SoftwareX 23 (2023) 101516

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

DeepMTP: A Python-based deep learning framework for multi-target prediction

Dimitrios Iliadis *, Bernard De Baets, Willem Waegeman

KERMIT, Department of Data Analysis and Mathematical Modeling, Ghent University, Coupure links 653, B-9000 Ghent, Belgium

ARTICLE INFO

Article history:
Received 22 May 2023
Received in revised form 9 August 2023
Accepted 25 August 2023

Keywords:
Multi-target prediction
Multi-label classification
Multivariate regression
Multi-task learning

ABSTRACT

DeepMTP is a python framework designed to be compatible with the majority of machine learning sub-areas that fall under the umbrella of multi-target prediction (MTP). Multi-target prediction includes problem settings like multi-label classification, multivariate regression, multi-task learning, matrix completion, dyadic prediction, and zero-shot learning. Instead of using separate methodologies for the different problem settings, the proposed framework employs a single flexible two-branch neural network architecture that has been proven to be effective across the majority of MTP problem settings. To our knowledge, this is the first attempt at providing a framework that is compatible with more than two MTP problem settings. The source code of the framework is available at https://github.com/diliadis/DeepMTP and an extension with a graphical user-interface is available at https://github.com/diliadis/DeepMTP_gui.

Code metadata

Current code version	0.0.22
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00336
Permanent link to Reproducible Capsule	
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	python
Compilation requirements, operating environments & dependencies	several python packages described in the repository
If available Link to developer documentation/manual	https://deepmtp.readthedocs.io/en/latest/
Support email for questions	dimitrios.iliadis@ugent.be

1. Introduction

Over the last decade the area of machine learning has gained enormous popularity, partially due to the release of software libraries that implement popular models. The WEKA library [1] was one of the first successful attempts of free software for machine learning. It offered tools for data preprocessing and visualization, as well as implementations of several popular at the time machine learning algorithms, in an intuitive graphical user interface (GUI). Almost a decade later, scikit-learn [2] was introduced, offering various classification, regression and clustering algorithms in a flexible Python package. More recently, python-based libraries like Pytorch [3] and Tensorflow [4] used automatic differentiation to aid with the rapid implementation of neural networks that dominated the state-of-the-art in machine learning research.

The aforementioned functionality is usually available for models compliant with standard classification, regression and clustering tasks. A less known task that has wide application potential is Multi-target prediction (MTP). Multi-target prediction is a term used to group several sub-areas of machine learning that have one major commonality, the simultaneous prediction of multiple targets of diverse type. These sub-areas include multi-label classification, multivariate regression, multi-task learning, matrix completion, zero-shot learning and dyadic prediction.

As mentioned in our previous work [5], these subareas developed fairly isolated despite their similarities. This isolation is also reflected at the software level. In terms of libraries, Multi-label classification has enjoyed the most activity, with MULAN

* Corresponding author.
E-mail address: dimitrios.iliadis@ugent.be (Dimitrios Iliadis).

https://doi.org/10.1016/j.softx.2023.101516

being the most popular release. Developed by [6] it offers an API with an extensive range of methods and datasets. Subsequent packages offered additional methods and general functionality [7,8]. For other MTP problem settings like Multi-task learning [9–11], Zero-shot learning and Matrix completion [12–14], the only available options are usually GitHub repositories designed for hyper-specific use cases and input types.

We propose a python package that implements and organizes all of the effort and ideas presented in our previous work [15]. More specifically, the package implements a two-branch neural network that is flexible enough to work with the majority of the MTP problem settings while achieving competitive performance compared to other related packages [15]. This architecture itself is offered with additional functionality, spanning most of the steps of a typical machine learning pipeline. To our knowledge this is the first library that offers out-of-the-box compatibility with more that two MTP problem settings.

2. The basics of multi-target prediction

Multi-target prediction can be seen as an umbrella term that covers all the aforementioned sub-areas of machine learning. Borrowing the basic terminology of our previous work, every MTP problem setting can be characterized by an instance set $\mathcal{X}$, a target set $\mathcal{T}$ and a score set $\mathcal{Y}$. The dataset $\mathcal{D}$ can be decomposed into triplets $(\mathbf{x}_i, \mathbf{t}_j, y_{ij})$ where $\mathbf{x}_i \in \mathcal{X}$ represents an instance, $\mathbf{t}_j \in \mathcal{T}$ represents a target, and $y_{ij} \in \mathcal{Y}$ is the score that summarizes the relationship of the instance $\mathbf{x}_i$-target $\mathbf{t}_j$ pair. The number of instances and targets are finite, so $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. The scores can then be naturally arranged in an $n \times m$ matrix $\mathbf{Y}$ that remains incomplete in the majority of MTP problem settings. To complete the definition of an MTP problem we also have to define the relation of instance and targets between the train set and test set. Even though the basic objective is to predict the score for any unobserved instance-target pair $(\mathbf{x}, \mathbf{t}) \in \mathcal{X} \times \mathcal{T}$, the aforementioned relationship can inform us about the difficulty of a prediction task. Additionally, the existence of features for the instances and/or targets (also called side information) can determine the feasibility of the generalization objective.

3. A two-branch neural network architecture

The two-branch architecture used by DeepMTP was first popularized by [16] in the area of collaborative filtering. In our previous work we showed that the same basic architecture can be used as a successful benchmark for many MTP problem settings. In its simplest form, the network is comprised of two branches that encode the input features of a specific instance and target into two embedding vectors that are then combined. In our previous work [15], the two embeddings were concatenated and then passed to a series of fully-connected layers that terminated into a single output node predicting the interaction score for a given instance-target pair. This approach was introduced by [16] as an improvement over a simpler version that just computes the dot product between the two embeddings, introducing a non-linearity and enabling the modeling of more complex relationships. In contrast to this notion, the subsequent publication of work [17,18] that questions this idea, combined with our preliminary internal testing, led us to offer both versions in the package, and set the dot product as the default option. A more detailed comparison between the two versions is under development and will be published in the near future.

4. The DeepMTP framework

The DeepMTP package offers an implementation of the previously mentioned two-branch neural network, but also provides additional functionality at every major step of a typical machine learning pipeline. Our goal is to provide the user with a package that is simple enough to use across all MTP problem settings and feature rich to enable easy benchmarking. We will provide a summary (Fig. 1) of the functionality at every step below:

4.1. Input formats & dataset processing

At the input step, DeepMTP is designed to support two input formats, reflecting the two main views of the score matrix in a typical MTP dataset. The *matrix view* is the popular choice for settings where the score matrix is fully observed (multi-label classification, multivariate regression) and the default format for popular multi-label classification and multivariate regression data repositories. The second input format is the *triplet view*, which is considered the most flexible and space efficient of the two. This format is common in MTP problem settings where most of the possible instance-target pairs are missing from the dataset (multi-task learning, matrix completion, dyadic prediction). In terms of available side information, DeepMTP supports different input formats to better accommodate various input types. From tabular data in a single CSV file to a collection of images in a specific directory, the library defines rules that users have to follow. The data preprocessing step is considered the most important in the entire pipeline of DeepMTP. In this stage, several characteristics are detected from the score matrix and side information data. The extracted information is essential for the suggestion of the most relevant MTP problem setting, as well as the configuration of the two-branch neural network architecture. By examining the relationship of the instance and target ids between the train and test set, DeepMTP detects the user's generalization objective. At the same time, the existence of side information for instances and/or targets largely determines the feasibility of the detected generalization goal. The recognition of the type of side information guides the selection of an appropriate type of neural network that is used in the corresponding branch. Finally, the detection of the type of values present in the score matrix uncovers the type of prediction task (classification, regression) and determines which loss is used during the training phase (binary cross-entropy loss, mean squared error loss).

4.2. Configure and train–validate–test the model

The configuration of the neural network and of all other aspects that relate to training, validation and testing have to be defined at this step. This is currently achieved by calling the *generate_config* function and by passing all the relevant information as parameters. In this way, the user can override default parameters or even parameters that were detected in the data processing step. The configuration is then used to initialize the neural network and start the train, validation and testing process. Everything related to the implementation of the two-branch architecture as well as several sub-architectures that can be used in the branches is implemented using Pytorch [3]. The design is flexible enough, so that users can introduce their own Pytorch-based models in the two branches. To assist users with the selection of hyperparameters, we decided to automate the step by benchmarking different HPO methods using the two branch architecture across multiple MTP problem settings. The details of this benchmarking will be available in a future publication, but based on this work, we decided to offer Hyperband [19], a bandit-based configuration evaluation approach as a built-in HPO method.

4.3. Additional functionality

Even though the standard machine learning pipeline is comprised of very important steps that help to form a valid

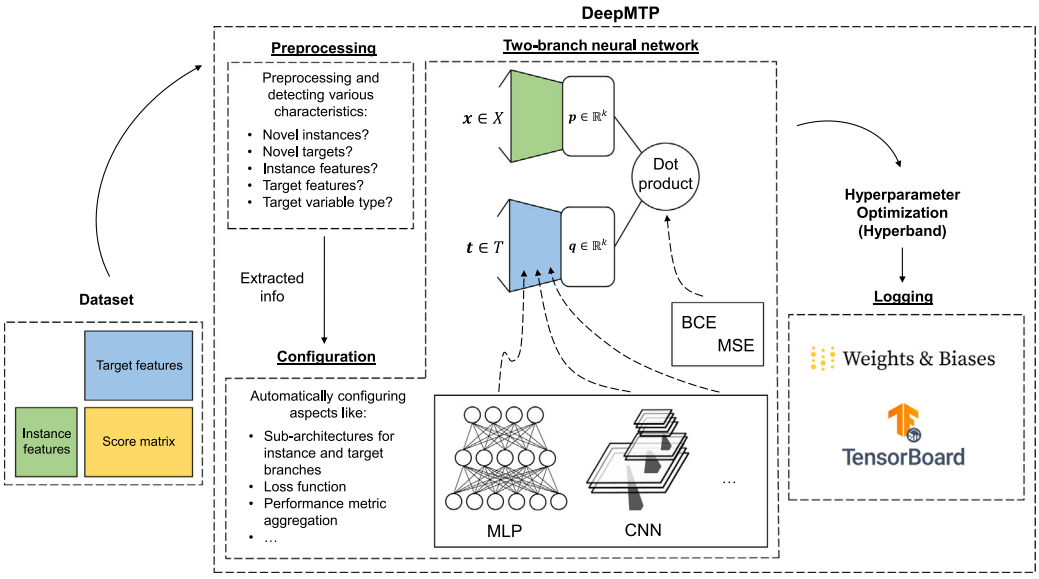

Fig. 1. A summary of the DeepMTP framework.

experiment, additional functionality is needed to aid users with the practical aspects of experimentation. DeepMTP offers three logging options or varying functionality and flexibility (local text files, TensorBoard [4], Weights & Biases [20]). To encourage quick experimentation with many of the MTP problem settings, we also included built-in datasets. The DeepMTP package is designed so that a user can obtain a trained model with only a few lines of code, but in order to maximize the potential audience, we also developed an extension that offers a graphical user interface. The extension is implemented using the streamlit library and can be deployed with minimal effort.

5. Impact and conclusions

The goal of this paper was to present DeepMTP, a python package suitable for MTP problems. Existing packages can support at most two MTP problem settings while DeepMTP is compatible with at least five. This is achieved using a two-branch neural network that is flexible enough to adapt to the intricacies of every MTP problem. The modular design of the architecture also makes the package future-proof as new architectures can be made available in the form of branch options. Various automations are also implemented to use the dataset, extract useful information, and configure the underlying neural network. The package is not limited to the implementation of the neural network, as contrary to existing MTP-related software solutions, it offers additional functionality aimed at helping users with the everyday needs of experimentation (built-in hyperparameter optimization and automated experiment logging). To further attract users from other scientific fields, an extension of the basic package is implemented, offering the option to run experiments without writing code.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The datasets used are already available online.

Acknowledgment

This research received funding from the Flemish Government under the “Onderzoeksprogramma Artificiele Intelligentie (AI) Vlaanderen” programme.

References

[1] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The WEKA data mining software: an update. ACM SIGKDD Explor Newslett 2009;11(1):10–8.
[2] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. Scikit-learn: Machine learning in Python. The Journal of Machine Learning Research (JMLR) 2011;12:2825–30.
[3] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, et al. Pytorch: An imperative style, high-performance deep learning library. Adv Neural Inf Process Syst 2019;32.
[4] Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16). 2016, p. 265–83.
[5] Waegeman W, Dembczyński K, Hüllermeier E. Multi-target prediction: a unifying view on problems and methods. Data Mining Knowledge Discovery (KDD) 2019;33(2):293–324.
[6] Tsoumakas G, Spyromitros-Xioufis E, Vilcek J, Vlahavas I. Mulan: A java library for multi-label learning. The Journal of Machine Learning Research (JMLR) 2011;12:2411–4.
[7] Read J, Reutemann P, Pfahringer B, Holmes G. MEKA: A multi-label/multi-target extension to WEKA. The Journal of Machine Learning Research (JMLR) 2016;17(21):1–5.
[8] Szymański P, Kajdanowicz T. Scikit-multilearn: A python library for multi-label classification. The Journal of Machine Learning Research (JMLR) 2019;20(6):1–22.
[9] Ranjan R, Patel VM, Chellappa R. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 2017;41(1):121–35.
[10] Cao H, Zhou J, Schwarz E. RMTL: an R library for multi-task learning. Bioinformatics 2019;35(10):1797–8.
[11] Rahimi F, Milios EE, Matwin S. MTLV: a library for building deep multi-task learning architectures. In: Proceedings of the 21st ACM symposium on document engineering (DocEng). 2021, p. 1–4.
[12] Hug N. Surprise: a python library for recommender systems. Journal of Open Source Software (JOSS) 2020;5(52):2174.
[13] da Costa A, Fressato E, Neto F, Manzato M, Campello R. Case recommender: a flexible and extensible python framework for recommender systems. In: Proceedings of the 12th ACM conference on recommender systems (RecSys). 2018, p. 494–5.

[14] Radhakrishnan A, Stefanakis G, Belkin M, Uhler C. Simple, fast, and flexible framework for matrix completion with infinite width neural networks. Proceedings of the National Academy of Sciences of the United States of America (PNAS) 2022;119(16):e2115064119.
[15] Iliadis D, De Baets B, Waegeman W. Multi-target prediction for dummies using two-branch neural networks. Mach Learn 2022;111(2):651–84.
[16] He X, Liao L, Zhang H, Nie L, Hu X, Chua T-S. Neural collaborative filtering. In: Proceedings of the 26th international conference on world wide web (WWW). 2017, p. 173–82.
[17] Ferrari Dacrema M, Boglio S, Cremonesi P, Jannach D. A troubling analysis of reproducibility and progress in recommender systems research. ACM Trans Inform Syst (TOIS) 2021;39(2):1–49.
[18] Rendle S, Krichene W, Zhang L, Anderson J. Neural collaborative filtering vs. matrix factorization revisited. In: Fourteenth ACM conference on recommender systems (RecSys). 2020, p. 240–8.
[19] Li L, Jamieson K, DeSalvo G, Rostamizadeh A, Talwalkar A. Hyperband: A novel bandit-based approach to hyperparameter optimization. J Mach Learn Res (JMLR) 2017;18(1):6765–816.
[20] Biewald L. Experiment tracking with weights and biases. 2020, Software available from wandb.com URL https://www.wandb.com/.