OrpheusDT - Orpheus Decision Tree/Database for Training

A Python Module for Managing Machine Learning Projects

Yi-Hsuan Lin
Engineering Science and Ocean Engineering
National Taiwan University
Taipei, Taiwan
r10525068@ntu.edu.tw

Yu-Shiang Huang
Data Science Degree Program
National Taiwan University
Taipei, Taiwan
r09946004@ntu.edu.tw

Min-Hung Lo
Electrical Engineering
National Taiwan University
Taipei, Taiwan
r09921134@ntu.edu.tw

Cheng-Hua Liao
Psychology
National Taiwan University
Taipei, Taiwan
b07207063@ntu.edu.tw

ABSTRACT

While training machine learning model, people often have a hard time managing different model version and data; moreover, it's not easy to recall what exactly it is in the messy file system. To address this situation, we present a system that manage both ML model and data. Our system also provides functions to acquire information and metadata.

CCS CONCEPTS

• **Computing methodologies**;

KEYWORDS

Sklearn, Machine learning, Database management system

ACM Reference Format:
Yi-Hsuan Lin, Yu-Shiang Huang, Min-Hung Lo, and Cheng-Hua Liao. 2022. OrpheusDT - Orpheus Decision Tree/Database for Training: A Python Module for Managing Machine Learning Projects. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

With the rapid development of machine learning, a large number of convenient toolkits are flourishing, providing user convenient interface to leverage the machine learning models. For example, Sklearn[1] implements most of statistical learning models and useful training helper function in machine learning field, aggregating them into an unified API format. PyTorch[2] provides modularized class and methods for user to customize deep learning models and the design the parameters updating pipeline. Aside from model implementation, there are also lots of toolkit aim at providing useful functions to make training process more efficient and observable. For instance, Optuna[3] provides high-level API for user to design hyperparameters search experiments and find the optimal solutions with high efficiency. WandB[4] enables user to observe training process and compare different hyperparameter setting through well-designed panels with only few lines of codes insert into original scripts. However, there are data preserving issue with these packages. The saving mechanism in these package often relies on python standard library and binary format, leading to information loss and some difficulties when reusing previous results. Also, it is hard to query information from historical data, which is often seen in database field, when adopting these packages. While there already exists some database architecture with high availability and scalability, we want to build a unified interface to aggregate saving, training and querying functions on the top of existing toolkits. Moreover, we aim at integrating the metadata that had already been preserved by the these toolkits. We turn them into useful functions that benefit the user more from our system than using each toolkit independently.

Our contribution can be summarized as follows:

- We propose orpheusDT, a version control module that manage the model and the dataset in machine learning pipelines, which also supports automatic snapshot and hyperparameter tuning during training.
- Orpheus also utilizes the metadata saved in each training, so that the user can observe the difference between models, list the models sorted by evaluations scores, and make recommendation on model choosing.

2 RELATED WORK

2.1 Sklearn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification,

[1] https://scikit-learn.org/stable/
[2] https://pytorch.org

[3] https://optuna.org
[4] https://wandb.ai

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

regression, clustering and dimensionality reduction via a consistence interface in Python. This library is largely written in Python. While Sklearn provides lots of convenient APIs for user to adopt machine learning models, its official documentation[5] suggest using pickle[6], a python object serialization module for model saving and reloading. The design is easy to use but has lots of drawbacks. First, the module saves files into binary form, which leads to information loss outside Python environment. Also, user may forget the original model setting we reload the model. Furthermore, the pickle module is unsecure[7], unpickling files itself has security concern naturally. These problems constitute the desirability of our project.

2.1.1 Estimator tags. The origin of estimator tags are the tests run on almost every estimator class in Scikit-learn to check if they comply with the basic API conventions. There are annotations of estimators that allow programmatic inspection of their capabilities, for example, sparse matrix support, supported output types, and supported methods. The estimator tags are a dictionary containing these annotations. These tags are used in the common checks performed by the check estimator function and then parametrize with checks decorator. Tags determine which checks to perform and which kind of input data is appropriate. Tags can depend on estimator parameters or even system architecture, which means that in general it can only be determined at runtime.

2.2 Optuna

Optuna[8] is an automatic hyperparameter optimization software framework, particularly designed for machine learning. It features an imperative and define-by-run style user API. Thanks to our define-by-run API, the code written with Optuna enjoys high modularity, and the user of Optuna can dynamically construct the search spaces for the hyperparameters. In the past, users need to design hyperparameters searching process manually. With the help of Optuna, we can easily enable our module the ability to search hyperparameters automatically with high efficiency provided by the parallel computing mechanism in Optuna.

3 ARCHITECTURE

Under OrpheusDT, a database manager is used to connect, import and query to MongoDB. Each task name corresponds to one database. A database consists of two collections, data and metadata collection. An illustration of database architecture is shown in figure 1.

Data collection consists of four keys, data name, X, Y and X columns. X columns are recorded since they can be further used for model explanation. In the case of our demo, it shows the branching of a decision tree.

Meta collection records one training. Thus, it contains data, model and other metadata. We record data by its name, which is already stored in the data collection. For model, we record the model's name and a JSON file that could reconstruct the model.

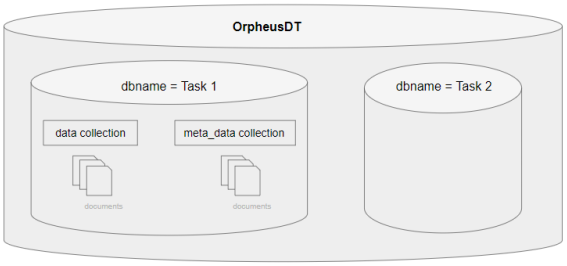

Figure 1: `Database architecture`

For metadata, we record evaluation score, sklearn version, username, training timestamp and estimator tags. The JSON format of metadata collection is shown in Figure 2.

```
metadata_dict = {
    "data_name": data_name,
    "model_name": model_name,
    "model_dict": model_json,
    "meta_dict": {
                    "evaluation_score": score,
                    "scikit_learn_version": sklearn_version,
                    "user": username,
                    "train_timestamp": current_time,
                    "estimator_tags": tags
                  }
}
```

Figure 2: Metadata collection

4 METHOD

In this section, we introduced the mechanism behind our functions. By utilizing several modules, we achieve functionalities of model serialization, overloading, MongoDB connection and hyperparameters tuning.

4.1 Model serialization

Instead of serializing the model into a pickle file, we serialize sklearn model into JSON with the package `sklearn-json` which is available on PyPI. By doing so, we avoid the security issues in which a pickle file might contain malicious code. The string like JSON file reveals more information about the model as well.

4.2 Function overloading

In our system, there are two situations that need to handle different implementations of a function based on the input parameter. One of them is *train* so that three training modes as mentioned in section one are allowed. The second one is the method that saves data into database. After each training session, this method automatically saved the data that is already the class attribute. Also, this function could be called manually by the user with data input.

When it comes to implementation, the concept of overloading simply does not apply to Python. This is because that Python is a late binding object-oriented and dynamic language, which means

[5]https://scikit-learn.org/stable/model_persistence.html
[6]https://docs.python.org/3/library/pickle.html
[7]Declared on its documentation
[8]https://optuna.org

it does not have to declare separate functions for separate types. To overcome this limitation, we apply the package called `multiple dispatch`, which allows function overloading by specifying input parameter in advance.

4.3 MongoDB operation

Pymongo is the most common tool for interacting with MongoDB database from Python. Our functionality span over client connection, insertion, query, deletion. It's worth mentioning that we use `update()` instead of `insert()`, as every data and model naming are unique. For query operation, we utilize `find()` and `aggregation()`. Basically, `find()` is used for querying a single document, while `aggregate()` for multiple documents. For example, extracting data and model are classified into the first category. On the other hand, listing models above certain score and system overview fall into the second category.

4.4 Environment check

The version of sklearn matters. Different version may result in different evaluation result. Furthermore, some attributes are modified that results in unexpected outcome.

To remind the user of this issue, the corresponding sklearn version of the model exported from the database will be compared with that of the current environment. A warning will pop up if the two are not compatible.

4.5 System snapshot

In case of unexpected events happened during training, such as incorrect data pattern or CUDA-out-of-memory, we take a snapshot of the entire database every time before training. By doing so, the user can easily restore the environment right before training.

4.6 Optuna Aggregation

We aggregate this modern hyperparameter searching package into our system. When calling `.train()` method, the method will trigger the corresponding `.fit()` method according to the selected model and do grid search in the often-used hyperparameter range with cross validation. Our implementation is now restricted to decision tree models since it is a proof-of-concept version so far. We leave the design of different hyperparameter search mechanism for different types of models as future work.

4.7 Model audition

As mentioned in section 2.1.1, estimator tags can be look upon as the capabilities and features of a model. Once our system had been used for a while, there may exist a number of models with different usages. Imagining that when we face a new task, we would like to make good use of the old model. Yet, we might have trouble choosing a suitable model to use, and this function is here to help. Based on the knowledge of the new task and its data property, each estimator tag could be set to true or false in a dictionary-like structure. Inputting this dictionary triggers a cross alignment process that evaluate each model. The more tags the model meets the requirement, the more likely the model is what we are looking for. We illustrate the function in Figure 3.

Model audition

```
example_tags = {
    "requires_positive_X": False,
    "X_types": [
        "2darray"
    ],
    "multioutput": True,
    "allow_nan": False,
}

orpheusDT.model_audition(example_tags)
```

data_name	model_name	requires_positive_X	X_types	multioutput	allow_nan	pass_rate
version1	dfc_v1	X	pass	pass	pass	75%
version1	dfc_v2	pass	pass	pass	pass	100%
version1	dfc_v3	pass	pass	pass	pass	100%
version1	dfc_v4	pass	pass	pass	pass	100%
version2	dfc_v1	X	pass	X	pass	50%
version3	dfc_v1	pass	pass	pass	pass	100%
version4	dfc_v1	X	pass	pass	pass	75%

Figure 3: `.model_audition()` method example

4.8 Show difference between models

With multiple data and model version on hand, one of the most common task is to compare different training results. Our module not only supports querying historical training details from the database, but provides `.show_diff()` method for user to compare different model fitting outcome with the latest model in the database. The illustration of this function applied on decision tree model is showed in Figure 4.

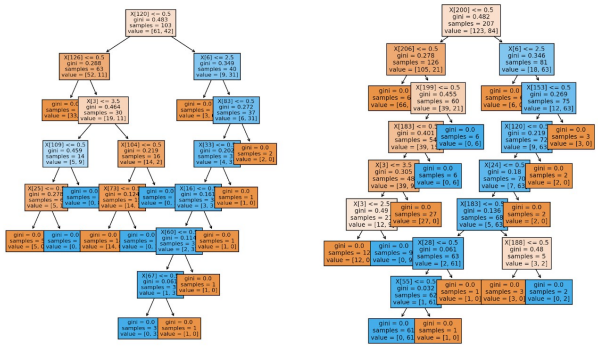

Figure 4: `.show_diff()` method example

5 CONCLUSION

The work OrpheusDT we proposed is an interface that integrates model training, data persistence and information retrieval. Once the data or the model is named and saved to the database, the user could easily reuse them. Hyperparameters are tuned automatically during the training process, which accelerates the develop process. OrpheusDT is designed to be used by a team, we provide system overview that monitors all the activities. The in-built environment check makes sure the training process is compatible. With these features mentioned above, OrpheusDT combined different existed modules and turns them to be more powerful.

6 FUTURE WORK

In our work, our environment check is limited to sklearn version. However, if we want to further extend our work to deep learning, there will be much more compatibility issues. Exporting and importing a container might be another type of metadata saved in

database. Correspondingly, model audition on deep learning would be a great challenge, tests on the capabilities of neural network models need to be done in the backend. Designing the tests requires careful surveys beforehand, which are not covered in our present work. Another important component during training is the learning curve. This could also be saved to the database, and the trend of a curve could be a filter condition utilized in the query language. We left these parts as future work.

REFERENCES

A DISTRIBUTION OF WORK

Name	items
Yi-Hsuan Lin	implement backup() and restore() method
Yu-Shiang Huang	aggregate Optuna; implement .show_diff() method;
Min-Hung Lo	database structure design; implement database insert and query functions
Cheng-Hua Liao	interface design