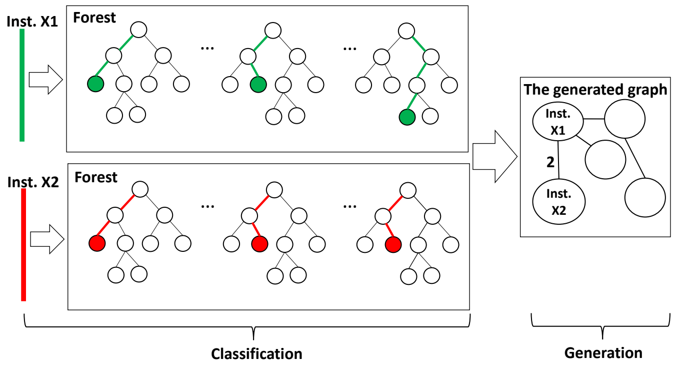


Figure 3: Representation step. Inst. X1 and Inst. X2 fall in the same leaf in two trees, and therefore the co-occurrence score of them is two.

4.2.3 The modeling step: ranking meta-learning model. To train a ranking classifier capable of utilizing the meta-features described above, we produce a large labeled training set using the following process:

1) Given a set of datasets D and a set of learning algorithms L, we evaluate all combinations of $d \in D$ and $l \in L$. We denote the result of this evaluation as $R_{l,d}$.

2) For each combination of $d \in D$ and $l \in L$, we create a set of meta-features that consists of the following: *a)* M_d – the set of meta-features generated in the extraction step (described in Subsection 4.2.2); *b)* M_l – a single discrete feature describing l; and *c)* $R_{l,d}$ – the results of the evaluation of l on d.

3) We train the XGBoost algorithm [6] on the joint set $\{M_d \cup M_l, R_{l,d}\}$ where we aim to predict the true ranking of algorithm l based on its performance $R_{l,d}$. We chose XGBoost as our ranking algorithm, since previous work [4] has shown that it well suited to this.

The result of this step is a trained meta-ranking model, capable of ranking every $l \in L$ for previously unseen datasets. The modeling step is presented in Algorithm 2.

4.3 The test phase

In this phase, we attempt to produce a list of learning algorithms, ranked by their predicted performance on a previously unseen dataset d_{new}. This process is described in Figure 4 and Algorithm 3. The steps of this phase are as follows:

Algorithm 2 Ranking meta-learning model generation

```
1:  procedure GenerateModel(datasets $D$, algorithms $L$)
2:      $MetaFeatures \leftarrow \emptyset$
3:      for each $d$ in $D$ do
4:          $M_d \leftarrow$MetaFeatureExtraction(d)   ▷ See Section 4.2.2
5:          for each $l$ in $L$ do
6:              $R_{l,d} \leftarrow$ EvaluatePerformance($d$,$l$)
7:              $M_l \leftarrow$ DiscreteFeatureExtraction($l$)
8:              $features \leftarrow (M_d \cup M_l, R_{l,d})$
9:              $MetaFeatures \leftarrow (features \cup MetaFeatures)$
10:     $RankingModel \leftarrow XGBoost$(MetaFeatures)
11:     return $RankingModel$
```

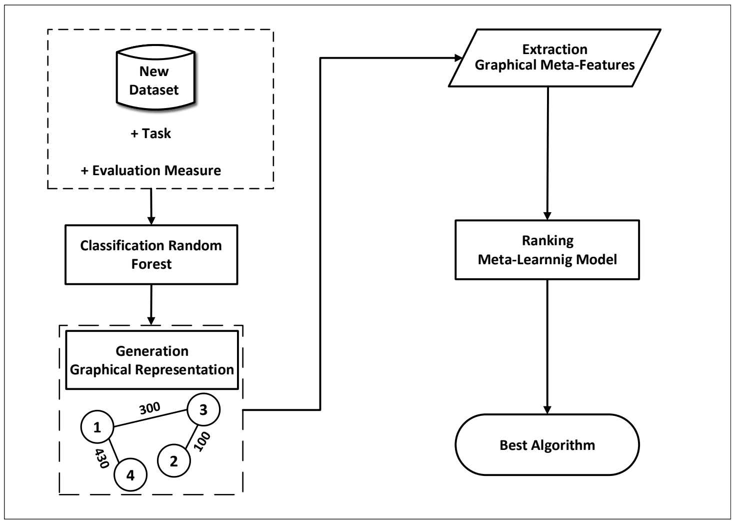

New Dataset
+ Task
+ Evaluation Measure
Classification Random Forest
Generation Graphical Representation
300
430
100
1
2
3
4
Extraction Graphical Meta-Features
Ranking Meta-Learnnig Model
Best Algorithm

Figure 4: AutoGRD testing flow

1) We generate $M_{d_{new}}$ the set of meta-features described in Section 4.2.2 for d_{new}.

2) For each $l \in L$, we generate M_l and concatenate it to a copy of $M_{d_{new}}$.

3) Once $M_{d_{new}} \cup M_l$ has been generated for every $l \in L$, we provide all meta-feature vectors to the trained XGBoost model and use it to produce $R_{l,d_{new}}$ – a ranked list of all algorithms, ordered by their predicted performance.

5 EVALUATION

In our evaluation, we examine our method as a meta-learning approach, in terms of its accuracy in the task of recommending the appropriate algorithm for a problem, and compare the advantages of our method in this respect to those of other methods.

We evaluate AutoGRD for two types of tasks: *classification* and *regression*. In our evaluation we used 150 and 100 datasets for the classification and regression tasks, respectively. All datasets are

Algorithm 3 Testing phase

```
1: procedure TestingPhase(dataset $d_{new}$, algorithms $L$)
2:     $M_{d_{new}} \leftarrow$ MetaFeatureExtraction($d_{new}$)    ▷ See Section 4.2.2
3:     $MetaFeatures \leftarrow \emptyset$
4:     for each $l$ in $L$ do
5:         $\mathrm{M}_l \leftarrow$ DiscreteFeatureExtraction($l$)
6:         $features \leftarrow (M_{d_{new}} \cup \mathrm{M}_l)$
7:         $MetaFeatures \leftarrow MetaFeatures \cup features$
8:     $\mathrm{R}_{l,d_{new}} \leftarrow RankingModel$.Rank(MetaFeatures)    ▷ See Algorithm 2
9:     return $R_{l,d_{new}}$
```

available in the following online repositories: UCI,[2], OpenML,[3] and Kaggle.[4]

5.1 Experimental setup

We used the following setup in all of our experiments:

1) For the purposes of training and evaluation, we used a leave-one-out approach: for each evaluated dataset d_i, we trained the ranking classifier using meta-features from $d_j \in D$ where $i \neq j$ and D is a collection of datasets.

2) Since the performance of Auto-Weka increases with running time, we set its 'timeLimit' parameter to 15 minutes in order to provide a fair comparison of the methods.

3) Since many of the algorithms evaluated in this study are in fact different implementations of the same algorithm (Random Forest, for example, has eight variations), we group the algorithms into "families". There are 17 and 10 families in the classification and regression experiments, respectively. [5] AutoGRD and the meta-learning baselines, therefore produce a ranked list of families rather than algorithms. For each family, we automatically select the algorithm with the highest performance on the dataset. This is the same experimental setting used in [32].

4) We trained 500 trees with a maximal depth of eight and the default configuration(as defined in scikit learn user guide) for the minimum number of samples in a leaf node, for the Random Forest algorithm in the representation step (see Subsection 4.2.1 for details)

5) We used the XGBoost algorithm to train the algorithm ranking model. We chose the objective function: 'rank:pairwise' and set the algorithm's parameters empirically using the leave-one-out approach. Our model contains shallow trees with 220 and 150 estimators for classification and regression, respectively. Shallow trees are appropriate, because we have few instances and bushy trees tend to overfit in this case.

6) For reasons of efficiency, we used a distributed MapReduce algorithm [8] to generate the graph for each dataset.

7) The authors of [35] recommend setting the value $m = 15$ when dealing with a large collection of datasets. We trained the GCD algorithm based on this recommendation, resulting in a GCM matrix of $15 * 15$ (see Subsection 4.2.2 for details). This configuration resulted in the generation of $15^2 = 225$ meta-features, a number which could be reduced in half to 105, because the matrix is symmetrical.

[2]https://archive.ics.uci.edu/ml
[3]https://www.openml.org
[4]www.kaggle.com
[5]The list of families and their associated algorithms will be available in the code repository

5.2 Measures

1) Since our goal is to recommend the top-performing algorithm for a given dataset, task and evaluation measure combination, we used the *relative maximum value* (RMV) measure. For each learning algorithm, RMV calculates the ratio between the evaluation measure value that an algorithm achieved to the best value achieved by any learning algorithm in the same setting. For example, if the top performing algorithm on the *German credit* dataset achieved an accuracy of 99.5% and another algorithm achieved an accuracy of 95%, their RMV values would be one and 0.954, respectively.

2) We used the *accuracy measure* for the evaluation of the classification task and - Mean Squared Error (MSE) measure for the regression task. As explained above, these two values were used to calculate the RMV.

5.3 Baselines

We compare AutoGRD to four baselines: **(a)** A brute-force evaluation of all possible algorithms. This baseline yields the best possible results, but it is expensive both in time and computing resources. We calculate the RMV measure using this baseline. **(b)** The dataset-based version of AutoDi, proposed in the study by [32]. Like AutoGRD, AutoDi is a meta-learning approach that does not require exhaustive evaluation of all algorithms. **(c)** The Random Forest algorithm, which is the algorithm with the *highest overall average performance* across all datasets, both for the regression and classification tasks. **(d)** Auto-Weka, a popular Bayesian approach for automatic algorithm selection.

5.4 Evaluation results: classification

We conducted our experiments on 150 datasets[6], all available from online repositories. In order to conduct our evaluation, we had to obtain the performance of every possible dataset/algorithm combination. We obtained this information from two sources:

1) For 102 datasets, the performance for all algorithms was obtained from the extensive study conducted by [12].

2) For the remaining 48 datasets, we obtained various algorithms' accuracy measures from the OpenML repository. Since the OpenML platform enables users to submit algorithms and have their performance evaluated, we downloaded and extracted the information relevant to the algorithms we evaluate. All of the datasets had at least 60 unique submissions.

Overall, we evaluated seventeen families of classification algorithms, consisting of a total of 179 algorithms.

The results of our evaluation are shown in Table 1 and Figure 5, where it is clear that AutoGRD has both the highest average RMV and the smallest standard deviation of all of the methods evaluated.

We used the Friedman test to validate the statistical significance of differences between the evaluated methods [9]. The null hypothesis that the four methods perform the same and the observed

[6]https://bit.ly/2LNKPex

differences are merely random was rejected with p < 0.01. We proceeded with Wilcoxon signed-rank post-hoc tests and conclude that the differences between AutoGRD and all other methods were found to be statistically significant with p<0.01.

Table 1: The average RVM of the evaluated methods over 150 classification datasets. '*' denotes that the difference from other values in the same line is statistically significant (p<0.01).

Measure	Random Forest	AutoDi: dataset-based	Auto-Weka	AutoGRD
Average RMV	96.3	95.6	96.1	99.4 (*)(*)(*)
Stdev (%)	6	7	6	1

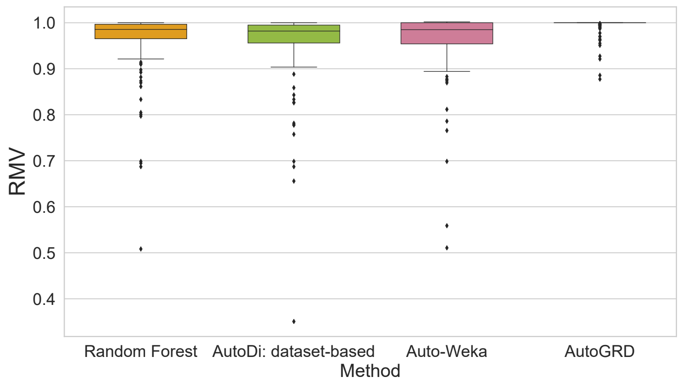


Figure 5: Distribution of the RMV scores for AutoGRD and the various baselines (the solid boxes represent the IQR based on performance).

Table 1 presents a comparison of our approach to the baselines. Similarly to our approach, the Random Forest and AutoDi:dataset-based baselines do not require an extensive evaluation of all algorithms in order to produce a recommendation. It is clear that AutoGRD outperforms both baselines by a wide margin, with a minimum RMV of 0.87. In addition, it is clear that AutoGRD consistently outperforms the Auto-Weka baseline both in performance and consistency (i.e., small standard deviation). This conclusion is also supported by Figure 5, which presents the distribution, median, and range of the results.

The superior performance of our approach compared to Auto-Weka is even more significant for the following two reasons: *1)* The average running time (during the test phase) of Auto-Weka per dataset was 13 minutes, compared with one minute for AutoGRD; *2)* Auto-Weka *continuously interacts* with the evaluated datasets by testing different algorithms and adapting its exploration strategy over time based on the results. AutoGRD, on the other hand, is a "single-shot" method that only interacts with the dataset once in order to obtain the meta-features. Despite this evident disadvantage, AutoGRD is significantly better than Auto-Weka.

5.5 Discussion: classification results

Additional analysis of the results presented in the previous section yielded a few other interesting insights regarding the differences between AutoGRD and the baselines.

AutoGRD is consistently better at identifying the optimal algorithm. In addition to obtaining the top RMV score (99.4%), our approach achieved a perfect RMV on 118 of the 150 evaluated datasets. As shown in Table 2, this value is significantly higher than any of the baselines.

Table 2: The number of classification datasets for each method for which the optimal algorithm is recommended.

Method Name	Number of Datasets with Top Performance
Random Forest	27 (18%)
AutoDi: dataset-based	27 (18%)
Auto-Weka	47 (31.3%)
AutoGRD	**118 (78.6%)**

AutoGRD rarely recommends an ineffective algorithm. Figure 6 presents the RMV score distribution of AutoGRD and the baselines (each approach only proposes its top selection). It is clear to see that AutoGRD is much more effective in its predictions, with RMV of ~0.9 as its lowest score.

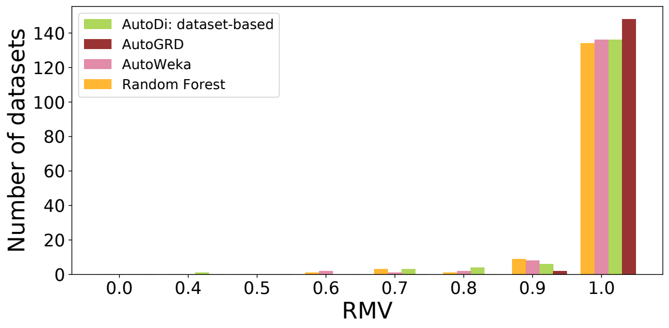


Figure 6: RMV score distribution histogram of the evaluated method across 150 classification datasets

Since both AutoGRD and AutoDi: dataset-based are meta-learning, "single-shot" methods, we were interested in better understanding the differences between them. Our analysis points to the following conclusions:

AutoGRD is more generic than the AutoDi: dataset-based. Table 3 presents the three datasets for which AutoGRD and AutoDi: dataset-based had the largest differences in performance (RMV score). In each case, AutoGRD outperforms AutoDi by a significant margin. These results lead us to conclude that our method is better suited in cases in which the dataset in question is infrequently used in the dataset collection or has characteristics that may make commonly-used algorithms ineffective (i.e., a small number of features or instances). In cases such as these, our approach's ability to represent the dataset as a graph can provide more useful insights regarding the dataset, thus leading to better algorithm selection.

Table 3: The top-3 datasets with the largest RMV difference between AutoGRD and AutoDi: dataset-based method in classification task. Positive values represents better performance by AutoGRD

Dataset Name	Performance Difference	Suggested Reason
image-segmentation	0.64	Infrequent dataset type in our datasets collection
teachingAssistant	0.31	Small number of instances
visualizing-livestock	0.295	Small number of instances and features

AutoGRD is consistently better than AutoDi: dataset-based. Untilthis point we have evaluated the performance of AutoGRD and the baselines based on their top recommendation (i.e., a single algorithm). Now, in order to determine whether the *overall recommendation quality* of AutoGRD is higher than AutoDi: dataset-based, we evaluate these algorithms based on their top-X-ranked algorithms. In Figure 7 we present the RMV results when choosing the best-performing algorithm out of the top-X recommendations. While both methods show increased performance, AutoGRD achieves a perfect RMV after evaluating only three algorithms. Furthermore, even using the best of its top five recommendations does not bring AutoDi: dataset-based to the same level of performance as AutoGRD with a single pick. Next, we examine the average number of algorithms each algorithm needs to sample in order to achieve perfect performance. For this purpose, we calculated the Mean Reciprocal Rank (MRR), which is a statistic measure used for evaluating the rank of the first correct answer:

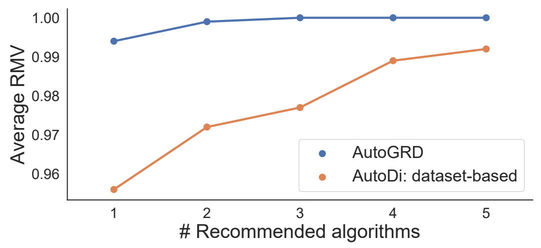


Figure 7: RMV averaged over 150 classification datasets obtained by AutoGRD and AutoDi: dataset-based method vs the number of recommended algorithms

$$MRR = \frac{1}{D} \sum_{i \in D} \frac{1}{rank_i} \quad (1)$$

Where D represents the number of datasets evaluated and $rank_i$ is the ranking of the algorithm with the highest performance achieved for the particular dataset i. Table 4 presents the results of our analysis. AutoGRD's MRR is significantly higher than that of AutoDi: dataset-based (p<0.01), with much less algorithms needed, on average, to obtain a perfect RMV.

Comparing AutoGRD to a hybrid AutoDi. The top-performing version of AutoDi reported in [32] was a hybrid of the dataset-based meta-features and information extracted from hundreds of thousands of academic papers. While we do not consider this version, which we refer to as AutoDi:hybrid, to be a "fair" competitor due to its use of external data sources, we present a comparison of its performance to AutoGRD.

Table 4: AutoGRD and AutoDi:dataset-based methods' MRR and the average rank position of the algorithm with the maximum accuracy on 150 classification datasets.

Method Name	MRR	Average Rank Position of Maximum Accuracy Algorithm
AutoGRD	0.874	1.24
AutoDi: dataset-based	0.358	5.82

Since AutoDi:hybrid was tested on the 102 datasets reported in Fernández-Delgado et al. [12], we limit our evaluation to this set. The results of the comparison are presented in Table 5, which shows that AutoGRD outperforms AutoDi:hybrid by a statistically significant margin (p<0.01 based on the Wilcoxon signed-rank test). Not only did AutoGRD outperform AutoDi:hybrid with an RMV of 99.4%, it also obtained a perfect RMV on 78 of the 102 analyzed datasets, compared with only 47 for AutoDi:hybrid.

Table 5: Average RMV for AutoDi:hybrid and AutoGRD in 102 classification datasets from the study by [12]. '*' denotes that the difference from other values in the same line is statistically significant (p<0.01).

Measure	AutoDi:hybrid	AutoGRD
Average RMV	98.6	99.4(*)
Stdev(%)	2	1
Number of datasets with top performance	47 (46%)	76 (74.5%)

5.6 Evaluation results: Regression

We conducted our experiments on 100 datasets,[7] obtained both from both the OpenML repository and Kaggle.[8] Since results were not available for all datasets, we used the Weka ML framework [17] to obtain results for all dataset/algorithm combinations. Overall, we evaluated ten families of regression algorithms, consisting of a total of 37 algorithms.

The results of our evaluation, presented in Table 6 and Figure 8, show again that AutoGRD outperforms the baselines. The score distributions presented in Figure 8 support this conclusion.

We used the ANOVA test to validate the statistical significance of differences between the evaluated methods. The null-hypothesis that the four methods perform the same and the observed differences are merely random was rejected with p < 0.01. We proceeded with Wilcoxon signed-rank post-hoc tests and conclude that AutoGRD outperforms the dataset-based version of AutoDi and AutoWeka with p<0.01 and the Random Forest algorithm with p<0.05.

[7]https://bit.ly/2Qblppv
[8]www.kaggle.com

Table 6: Average RVM of the evaluated methods over 100 regression datasets. '*' and '-' denote that the difference from other values in the same line is statistically significant (p<0.01, p=0.05 respectively).

Measure	Random Forest	AutoDi: dataset-based	Auto-Weka	AutoGRD
Average RMV	94.88	90.23	84.7	96.66 (-)(*)(*)
Stdev (%)	14.75	21.77	28.34	10.15

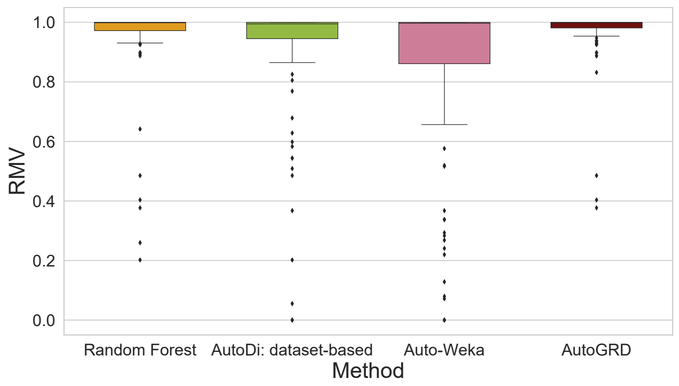


Figure 8: The distribution of the RMV scores for AutoGRD and the various baselines. The solid boxes represent the IQR based on performance.

5.7 Discussion: regression results

We compare the performance of AutoGRD to that of the meta-learning baseline, AutoDi: dataset-based.

AutoGRD rarely recommends an ineffective algorithm. Similarly to our analysis of the classification task, we again find that AutoGRD is more consistent in recommending high-performing algorithms (see Figure 9). Moreover, while AutoDi:dataset-based occasionally recommends algorithms that produce an RMV value of zero, the lowest value achieved by AutoGRD is 0.4.

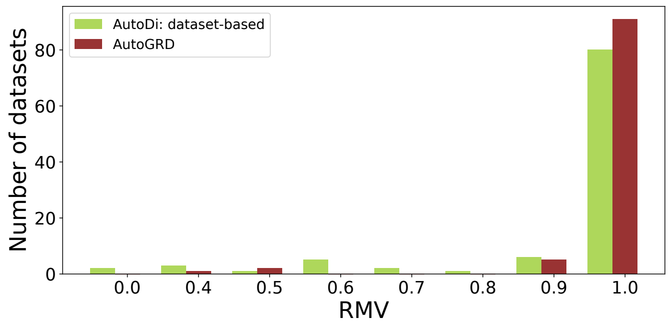


Figure 9: RMV distribution histogram of AutoGRD and AutoDi:dataset-based across 100 regression datasets

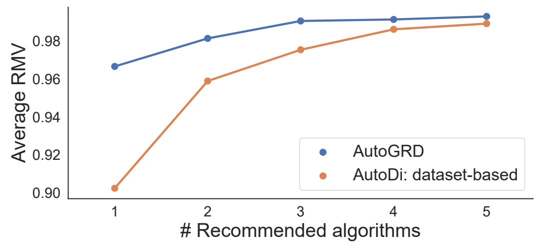
AutoGRD is a more generic and robust solution. We analyzed three cases in which the difference in performance between AutoGRD and AutoDi:dataset-based was the largest in an attempt to identify the cause of the difference. The results of our analysis are presented in Table 7. It is clear that our approach is capable of adapting to scenarios where data is scarce or has high dimensionality. We argue that the results validate our hypothesis that modeling the interactions among the instances of the dataset is more indicative than only analyzing the general statistical traits of the dataset.

Table 7: Three datasets from the top-5 datasets with the largest RMV difference between AutoGRD and AutoDi:dataset-based in regression tasks. Positive values represent better performance by AutoGRD and vice versa

Dataset Name	Performance Difference	Suggested Reason
baseball-team	1	Small number of instances
analcatdata-vehicle	0.98	Small number of instances
tecator	0.78	Large number of features

AutoGRD is consistently better than AutoDi:dataset-based. We examine overall recommendation quality for both methods by analyzing the performance of the best top-X recommended algorithms. The results, presented in Figure 10, show that AutoGRD is consistently better than AutoDi, although the gap between the two methods narrows considerably as more algorithms are evaluated.

Figure 10: RMV averaged over 100 regression datasets obtained by AutoGRD and AutoDi: dataset-based method vs the number of recommended algorithms.



It should be noted, however, that AutoGRD achieves an average RMV of 99% after evaluating three algorithms on average, while AutoDi requires five to achieve the same level of performance.

5.8 Running time

One possible shortcoming of AutoGRD is the complexity of the GCD algorithm used to generate the latent meta-features (see Subsection 4.2.2). GCD has a worst-case complexity of $O(V^4)$, where V is the number of vertices in the analyzed graph. Table 8 presents the running time of AutoGRD and AutoDi:dataset-based on different sized datasets, both for classification and regression tasks. As mentioned previously, the running time of Auto-Weka is 13 minutes in average.

Table 8: The running time of AutoGRD on various datasets. The running time format is HH:MM:SS.

Dataset Name	Dataset Size	Task	AutoGRD Running time	AutoDi: dataset-based Running time
breast-tissue	106	classification	00:00:07	00:00:01
autoHorse	203	regression	00:00:04	00:00:01
arrhythmia	452	classification	00:00:23	00:00:26
boston	506	regression	00:00:08	00:00:01
stock	950	regression	00:00:12	00:00:01
house-prices	1460	regression	00:01:31	00:00:02
contrac	1473	classification	00:04:29	00:00:01
cardiotocography-10clases	2126	classification	00:09:21	00:00:01

While it is clear that running AutoGRD on large datasets will require more time, the superior performance of our algorithm is likely to justify the larger computational cost in many cases. One should also bear in mind that solutions such as Auto-Weka require training multiple algorithms on the same dataset whice can also take considerable time, particularly for complex ensemble algorithms. Nonetheless, the complexity of the GCD algorithm is one aspect of the algorithm we plan to address in future research.

6 CONCLUSIONS AND FUTURE WORK

In this study, we introduced AutoGRD, a meta-learning method for algorithm recommendation. By proposing a novel way of representing datasets based on the interactions of their instances, we were able to develop an effective meta-learning method for ranking learning algorithms on previously unseen datasets. Our proposed approach outperformsleading existing solutions such as AutoDi and Auto-Weka, while also proving itself highly effective with "challenging" datasets (e.g., those with few instances or high dimensionality).

For future work, we plan to pursue two research directions. First, we plan to further enhance the capabilities of AutoGRD by enabling it to recommend both algorithms and hyperparameter configurations, and deal with special datasets such as images. Second, we intend to use graph approximation methods in order to create a more efficient, scalable version of the GCD algorithm used in our meta-feature generation.

ACKNOWLEDGMENTS

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) Data-Driven Discovery of Models (D3M) Program.

REFERENCES

[1] H. Bensusan and C. Giraud-Carrier. 2000. Discovering Task Neighbourhoods through Landmark Learning Performances *(PKDD '00)*. 325–330.

[2] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. 2008. *Metalearning: Applications to Data Mining.* Springer Publishing Company, Incorporated.

[3] Pavel B. Brazdil, C. Soares, and Joaquim Pinto da Costa. 2003. Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results. *Machine Learning* 50, 3 (2003), 251–277.

[4] Z. Cao, T. Qin, Tie-Yan Liu, Ming-Feng Tsai, and H. Li. 2007. Learning to Rank: From Pairwise Approach to Listwise Approach *(ICML '07)*. 129–136.

[5] S. Cavallari, Vincent W. Zheng, H. Cai, Kevin Chen-Chuan Chang, and E. Cambria. 2017. Learning Community Embedding with Community Detection and Node Embedding on Graphs *(CIKM '17)*. 377–386.

[6] T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. *CoRR* abs/1603.02754 (2016).

[7] Silvia N. das Dôres, L. Alves, Duncan D. Ruiz, and Rodrigo C. Barros. 2016. A Meta-learning Framework for Algorithm Recommendation in Software Fault Prediction *(SAC '16)*. 1486–1491.

[8] J. Dean and S. Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 107–113.

[9] J. Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.* 7 (2006), 1–30.

[10] I. Drori, Y. Krishnamurthy, R. Rampin, R. Lourenço, J. Ono, K. Cho, C. Silva, and J. Freire. 2018. AlphaD3M: Machine Learning Pipeline Synthesis *(AutoML Workshop at ICML)*.

[11] J. Feng and Z. Zhou. 2018. Autoencoder by forest. In *AAAI Conference on Artificial Intelligence.*

[12] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems? *J. Mach. Learn. Res.* 15 (2014), 3133–3181.

[13] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. 2015. Efficient and Robust Automated Machine Learning *(NIPS'15)*. 2755–2763.

[14] M. Feurer, J. T. Springenberg, and F. Hutter. 2015. Initializing Bayesian Hyperparameter Optimization via Meta-learning *(AAAI'15)*. 1128–1135.

[15] P. Goyal and E. Ferrara. 2018. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowl. -Based Syst.* 151 (2018), 78–94.

[16] A. Grover and J. Leskovec. 2016. Node2Vec: Scalable Feature Learning for Networks *(KDD '16)*. 855–864.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P.r Reutemann, and I. Witten. 2009. The WEKA data mining software: an update. *ACM SIGKDD* 11 (2009), 10–18.

[18] F. Hutter, Holger H. Hoos, and K. Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration *(LION'05)*. 507–523.

[19] G. Katz, E. C. R. Shin, and D. Song. 2016. ExploreKit: Automatic Feature Generation and Selection. In *ICDM.*

[20] C. Lemke, M. Budka, and B. Gabrys. 2015. Metalearning: a survey of trends and technologies. *Artificial Intelligence Review* 44 (2015), 117–130.

[21] L. Li, Kevin G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. 2017. Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits. In *5th International Conference on Learning Representations.*

[22] M. A. Muñoz, Y. Sun, M. Kirley, and S. K. Halgamuge. 2015. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences* 317 (2015), 224 – 245.

[23] R. S. Olson and J. H. Moore. 2016. TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning *(Proceedings of Machine Learning Research)*, Vol. 64. 66–74.

[24] Y. Peng, Peter A. Flach, C. Soares, and P. Brazdil. 2002. Improved Dataset Characterisation for Meta-learning. 141–152.

[25] B. Perozzi, R. Al-Rfou, and S. Skiena. 2014. DeepWalk: Online Learning of Social Representations *(KDD)*. 701–710.

[26] F. Pinto, C. Soares, and J. Mendes-Moreira. 2016. Towards Automatic Generation of Metafeatures. In *Pacific-Asia.* 215–226.

[27] M. D. Plummer. 2007. Graph factors and factorization: 1985–2003: A survey. *Discrete Mathematics* 307 (2007), 791 – 821.

[28] M. Reif, F. Shafait, M. Goldstein, T. Breuel, and A. Dengel. 2012. Automatic Classifier Selection for Non-Experts. *Pattern Analysis and Applications* 17 (2012), 83–96.

[29] L. Rokach. 2016. Decision forest: Twenty years of research. *Information Fusion* 27 (2016), 111 – 125.

[30] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. 2015. LINE: Large-scale Information Network Embedding *(WWW)*. 1067–1077.

[31] C. Thornton, F. Hutter, Holger H. Hoos, and K. Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms *(KDD '13)*. 847–855.

[32] R. Vainshtein, A. Greenstein-Messica, G. Katz, B. Shapira, and L. Rokach. 2018. A Hybrid Approach for Automatic Model Recommendation. ACM, 1623–1626.

[33] J. Vanschoren. 2010. Understanding machine learning performance with experiment databases. *lirias. kuleuven. be* (2010).

[34] Joaquin Vanschoren. 2018. Meta-Learning: A Survey. *CoRR* abs/1810.03548 (2018).

[35] O. Nebil Yaveroglu. 2013. Graphlet correlations for network comparison and modelling : World Trade Network example.

[36] Ö. Yaveroğlu, N.l Malod-Dognin, D. Davis, Z. Levnajić, V. Janjic, R. Karapandza, A.r Stojmirovic, and N. Przulj. 2014. In *Scientific reports.* 4547.

[37] Z. Zhou and J. Feng. 2017. Deep forest: Towards an alternative to deep neural networks. (2017).