RankML: a Meta Learning-Based Approach for Pre-Ranking Machine Learning Pipelines

Doron Laadan[1 *], **Roman Vainshtein**[2 *], **Yarden Curiel**[3], **Gilad Katz**[4], **Lior Rokach**[5]
[1,2,3,4,5] Department of Software and Information Systems Engineering
Ben-Gurion University of the Negev, Beer Sheva, Israel
laadan@post.bgu.ac.il, romanva@post.bgu.ac.il, curiey@post.bgu.ac.il, giladkz@bgu.ac.il, liorrk@bgu.ac.il

Abstract

The explosion of digital data has created multiple opportunities for organizations and individuals to leverage machine learning (ML) to transform the way they operate. However, the shortage of experts in the field of machine learning – data scientists – is often a setback to the use of ML. In an attempt to alleviate this shortage, multiple approaches for the automation of machine learning have been proposed in recent years. While these approaches are effective, they often require a great deal of time and computing resources. In this study we propose RankML, a meta-learning based approach for predicting the performance of whole machine learning pipelines. Given a previously-unseen dataset, a performance metric, and a set of candidate pipelines, RankML immediately produces a ranked list of all pipelines based on their predicted performance. Extensive evaluation on 193 datasets, both in regression and classification tasks, shows that our approach achieves results that are equal to those of state-of-the-art, computationally heavy approaches.

Introduction

Machine learning (ML) has been successfully used in a broad range of applications, including recommender systems (Covington, Adams, and Sargin 2016), anomaly detection (Chandola, Banerjee, and Kumar 2009), and social networks analysis (Wang et al. 2015). This trend has been driven by the enormous growth in the creation of digital data, which enables organizations to analyze and derive insights from almost every aspect of their activities. The growth in the use of ML, however, has not been accompanied by a similar growth in the number of human experts capable of applying it, namely data scientists.

To overcome the shortage in skilled individuals, multiple approaches for automatic machine learning (AutoML) have been proposed in recent years. While earlier studies focused on specific tasks in the ML pipeline – hyper-parameter tuning (Bergstra and Bengio 2012), feature engineering and feature selection (Khalid, Khalil, and Nasreen 2014), etc. – recent studies such as (Drori et al. 2018; Olson and Moore 2019; Thornton et al. 2012; Feurer, Springenberg, and Hutter 2015) seek to automate the creation of the entire ML pipeline end-to-end.

Despite its large diversity, both in the modeling of the problem and in the algorithms used, the field of automatic ML pipeline generation is computationally expensive as well as time consuming. The reasons for these shortcomings include a very large search space both for algorithms and pipeline architectures, the need to perform hyper-parameter optimization, and the fact that evaluating even a single pipeline on a very large dataset may require hours. Another significant shortcoming of most existing approaches is their inability to learn from previously analyzed datasets, which forces them to start "from scratch" with every new dataset. Few approaches such as (Feurer, Springenberg, and Hutter 2015) do try to utilize previous knowledge but do so on a very limited knowledge-base and with basic meta-features.

In this study we present RankML, a novel meta learning-based approach for ML pipeline performance prediction. Given a dataset, a set of candidate pipelines and an evaluation metric (e.g., classification, regression), RankML produces a ranked list of all candidate pipelines based on their expected performance with regard to the metric. This list is produced based on knowledge gained from previously-analyzed datasets and pipelines combinations, without testing any of the pipelines on the current dataset.

We compare the performance of RankML to those of a current state-of-the-art pipeline generation approach – the TPOT (Olson and Moore 2019) framework. The results of the evaluation, conducted both on classification and regression problems, show that our approach achieves comparable results to the baseline at a fraction of the time, with statistical analysis showing that the two are in fact indistinguishable.

Our contributions in this paper are as follows:

- We present RankML, a meta learning-based approach for the ranking of ML pipeline based on their predicted performance. RankML leverages insights from previously analyzed datasets and pipelines combinations, and is therefore capable of producing predictions without running on the current datasets.
- We propose a novel meta-learning approach for pipeline analysis. We derive meta-features both from the analyzed

[0]contributes equally to this work

dataset and the pipeline's topology, and demonstrate that this combination yields state-of-the-art results.

- Finally, we published a large, free access dataset of pipelines, and their performance results on analyzed datasets both in regression and classification tasks. This dataset will be available publicly for any potential future research.

Related Work

Automated machine learning

Automated machine learning (AutoML) is the process of automating the application of machine learning to real-world problems, without human intervention. The goal of this field of research is usually to enable non-experts to effectively utilize "off the shelf" solutions or save time and effort for knowledgeable practitioners.

At its core, the problem AutoML is trying to solve is as follows: given a dataset, a machine learning task and a performance criterion, solve the task with respect to the dataset while optimize the performance (Drori et al. 2018). Finding an optimal solution gets especially challenging due to the growing amount of machine learning models available and their hyper-parameters configurations, which could severely affect the performances of the model (Luo 2016; Thornton et al. 2012).

Multiple approaches have been proposed to tackle the above problem. These approaches range from automatic feature engineering (Katz, Shin, and Song 2017) to automatic model selection (Vainshtein et al. 2018). Some approaches attempt to automatically and simultaneously choose a learning algorithm and optimize its hyper-parameters. This approach is also known as *combined algorithm selection and hyperparameter optimization problem* (CASH) (Thornton et al. 2012; Feurer et al. 2015). More recently, several studies (Drori et al. 2018; Olson and Moore 2019) proposed the automation of the entire work-flow, building a complete machine learning pipeline for a given dataset and task.

Automating the creation of entire ML pipelines is difficult due to the extremely large search space, both of the pipeline architecture and the algorithms that make it up. Furthermore, the fact that the performance of each algorithm is highly dependent on the input it receives from the previous algorithm(s) adds another dimension of complexity. To overcome this challenge, different studies propose a large range of approaches. (Milutinovic et al. 2017). TPOT and Autostacker (Olson and Moore 2019; Chen et al. 2018) for example, use genetic programming to create and evolve the pipelines while auto-weka and auto-sklearn (Thornton et al. 2012; Feurer et al. 2015) use Bayesian Optimization to solve the CASH problem. Another recent approach is used by autoDi (Vainshtein et al. 2018), which applies word embedding of domain knowledge gathered from academic publications and dataset meta-features to recommend a suitable algorithm.

In the majority of cases, most of those methods perform well and produce high, competitive performances results. However, most works in the field suffer from two main shortcomings. First and foremost, applying these approaches is very computationally expensive, with running times that can easily reach days for large datasets (Luo 2016; Olson and Moore 2019; Thornton et al. 2012). The second shortcoming is that most state-of-the-art methods are not sufficiently generic and rely on their underlying code packages to run (e.g., the use of scikit-learn for auto-Sklearn and TPOT). This limitation may prevent automatic pipeline generation frameworks to generalize properly.

Noticeably, several studies propose (albeit partial) solutions to these two challenges. AlpahD3M (Drori et al. 2018) strives to use a broad set of primitives to synthesize a pipeline and set the appropriate hyper-parameters no matter the data, while autoDi generates a model offline that can be applied almost instantly at runtime. In addition, auto-sklearn uses a meta-learning approach to decrease the time of the Bayesian optimization problem (Feurer, Springenberg, and Hutter 2015). Additional solutions are also proposed in the literature.

Meta-learning

Meta-learning, or learning to learn, is commonly used to describe the scientific approach of observing different machine learning algorithms performances on a range of learning tasks. We then use those observations – the meta-data – to learn a new task or to improve an existing algorithm's performance. Simply put, meta-learning is the process of an understanding and adapting learning itself on a higher level (Lemke, Budka, and Gabrys 2015). Instead of starting 'from scratch', we leverage previously-gained insights.

Throughout the years, multiple studies have been exploring the application of meta-learning in various domains such as ensemble methods (Brazdil et al. 2008; Wolpert 2002; Sagi and Rokach 2018), algorithms recommendations (Vainshtein et al. 2018; Brazdil, Soares, and Da Costa 2003), meta-learning systems and transfer learning (Vilalta and Drissi 2002). Many state-of-the-art AutoML methods use meta-learning as a way of improving their accuracy and speed (Feurer, Springenberg, and Hutter 2015; Santoro et al. 2016) and multiple studies describe ways to create meta knowledge usable by machine learning algorithms (Brazdil, Soares, and Da Costa 2003; Katz, Shin, and Song 2017).

Meta knowledge usually involves creating significant and meaningful meta-features on the datasets or the models used (Feurer, Springenberg, and Hutter 2015; Katz, Shin, and Song 2017; Vainshtein et al. 2018). The majority of meta-features can be divided into five "families", as shown in (Feurer, Springenberg, and Hutter 2015). Examples of such families include Landmarking (Pfahringer, Bensusan, and Giraud-Carrier 2000), which achieves state-of-the-art results but is computationally-heavy. Another example is the derivation of meta-features on performance, which is easy to extract but does not necessariliy yield optimal results (Lemke, Budka, and Gabrys 2015). The design of such meta-features, although an important process, is also considered a challenge for meta-learning (Brazdil et al. 2008; Brazdil, Soares, and Da Costa 2003). For that reason, In recent years several frameworks were proposed for the automatic extraction of meta-features such as as(Pinto, Soares,

and Mendes-Moreira 2016).

Recently, several studies proposed the use of meta-features and learning to improve the AutoML process. AutoDi (Vainshtein et al. 2018) and AutoGRD (Cohen-Shapira 2019) , for example, use meta-features of datasets to rank different machine learning algorithms and already achieved good results in model recommendation task. Katz et al. (Katz, Shin, and Song 2017), used meta-features for automatic feature engineering. Our approach will utilize these meta-features with a combination of pipelines topology. It will be, to the best of our knowledge, the first time such combination is explored.

Problem Formulation

We use the same notations and definitions as previous works in this field, specifically (Olson and Moore 2019) and (Drori et al. 2018):

Primitives. A set of different algorithms that can be applied to a dataset as part of the data mining work-flow. We divide primitives into four different families:

- **Data pre-processing.** Consisting ot algorithms for data cleaning, balancing, resampling, label encoding, and missing values imputation.
- **Feature pre-processing.** Consisting of algorithms such as PCA and SMOTE.
- **Features engineering and extraction.** Consists of algorithms used for discretization and feature engineering.
- **Predictive models.** Consists of all algorithms used tp produce a final prediction. This family includes algorithms for classification, regression, ranking etc. Relevant deep architectures are also included in this family.

Pipeline. A directed acyclic graph (DAG) $G = \{V, E\}$, where the vertices of the graph are *primitives* and the edges of the graph determine the primitives' order of activation and input. The pipeline constitutes a complete data mining work-flow: its design can be a simple one-way DAG in which each primitive output is the input to the next primitive or it could have a more complex design in which the input for a primitive is the concatenation of the output of several primitives.

Objective function. We define a *AutoML task* $\mathcal{T}(D, T, M)$ consisting of tabular dataset D with m columns and k instances, a machine learning task T and a performance metric M. Additionally, we assume a list of candidate machine learning pipelines, or *pipelines* $\mathcal{C} = \{c_1, c_2 \ldots, c_{N_c}\}$ for the given AutoML task. Given that pipeline $c \in \mathcal{C}$ is able to produce predictions over the specific AutoML task $\mathcal{T}$, our goal is to produce an ordered list of the candidates pipelines $\mathcal{C}_{Ranked}$, order by the following function:

$$\arg\min_{c \in \mathcal{C}} \mathcal{E}(\mathcal{T}(D, T, M), c)$$

Where $\mathcal{E}$ is the error of pipeline c over the AutoML task $\mathcal{T}$.

The Proposed Method

Overview. Our process is presented in Figure 1. It consists of an *offline* and an *online* phase. In the offline phase, we generate and train multiple pipeline architectures on a large set of datasets and record their performance. Also, we extract meta-features that model both the dataset, the pipeline, and their interdependence. We then use these meta-features to train a ranking algorithm capable of scoring the final performance of given dataset-pipeline combinations without actually running them

In the online phase RankML receives a previously unseen dataset, a set of candidate pipelines and an evaluation metric. We then extract meta-features describing both the dataset and each of the candidate pipelines and use the ranking algorithm to produce a ranked list of the pipelines. The top-ranked pipelines are then evaluated, and the actual performance is recorded and added to our knowledge-base for future use.

It is important to point out that RankML is not limited in anyway in its source for candidate pipelines. The pipelines can be randomly generated or received from other pipeline generation frameworks. In this sense, our approach can function both as a stand-alone ML pipeline recommendation framework and as a preliminary step for other, more computationally intensive solutions.

In the remainder of this section we present the processes we use to extract the various meta-features used by our model. We then describe the process of training the meta-model.

Dataset Meta-Features

To create our dataset meta-features we build upon the previous work of (Katz, Shin, and Song 2017) and (Vainshtein et al. 2018) who successfully used dataset-based meta-features for AutoML related tasks. Our meta-features combine elements from both studies and can be divided into two groups:

- **Descriptive.** Used to describe various aspects of the dataset. This group of meta-features includes information such as the number of instances in the data, number of attributes, percentage of missing values and likewise.
- **Correlation-based.** Used to model the interdependence of features within the analyzed datasets. Meta-features of this group include correlation between different attributes and the target value, Pearson correlation between attributes and different aggregations such as average and standard deviation (among others).

Pipeline Representation Meta-Features

In order to make the pipeline representation compact and extendable, we chose to represent the pipeline's topology as a sequence of words. Each type of primitive is assigned a unique fixed-length hash, that is used to represent it. Next, we construct a sequence of hashes, to represent each pipeline with the order of hashes, determined by the pipeline's topology.

In order to make all pipeline representations consistent, we use the following rules to generate the representation:

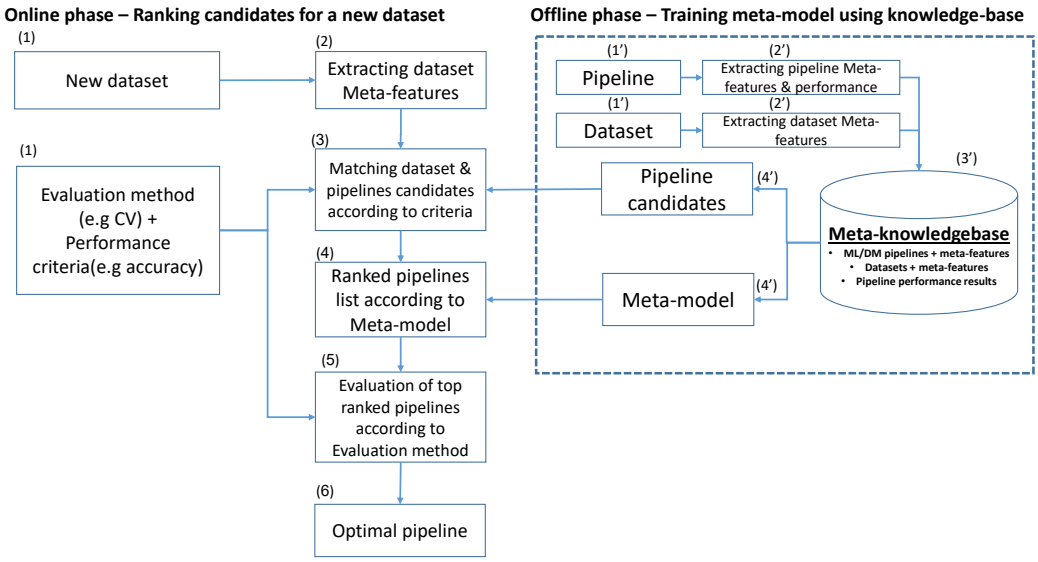

Figure 1: The Meta-model work-flow. in the *offline* phase knowledge is gathered and a ranking algorithm is trained. in the *online* phase a new dataset and task are presented. meta features are extracted from the dataset and a list of candidate pipelines is acquire from the knowledge base according to the task at hand. the ranking model is used to ranked the pipelines and the optimal pipeline is recommended.

- We sequence the pipeline in reverse order – from the final output to the inputs. A primitive has to be sequenced prior to any of its input primitives. This is the case both for primitive with single inputs and multiple inputs (like the combiner primitive in Figure 2)
- In the case of multiple or parallel sub-pipelines (as in Figure 2), the longest sub-pipeline is processed first. Ties are broken randomly.
- In order to make the representation consistent in length for all pipelines, we define a fixed maximal number of primitives for all pipelines. In the case of smaller pipelines, padding (in the form of a designated "blank" primitive) is used.

An example of such transformation on a TPOT based pipeline can be seen in figure 2. Using our approach, the representation of the pipeline will be as follows: *[Combiner,Primitive3,Primitive2,data,Primitive1,data]*

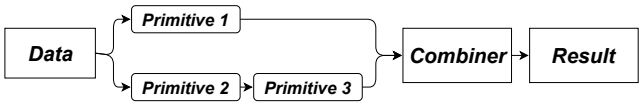

Figure 2: An example of a pipeline.

Training the Meta-Model

Following the creation of the meta-features representing both the dataset and the various pipelines, we train the meta-model used for the pipeline ranking. The training of the meta-model is performed offline on a large knowledge base consisting of multiple datasets and ML pipeline architectures. The offline phase in Figure 1 provides an overview of the process.

The training process is carried our as follows: for each dataset D in the knowledge base, we retrieve all possible task T (e.g., classification, regression) and evaluation metric M (e.g., AUC, accuracy). For each combination of $\{D, T, M\}$ we generate a large set of candidate pipelines C. We then train all combinations of $\{D, T, M, c\}$ where $c \in C$. The performance of all the pipelines are stored in the knowledge base for future use.

Finally, for each task and evaluation metric, we train a ranking algorithm using the information gathered during the offline evaluation described above. For each evaluated dataset and pipeline combination, we extract their corresponding meta-features and concatenate them. The joined meta-features vectors are used to train the ranking algorithm. The goal of the algorithm is to produce a list of all participating pipelines, ordered by their respective performance on the dataset.

Evaluation

Experimental Setup

We evaluate our proposed approach on two common tasks in the field of machine learning: classification and regression. For each task we assembles its own set of datasets and pipelines and trained a separate meta-model for the ranking task.

Dataset. We used 106 classification datasets and 87 regression datasets previously used in (Cohen-Shapira 2019).

These datasets are highly diverse with respect to number of instances, number of features, feature composition etc. All datasets are available in the following online repositories: UCI[1], OpenML[2] and Kaggle[3].

Pipelines generation. All the pipelines used in our training and evaluation were generated using TPOT (Olson and Moore 2019) a state-of-the-art framework for automatic pipeline generation and exploration. The pipelines generated by TPOT consist entirely of algorithms that can be found in the python *scikit-learn* package. TPOT uses genetic algorithms to iteratively improve is generated pipelines. Moreover, TPOT supports the creation of parallel pipelines, an option that greatly increases the diversity of our pipelines population.

We ran TPOT on each of our datasets and collected all the architectures generated during runtime. We used TPOT default settings – 100 pipelines per generation for 100 generations with a default primitives dictionary consisting of 30 primitives for classification tasks and 29 for regression tasks. This process resulted in an average of 9700 pipelines per dataset. Since TPOT generates some pipelines for multiple datasets, we were able to obtain both pipelines that are unique to specific datasets and pipelines that are trained on multiple datasets. The former group provides our model with diversity, while the latter provides a useful information on dataset-pipeline interactions.

While TPOT also performs hyper-parameter optimization in addition to its pipeline search, we consider this topic to be beyond the scope of our current work. Therefore, in cases where TPOT generated multiple pipelines with the same topology we record the performance of the top-performing pipeline. As a result, our knowledge base consisted of $179,905$ classification pipelines and $230,745$ regression pipelines. We make our entire database (datasets, pipeline architectures, and their performance) publicly available [4].

It is important to note that while our current knowledge base is comprised solely from TPOT generated pipelines, all of our meta-features are generic and can be applies to any type of ML pipeline representation. Our reasoning for using TPOT as the source of our pipelines is twofold: first, it is a state-of-the-art pipeline generation platform, so the chances of having at least some high-performing architectures to detect is high. Secondly, since we compare RankML's perofrmance to that of TPOT, having the same architectures in both cases ensures a fair comparison.

Meta-learner implementation. For our meta-learner we used XGBoost (Chen and Guestrin 2016), specifically the XGBRanker model with the pairwise ranking objective function. Previous work (Cao et al. 2007) has shown that it is highly suitable for producing ranked lists. Additionally, we used the following hyper-parameters settings: learning rate of 0.1, max depth of 8 and 150 estimators. We set $k = 10$ as the number of best pipelines the ranker returns. We set the algorithm's parameters empirically using the leave-one-out approach. Our model contains shallow trees of 150 estimators. A shallow tree is appropriate because we have few instances and bushy tree tends to overfit the data in this case.

Evaluation method. To test our ranking model we used a leave-one-out validation method. During the training phase, given the set of datasets d_i, we train a meta-model $\mathcal{M}_{d_i}$ using all remaining datasets. This resulted in creating 107 different meta-models for classification and 87 for regression that were used in the experiment.

During the test phase, for each $d_i \in D$, we used the matching $\mathcal{M}_{d_i}$ meta-model to rank all pipelines of the same task in our knowledge with the current dataset meta-features as describes in the online phase in Figure 1. Each dataset was split into train and test sets using a 75%/25% ratio. The K top-ranked pipelines (by $\mathcal{M}_{d_i}$) were then trained on the training set of dataset d_i and evaluated on its test set. The results of this evaluation were the "ground truth" against which we compared the performance of [Doron: TPOT?] RankML.

Baselines. We chose to compare the performance of our proposed approach to those of TPOT, a state-of-the-art automatic pipeline generation platform. It is important to stress again, however, that while TPOT evaluated each generated pipeline by running it on the evaluated dataset, RankML immediately produces a ranked list using its meta-model at minimal computational cost.

Results and Discussion

Classification results. Since we use TPOT default parameters, the framework generates 10,000 pipelines for each dataset. All pipelines are then evaluated on the dataset's training set, and finally a single pipeline is produced. RankML, on the other hand, utilizes the meta-model to rank *all* the pipelines in the knowledge base with respect to the analyzed dataset and then returns its top-ranked pipelines. These pipelines are then trained on the dataset's train set and evaluated on the test set. It is important to note that if RankML evaluates the top-10 ranked pipelines, than it is still more efficient than TPOT by three orders of magnitude (10 to 10,000).

The results of the evaluation on 106 datasets are presented in Tables 1 and 2. It is clear that RankML's performance is very close to those of TPOT's even though it does not run any pipelines on the dataset prior to the recommendation. When recommending only a single pipeline, RankML's average performance is 93.5% of that of TPOT's. When 10 pipelines are recommended, that figure rises to 98.18%. Additionally, Table 2 shows that the percentage of datasets in which RankML achieved better-or-comparable perfomance to TPOT is 68%-73% (depending on the number of evaluated pipelines).

Figures 3 - 5 provide further analysis of RankML's performance. Figure 3 presents the performance of each framework on all classification datasets. Figure 4 presents the

[1]https://archive.ics.uci.edu/ml

[2]https://www.openml.org

[3]www.kaggle.com

[4]the knowledge base and meta learner will be made available pending acceptance

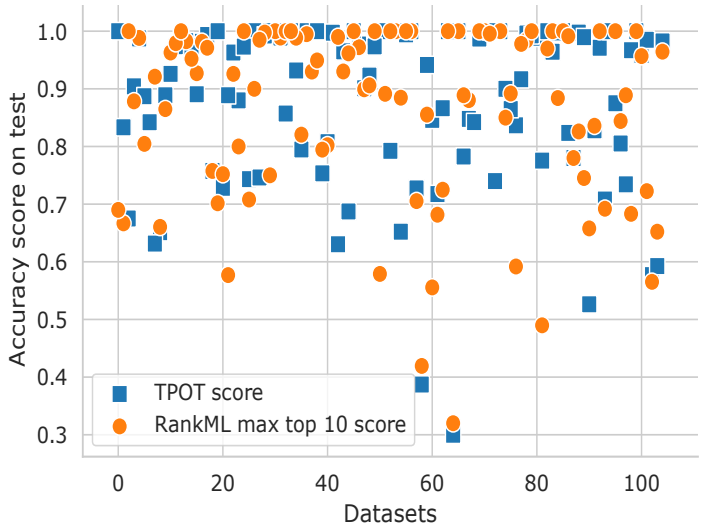

Figure 3: A scatter diagram showing the accuracy on each of the 106 datasets. We present the results for RankML max of top-10 ranking as well as TPOT.

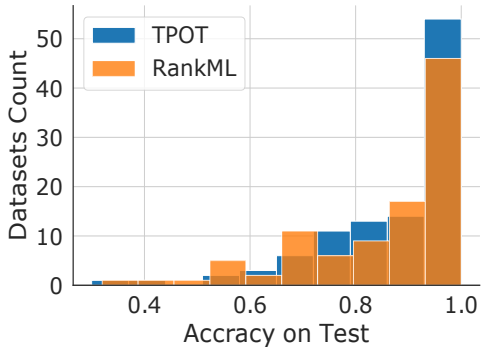

Figure 4: A histogram showing accuracy scores across 106 datasets. We present the results for RankML max of top-10 ranking as well as TPOT.

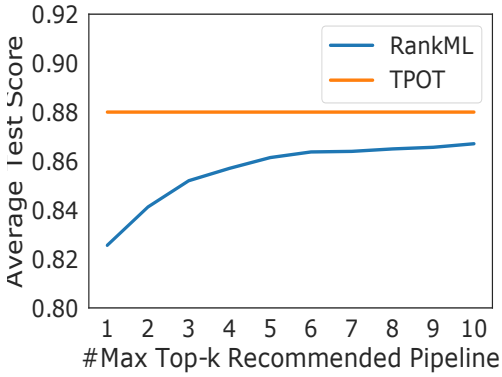

Figure 5: A plot showing the average test scores RankML achieves when using the optimal pipeline out of k-top recommendations for 106 classification datasets.

number of datasets that reached a specific level of accuracy using either TPOT or RankML.

[Doron: I cant understand why this part is here. didnt you said all of this above?] Table 1 and Figures 3 - 5 show the results of our evaluation across the 106 classification datasets. We compare the result of the first rank pipelines by RankML as well as the best pipeline out of the top five and ten recommendations RankML produce, against the baseline. RankML achieves state-of-the-art results when taking the maximum value out of the top ten recommended pipelines while having low standard deviation values. [Doron: here]

Figure 5 presents the average number of pipelines per dataset that need to be evalued in order to reach speicific levels of accuracy (TPOT's performance is presented as an upper bound). All analysis points to the fact that RankML achieves a level of perofmrance that is very close to – and sometimes surpasses – that of TPOT.

Finally, we use the Wilcoxon signed-rank test to determine whether the accuracy-based performance of TPOT is significantly better than that of our proposed approach. Using a confidence level of 95%, we were *not* able to reject the null hypothesis, meaning that there is no significant difference in the performance of the two approaches.

Method	**Average Accuracy**	**stdev**
TPOT	**0.883**	0.142
RankML #1 rank	0.826	0.180
RankML Max top-5 rank	0.861	0.154
RankML Max top-10 rank	0.867	0.153

Table 1: Average accuracy results across 106 datasets.

To test the diversity of our method we analyzed the pipelines produced by RankML. Table 4 presents the percentage of times primitives were used in RankML top-10 pipelines. RankML appears to often recommend pipelines

Method	Number of Datasets with BOC Performance(%)
RankML Max top-5 rank	67 (68%)
RankML Max top-10 rank	**72 (73%)**

Table 2: The number of classification datasets each method got better or comparable results against TPOT.
percentage is out of valid datasets.

that use some form of pre-processing primitives. This can be expected as most of the time pre-processing data can lead to better performances. Table 4 also shows that there are no dominant primitives in the recommendations, and the most used primitive appears only in 16% of the pipelines.

Regression results.

We conduct our evaluation of 87 datasets and use mean squared error (MSE) as our evaluation metric. The results of our evaluation are presented in Table 3, which shows that the percentage of dataset in which RankML achieved better-or-comparable performance to TPOT is 63%-67% (depending on the number of evaluated pipelines). Again, this result is particularly impressive given the fact that RankML does not conduct any evaluation on the analyzed dataset.

Figure 6 plots for each approach the number of dataset in which it outperformed the other as a function of K (the number of top-ranked pipelines evaluated). The results clearly show that RankML outperforms TPOT for $K \geq 4$, which means we only have to evaluate four pipelines on average to outperform our baseline.

Method	Number of Datasets with BOC Performance(%)
RankML Max top-5 rank	40 (63%)
RankML Max top-10 rank	42 (67%)

Table 3: The number of regression datasets each method got better or comparable results against TPOT.
percentage is out of valid datasets.

Discussion. Our evaluation clearly shows that RankML is able to achieve results that are either comparable to the state-of-the-art (classification) or significantly surpass it (regression). However, another significant metric is the required running time: while TPOT require 57 minutes per dataset on average, RankML requires only 50 seconds. These results are further proof to the effectiveness of our approach.

Conclusion - Future work

In this study we presented RankML, a novel meta learning-based approach for ranking machine learning pipelines. By exploring the interactions between datasets and pipeline topology, we were able to train learning models capable of identifying effective pipelines without performing computationally-expensive analysis. By doing so, we address one of the main shortcomings of AutoML-based systems: long running times and computational complexity.

For future work, we plan to extend and test our method on different machine learning tasks. Additionally, we intend to explore more advanced meta-representations both for the datasets and pipelines. Finally, we intend to use our method as a step for improving existing AutoML systems.

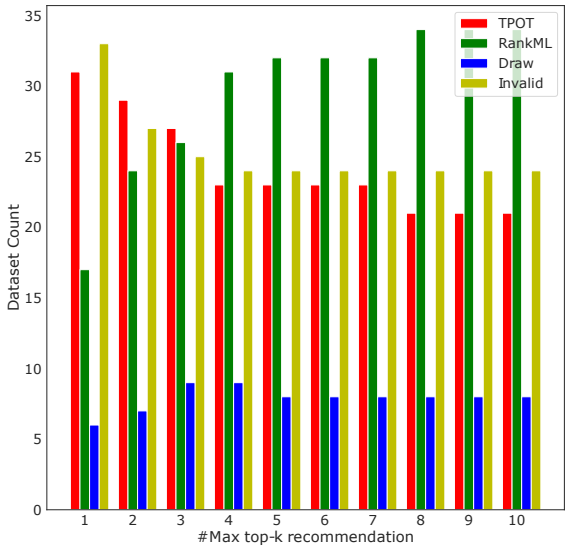

Figure 6: A bar plot showing the number of regression datasets each method receive better score on the test set when RankML used the best pipeline overall. Invalid (yellow) means that either TPOT or the recommended pipeline couldn't run on the test set for various reasons which include a known issue in the TPOT framework [6]. Draw means that there was less then 0.01 diffrence in score

Primitive	Family	Avg % of appearances
MaxAbsScaler	Data preprocessing	16%
StandardScaler	Data preprocessing	11%
KNeighborsClassifier	Predictive models	10%
RandomForestClassifier	Predictive models	4%
PCA	Feature preprocessing	0.4%

Table 4: A selection of primitives used in RankML recommended pipelines for classification. "Avg % of appearances" is out of all primitives.

References

[Bergstra and Bengio 2012] Bergstra, J., and Bengio, Y. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13(Feb):281–305.

[Brazdil et al. 2008] Brazdil, P.; Carrier, C. G.; Soares, C.; and Vilalta, R. 2008. *Metalearning: Applications to data mining*. Springer Science & Business Media.

[Brazdil, Soares, and Da Costa 2003] Brazdil, P. B.; Soares, C.; and Da Costa, J. P. 2003. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning* 50(3):251–277.

[Cao et al. 2007] Cao, Z.; Qin, T.; Liu, T.-Y.; Tsai, M.-F.; and Li, H. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, 129–136. ACM.

[Chandola, Banerjee, and Kumar 2009] Chandola, V.; Banerjee, A.; and Kumar, V. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41(3):15.

[Chen and Guestrin 2016] Chen, T., and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794. ACM.

[Chen et al. 2018] Chen, B.; Wu, H.; Mo, W.; Chattopadhyay, I.; and Lipson, H. 2018. Autostacker: A compositional evolutionary learning system. *arXiv preprint arXiv:1803.00684*.

[Cohen-Shapira 2019] Cohen-Shapira, Noy, R. R. V. G. K. R. L. 2019. Autogrd: Model recommendation through graphical dataset representation. In *Proceedings of the 28th international conference on Machine learning*. ACM.

[Covington, Adams, and Sargin 2016] Covington, P.; Adams, J.; and Sargin, E. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, 191–198. ACM.

[Drori et al. 2018] Drori, I.; Krishnamurthy, Y.; Rampin, R.; De, R.; Lourenco, P.; Ono, J. P.; Cho, K.; Silva, C.; and Freire, J. 2018. AlphaD3M: Machine Learning Pipeline Synthesis. *JMLR Work. Conf. Proc.* 1:1–8.

[Feurer et al. 2015] Feurer, M.; Springenberg, J. T.; Klein, A.; Blum, M.; Eggensperger, K.; and Hutter, F. 2015. Efficient and Robust Automated Machine Learning. *Proc. 28th Int. Conf. Neural Inf. Process. Syst.* 2755–2763.

[Feurer, Springenberg, and Hutter 2015] Feurer, M.; Springenberg, J. T.; and Hutter, F. 2015. Initializing Bayesian Hyperparameter Optimization via Meta-Learning. *Aaai* 1128–1135.

[Katz, Shin, and Song 2017] Katz, G.; Shin, E. C. R.; and Song, D. 2017. ExploreKit: Automatic feature generation and selection. *Proc. - IEEE Int. Conf. Data Mining, ICDM* 979–984.

[Khalid, Khalil, and Nasreen 2014] Khalid, S.; Khalil, T.; and Nasreen, S. 2014. A survey of feature selection and feature extraction techniques in machine learning. In *2014 Science and Information Conference*, 372–378. IEEE.

[Lemke, Budka, and Gabrys 2015] Lemke, C.; Budka, M.; and Gabrys, B. 2015. Metalearning: a survey of trends and technologies. *Artif. Intell. Rev.* 44(1):117–130.

[Luo 2016] Luo, G. 2016. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Netw. Model. Anal. Heal. Informatics Bioinforma.* 5(1):1–16.

[Milutinovic et al. 2017] Milutinovic, M.; Baydin, A. G.; Zinkov, R.; Harvey, W.; Song, D.; Wood, F.; and Shen, W. 2017. End-to-end training of differentiable pipelines across machine learning frameworks.

[Olson and Moore 2019] Olson, R. S., and Moore, J. H. 2019. TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning. 151–160.

[Pfahringer, Bensusan, and Giraud-Carrier 2000] Pfahringer, B.; Bensusan, H.; and Giraud-Carrier, C. 2000. Meta-Learning by Landmarking Various Learning Algorithms. *Proc. Seventeenth Int. Conf. Mach. Learn. ICML2000* 951(2000):743–750.

[Pinto, Soares, and Mendes-Moreira 2016] Pinto, F.; Soares, C.; and Mendes-Moreira, J. 2016. Towards automatic generation of metafeatures. In *Pacific-Asia*, 215–226.

[Sagi and Rokach 2018] Sagi, O., and Rokach, L. 2018. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8(4):e1249.

[Santoro et al. 2016] Santoro, A.; Bartunov, S.; Botvinick, M.; Wierstra, D.; Lillicrap, T.; and Deepmind, G. 2016. Meta-Learning with Memory-Augmented Neural Networks Google DeepMind. *Jmlr* 48:1842–1850.

[Thornton et al. 2012] Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2012. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms.

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

[Vilalta and Drissi 2002] Vilalta, R., and Drissi, Y. 2002. A perspective view and survey of meta-learning. *Artif. Intell. Rev.* 18(2):77–95.

[Wang et al. 2015] Wang, P.; Xu, B.; Wu, Y.; and Zhou, X. 2015. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences* 58(1):1–38.

[Wolpert 2002] Wolpert, D. H. 2002. The supervised learning no-free-lunch theorems. In *Soft computing and industry*. Springer. 25–42.