Niseko: a Large-Scale Meta-Learning Dataset

Zeyuan Shang [1], Emanuel Zgraggen [1], Philipp Eichmann [2], Tim Kraska [1]
Massachusetts Institute of Technology[1], Brown University[2]
{zeyuans, emzg, kraska}@mit.edu, peichman@cs.brown.edu

Abstract

Meta-learning has been used as an efficient technique to speed up building predictive models by learning useful experiences in a data-driven way. Previous works either focus on devising meta-learning algorithms for a specific category of tasks or using it as a building block in a machine learning system. Meta-learning data, however, can have impact much beyond this scope if it is publicly available. It can enable machine learning practitioners to take data-driven decisions at all steps of their workflows. In this paper, we present Niseko, an open-source large-scale meta-learning dataset with more than 5 million pipelines and 50 million pipeline runs on 300 datasets with clear-defined structures and easy-to-use APIs. We demonstrate Niseko's usefulness through several use cases.

1 Introduction

Meta-learning, or *learning to learn*, observes how various machine learning models perform across different datasets and tasks, and utilizes the learned meta-knowledge to speed up predictive modelling on new tasks. Besides, it also provides a data-driven perspective to understand how machine learning pipelines and primitive perform on different datasets and the underlying latent relationship between these seemingly unrelated datasets.

There have been lots of studies on meta-learning, and [6] gives an overview of meta-learning techniques, including learning from model evaluations, tasks properties and prior models. Prior work solely focuses on an algorithmic perspective of meta-learning, e.g., how it is used to tackle a specific category of tasks, or describe meta-learning as a building block of a automated machine learning system (AutoML), as in Auto-sklearn [4]. However, meta-learning data could enable and support many more use cases beyond the scope of AutoML, but is unfortunately oftentimes not readily available.

Inspired by this we release *Niseko*: a large-scale meta-learning dataset with easy-to-use APIs for analytics and production use. By randomly sampling machine learning pipelines from a search space consisting of 46 common machine learning primitives associated with hyper-parameters and running these pipelines on different sample sizes, we collected more than 5 million pipelines and 50 million pipeline runs on 300 datasets. Based on our experiences on analyzing this data and using it extensively in systems that heavily rely on meta-learning, we devise a set of powerful and easy-to-use APIs for Niseko. To enable reproducibility, Niseko supports converting a pipeline description to an executable Python script, therefore users are able to execute the pipeline inside their development environment.

We demonstrate the usefulness of Niseko and its associated APIs through a set of diverse use cases, including statistical analysis of primitives, effectiveness of meta-features, predicting model behaviors and meta-learning for AutoML. Ultimately, we believe that having access to a large-scale meta-learning repository can benefit machine learning practitioners and machine learning researchers by enabling them to make data-driven decision about choices in their workflows.

3rd Workshop on Meta-Learning at NeurIPS 2019 NeurIPS 2019, Vancouver, Canada.

2 Meta-Data Acquisition

We base our meta-data acquisition on an AutoML system *BlindedForReviewing* whose search space includes 17 feature processing primitives (i.e., encoding, scaling, feature selection), 15 classifiers, 14 regressors and their corresponding hyper-parameter distributions. All primitives are shown in Section A of the supplementary material. The search space is constructed by applying hand-crafted rules that with randomness such that primitives can be combined in any way (e.g., run min-max scaling on one column and standard scaling on another column, followed by a PCA).

2.1 Pipeline and Pipeline Run

Each pipeline is an end-to-end solution to a task, represented as a Directed Acyclic Graph (DAG) of primitives with fixed hyper-parameters. An example of pipeline is shown in Figure 1. Running such a pipeline on a dataset will produce a pipeline run, which consists of runtime information, such as training and testing time, predictive performance (e.g., accuracy).

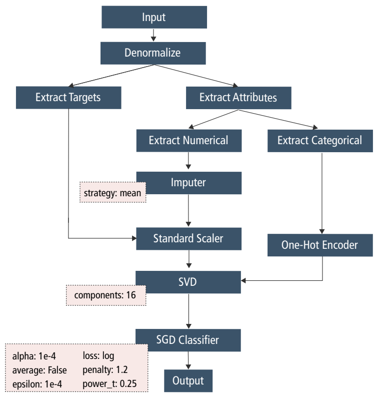


Figure 1: An example pipeline. The boxes in red show fixed hyper-parameters and they compose a pipeline with this DAG.

2.2 Pipeline Generation and Execution

Figure 2 visualizes an example search space. To promote the exploration of the heterogeneous and infinite search space, we employ random search to retrieve pipelines for evaluation. To be able to study the impact of training data size on pipeline performance, we first split the data into training and validation set, and then split the training data into ten equal-sized splits. We then train a pipeline on the first k splits where $1 \leq k \leq 10$ and report their performance on training and validation set. Therefore we gather ten pipeline runs for each evaluated pipeline.

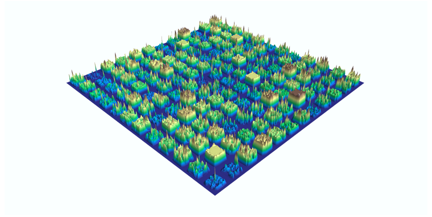


Figure 2: A visualization of an example search space. Each point is a pipeline and its height over the underlying ground denotes the performance, we grid the search space based on the models of pipelines.

2.3 Datasets, Tasks and Results

We use 300 tabular datasets provided by the DARPA Data Driven Discovery of Models program [2]. These 300 datasets are an aggregation from datasets on OpenML [7] and the UCI Machine Learning Repository [3]. Out of these 300, 220 are classification tasks, the smallest containing 151 records, the largest being 1025000 records large. The remainder are regression tasks, with sizes ranging from 159 to 89640 records. All the tasks and their sources are listed in the supplementary material.

For each pipeline, we store its description (including the structure and hyper-parameters), training time (on the full training set), test time and performance (e.g., accuracy). For each pipeline run, we store the size of training sample, training and validation performance (e.g., accuracy) and training and validation time. In other words, we can either look at the internal results (i.e., pipeline runs) to understand how a pipeline performs in training and validation, or look its final performance on the test set (from the pipeline).

For each dataset, we run our execution framework for twelve trials on a 20-core Intel(R) Xeon(R) CPU @ 2.30GHz machine with 60GB memory. For each trial we enforce a one-hour limit. Ultimately, we collect 5,274,555 pipelines and 51,204,676 pipeline runs. Note that since we keep a strict one-hour limit, some pipelines have not finished their runs, therefore the number of pipeline runs is not exactly ten times the number of pipelines.

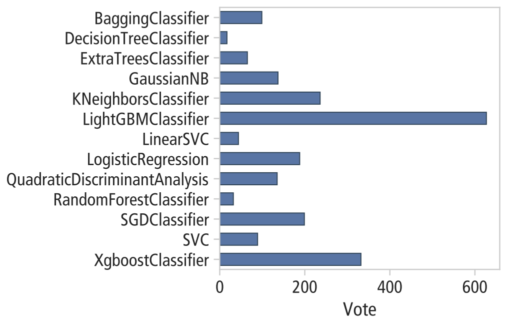
3 Niseko Use Cases

Our goal with Niseko is to help machine learning developers and researchers to make data-driven decisions when it comes to design machine learning solutions or systems. Having access to a large repository of meta-learning data has some obvious applications, such as avoid training a model from scratch or "warm-starting" an AutoML search process, but we believe it can be used in a variety of other scenarios.

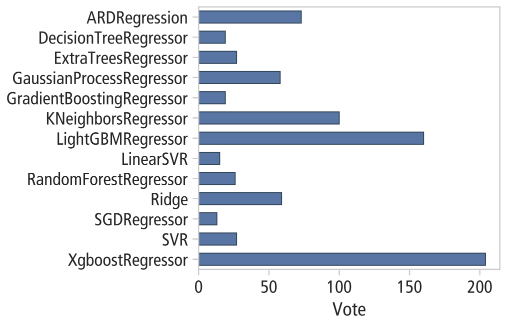
In this section we present a set of use cases and ultimately insights that can be gained through Niseko. These range from understanding and choosing what models and preprocessors to try when creating a new machine learning workflow (Section 3.1 and 3.2), over predicting model performance given a dataset, to simulating effectiveness of algorithms and building agents to replace heuristics in AutoML systems (see Appendix). Python notebooks of all the use cases can be found at `https://github.com/niseko-submission/niseko_submission`.

3.1 Performance of Models

We run a detailed analysis of the relative performance of predictive models in Niseko. For each dataset, we pick the top 10 pipelines and give the associated model of those pipelines a "vote". We sum up the votes of each model over all the datasets and visualize them in Figure 3. Overall, LightGBM [5] and Xgboost [1] showed the most robust performance (getting the most overall votes). While Decision Tree, LDA, Gaussian NB, Bayesian ARD Regression are rarely performing well. Figure 4, where we change the voting scheme to just include the top 1 performing pipeline, shows a paints a similar picture.



(a) Classification

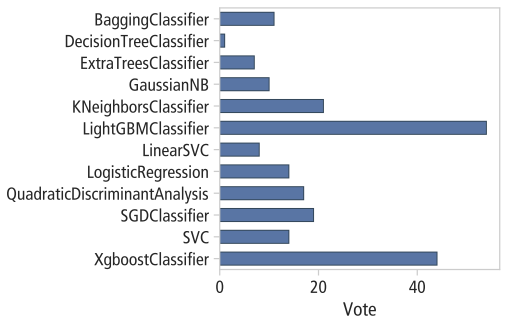


(b) Regression

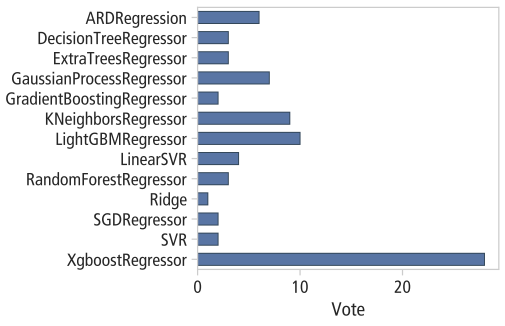
Figure 3: Model Votes (top 10)

3.2 Effectiveness of Feature Preprocessing

We perform a similar analysis for certain feature preprocessing methods. Specifically, we evaluate if one-hot encoding helps overall pipeline performance. For the datasets containing categorical features, we count how many of top 10 pipelines are with and without one-hot encoding. The results are shown in Figure 5. There are 73 datasets with categorical features and one-hot encoding helps in

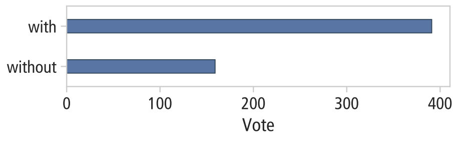


(a) Classification

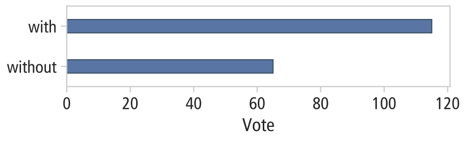


(b) Regression

Figure 4: Model Votes (top 1)

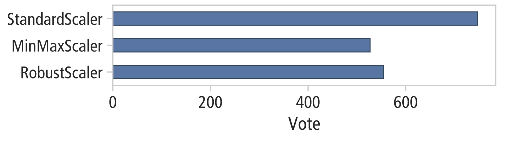


(a) Classification

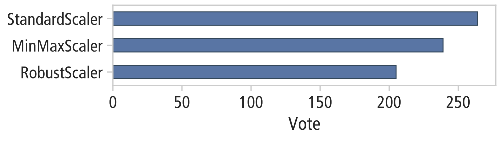


(b) Regression

Figure 5: One-hot encoding votes



(a) Classification

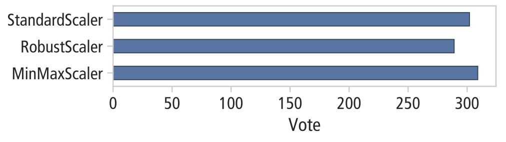


(b) Regression

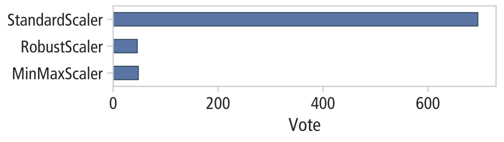
Figure 6: Scaling votes

most cases (as opposed to using ordinal encoding). However, there are 12 datasets where one-hot encoding doesn't help. We further drill down into these 12 and find that there are several reasons for this: many numerical features therefore categorical features are less important, the cardinality of categorical features is either too high (e.g., "name", "ID") or too low (e.g., for binary features ordinal encoding and one-hot encoding are essentially the same) or ordinal encoding is enough (e.g., for "level of education" where there exists a natural ordering).

We employ the same analysis for numeric scaling methods (note that a pipeline may use separate scaling methods for different features) in Figure 6 and found that for classification tasks, standardization is on average most helpful. While for regression tasks the differences overall seem rather minor. However, if we drill down into specific models we observe large differences. For example, Xgboost Regressor performance is largely indepenent on the sacling method that is applied, whereas SVR depends heavily on being combined with Standard Scaling (Figure 7).



(a) XgboostRegressor



(b) SVR

Figure 7: Scaling votes for particular models

4 Conclusion

We present Niseko, a large-scale meta-learning dataset and its associated APIs. Through a set of use cases we demonstrate the value of Niseko in extracting insights about machine learning pipelines and primitives, and their behaviour across different datasets. Ultimately, we believe that Niseko can benefit machine learning practitioners by enabling them to make data-driven decision when creating individual solutions for given problems, and machine learning researchers and system developers when building and prototyping algorithmic designs and automated systems. We have released all of our example code and data at `https://github.com/niseko-submission/niseko_submission`.

Acknowledgement

This research is funded by the DARPA Award 16-43-D3M-FP-040, NSF Award RI-1813444, NSF Award IIS-1562657 and NSF Award IIS-1514491 and supported by Google, Intel, and Microsoft as part of the MIT Data Systems and AI Lab (DSAIL).

References

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

[2] DARPA. *Data-Driven Discovery of Models (D3M).*

[3] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[4] Matthias Feurer et al. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[6] Joaquin Vanschoren. Meta-learning. pages 39–68. Springer, 2018. In press, available at http://automl.org/book.

[7] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.