SoftwareX 12 (2020) 100551

Contents lists available at ScienceDirect

SoftwareX

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

Original software publication

Deep-water framework: The Swiss army knife of humans working with machine learning models

Rudolf Ferenc [a], Tamás Viszkok [a], Tamás Aladics [a], Judit Jász [a], Péter Hegedűs [b,*]

[a] University of Szeged, Hungary
[b] MTA-SZTE Research Group on Artificial Intelligence, Hungary

ARTICLE INFO

Article history:
Received 6 December 2019
Received in revised form 10 April 2020
Accepted 12 June 2020

Keywords:
Deep-learning
Machine learning
Software analysis
Model management
Data visualization

ABSTRACT

Working with machine learning models has become an everyday task not only for software engineers, but for a much wider spectrum of researchers and professionals. Training such models involves finding the best learning methods and their best hyper-parameters for a specific task, keeping track of the achieved performance measures, comparing the results visually, etc. If we add feature extraction methods – that precede the learning phase and depend on many hyper-parameters themselves – into the mixture, like source code embedding that is quite common in the field of software analysis, the task cries out for supporting tools.

We propose a framework called Deep-Water that works similarly to a configuration management tool in the area of software engineering. It supports defining arbitrary feature extraction and learning methods for an input dataset and helps in executing all the training tasks with different hyper-parameters in a distributed manner. The framework stores all circumstances, parameters and results of training, which can be filtered and visualized later. We successfully used the tool in several software analysis based prediction tasks, like vulnerability or bug prediction, but it is general enough to be applicable in other areas as well, e.g. NLP, image processing, or even other non-IT fields.

Code metadata

Current code version	v1.0.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_368
Code Ocean compute capsule	N/A
Legal Code License	Apache 2.0
Code versioning system used	git
Software code languages, tools, and services used	python, Flask, scikit-learn, Tensorflow
Compilation requirements, operating environments & dependencies	Docker, Python 3.6 (Windows / Linux)
If available Link to developer documentation/manual	https://github.com/sed-inf-u-szeged/DeepWaterFramework/blob/master/README.md
Support email for questions	hpeter@inf.u-szeged.hu

1. Introduction

Machine learning based prediction models (or Artificial Intelligence by its marketing term as adopted by the wide public) play an important role in many of today's software engineering and research tasks. There is an abundance of problems, like image recognition, natural language processing, recommendation systems, just to name a few, which can be effectively solved with machine learning models (e.g. deep learning). Therefore, many software engineering and research tasks nowadays involve steps like finding the best learning methods and their best hyper-parameters for a specific task, keeping track of the parameters used and achieved performance measures, comparing the results

* Corresponding author.
E-mail addresses: ferenc@inf.u-szeged.hu (R. Ferenc), tviszkok@inf.u-szeged.hu (T. Viszkok), aladics@inf.u-szeged.hu (T. Aladics), jasy@inf.u-szeged.hu (J. Jász), hpeter@inf.u-szeged.hu (P. Hegedűs).

https://doi.org/10.1016/j.softx.2020.100551

visually, etc. Feature engineering, be it manual or automatic, adds another dimension to this already complex problem space. This brings in a whole new set of activities and artifacts that software engineers and researchers need to manage in their everyday work.

In this paper, we propose an open-source tool called Deep-Water Framework (DWF)[1] that aims to help researchers and practitioners, who experiment/work with machine learning models. DWF works similarly to a configuration management tool in the area of software engineering. It supports defining arbitrary feature extraction and learning methods for an input dataset and helps run all the training tasks with different hyper-parameters in a distributed manner. Its server component is responsible for scheduling the feature extraction and learning tasks and storing all the parameters and results of the training, while an arbitrary number of worker nodes execute the assigned tasks and report back the results. DWF supports filtering and searching the obtained data and even visualization with customizable dashboards.

We designed the tool to help our software analysis based research work, like predicting vulnerabilities or software bugs based on features extracted from the source code (e.g. source code metrics or source code embeddings). However, its architecture has been designed with extensibility in mind, so the tool should be easy to extend with new feature extraction mechanisms or learning methods and adapt it to other (possibly even non-IT) research fields or the industry.

The benefits of using DWF are:

- It supports scaling the hyper-parameter search of machine learning models by running the different configurations on separate working nodes from which an arbitrary number of instances can be registered.
- It integrates data pre-processing (e.g. standardization or re-sampling) and feature extraction with the model training, therefore, manages a whole experiment in one place.
- It automatically collects and stores all the relevant information during model training (e.g. learning parameters, 10-fold cross-validation results, hardware setup of the worker machine, or training times), which ensures full reproducibility.
- It automatically eliminates unnecessary model training, thus reduces learning times, by searching for already existing configurations in the database.
- It aids result comparison and evaluation through advanced filtering, searching and data visualization features.
- It is suitable for non-technical users[2] as well (e.g., default learning parameters or hyper-parameter search space is suggested by the tool).

Several existing tools [1–6] solve a similar issue as our proposed framework, which clearly shows the significance of the problem domain. However, most of them aid the configuration management of machine learning models at a very technical, low level (for a detailed comparison, see Section 2). We intended to provide a framework that supports researchers and practitioners to get involved in (e.g. software analysis based) machine learning experimentation with minimal effort and self-education, still obtaining professional results in a wide range (though admittedly not all) of standard learning tasks.

2. Related work

There exist a number of similar tools to the one proposed in this paper [1–8]. Some common properties of such tools are that they *(i)* focus mainly on deep learning frameworks, like PyTorch, Tensorflow, Keras; and *(ii)* assume that their users program their own neural networks using these frameworks. Typically, the aim of these tools is to find the best neural network model structure and settings for a given problem.

Some solutions, like DVC [9], focus on version controlling artifacts, provide a public benchmark, and support code sharing to make it easier to adapt a solution in different areas [10], or allow people to re-run others' experiments [11]. However, it is generally true that these tools are best suited for researchers with deep technical knowledge in machine learning, who use deep learning as their primary method for model building. Nonetheless, researchers who are not machine learning experts or not even from the IT field may also want to use machine learning techniques. DWF provides an out-of-the-box solution to apply for common prediction tasks with the option of easy extensibility. Therefore, DWF targets a wider range of possible users than already established tools, who need not know what are the typical hyper-parameter values to search for, have limited technical knowledge in creating their own DNN models, or just want to use standard machine learning algorithms like regression, decision trees, etc.

Another distinction between DWF and the other tools is the way it supports distributed training. Usually, the existing tools that provide such feature at all, achieve it via (payed) services in the cloud (e.g., Amazon or Google). DWF allows the users to build their own worker clusters from existing hardware (utilizing GPUs for instance) in their premises, which might be a better, cheaper, and faster option for many use-cases. Additionally, DWF supports end-to-end model training as it integrates data pre-processing and feature extraction into the whole learning process, which is also a unique feature compared to other solutions.

Table 1 highlights the main properties of the various frameworks mentioned above. The table shows that most of the frameworks share some common features (e.g. result data versioning, graphical dashboards), while there are also some differences in the functionality they provide (e.g. model versioning, ML workflow support). In addition, the licenses of these frameworks also vary, which affects their adoption by a wide range of users.

Therefore, we argue that DWF has its own benefits regarding a set of features, which makes it a viable choice in many scenarios. Most notably, only Azure ML Studio supports training models out-of-the-box, without having to write a single line of code. However, it is a very expensive, heavy-weight proprietary solution (together with Amazon SageMaker) not every user can or is willing to afford. Furthermore, DWF is not meant to compete with such complex proprietary solutions, but instead offer a free and open-source alternative to the widest possible set of users.

From the open-source alternatives, however, DWF provides the easiest, quickest adoption of ML experimentation with no costs and no need to master any ML framework to create a model to train. Additionally, to better support users with limited ML skills and/or having constraints in finding optimal parameters of models, DWF provides pre-defined hyper parameter profiles to automatically search for parameters using grid-search. Despite the fact that most of the frameworks support hyper-parameter optimization in one way or another, most of them still require the users to input the list or range of the parameters they want to test. This is of course something an experienced user might want to apply, but pre-defined profiles help less knowledgeable users to quickly start experimenting.

3. Design of the Deep-Water Framework

3.1. Architecture and communication

The framework relies on a loosely coupled cluster architecture (see Fig. 1). On the one hand, there is a master node (running

[1] https://github.com/sed-inf-u-szeged/DeepWaterFramework.

[2] Can mean IT professionals or researchers with limited machine learning background, but also non-IT people willing to use machine learning models.

Table 1
Comparison of ML frameworks.

Tool	Supported ML frameworks	Coding-less model training	Integration type	Distributed execution	Results dashboard
DWF	Tensorflow, scikit-learn[a]	Yes	Manual	Self-hosted	Yes
Polyaxon	All popular	No	Wrapping	Cloud or self-hosted	Yes
Studio.ml	Any Python ML framework	No	Wrapping	Cloud or self-hosted	Yes
Azure ML Studio	Azure ML	Yes	Independent	Cloud	Yes
Mlflow	Any	No	Manual	Self-hosted[b]	Yes
Sacred	Any Python ML framework	No	Manual	No	3rd party
Comet	All popular	No	Wrapping[c]	Cloud	Yes
Amazon SageMaker	SageMaker SDK[d]	No	Independent	Cloud	Jupyter Notebook
Weights & Biases	Any Python ML framework	No	Manual	Cloud or self-hosted	Yes
DeepDIVA	PyTorch	No	Wrapping	No	3rd party
CodaLab	Any	No	Wrapping	Cloud or self-hosted	Yes
DVC	Any	No	N/A[e]	Cloud	Terminal only
Tool	Data versioning	Model versioning	Workflow support	Hyperparameter search support	License
DWF	Yes	No	Fixed	List, range, pre-defined profiles	Apache 2
Polyaxon	Yes	Yes	Code	Range, search strategies	Apache 2/proprietary
Studio.ml	Yes	No	Code	List	Apache 2
Azure ML Studio	Yes	Yes	Visual	List, range	Proprietary
Mlflow	Yes	Yes	Code	List, range	Apache 2
Sacred	Yes	No	Code	List	MIT
Comet	Yes	Yes	Code	List, range, search strategies	Proprietary
Amazon SageMaker	Yes	Yes	Code	AutoML	Proprietary
Weights & Biases	Yes	No	Code	List, range, search strategies	Proprietary
DeepDIVA	Yes	Yes	Code	Range, SigOpt[f]	LGPL 3
CodaLab	Yes	Yes	Code	List	Apache 2
DVC	Yes	Yes	CLI pipeline	N/A	Apache 2

[a]Models from any Python ML framework can be added, but one needs to extract metrics to be uploaded.
[b]Requires a distributed environment, like Apache Spark, Databricks, etc.
[c]Minimal manual code addition is required in the code.
[d]Possibility to integrate with other frameworks, like Tensorflow.
[e]No real code integration takes place, but the user can configure what produced metric files they want to track.
[f]https://sigopt.com/.

the *DWF-server* module), which is responsible for scheduling the tasks and storing their results to elasticsearch.[3] It also provides a Web-based user interface to supervise these information, while Kibana[4] is used to create customizable dashboards for result data visualization. On the other hand, there are several worker nodes (running the *DWF-client* module). These nodes are working on specific tasks given by their master node. The framework uses a common storage space to share files among workers and the master, which is currently a Samba share, but it could be changed easily to an other solution (e.g., HDFS).

The communication works through a request–response model from the workers to the master node using a REST API. When a worker connects to the master for the first time, it sends its own parameters (e.g. operating system, software version, hardware information, see 3.3) in a HTTP request and receives a unique client ID for later identification. After the first connection, the worker nodes send a ping request every 30 s to the master node, so it knows that the workers are still alive. The master can respond with a task (i.e., a specific feature extraction or learning with appropriate parameters) to these ping requests, which the workers can execute. After a task is completed on a worker node, its result is sent back to the master, which stores it into the database.

The DWF-client executes the tasks using two modules:

- *FeatureAssembler* — a special module that can extract features from the underlying information source (program source code in our cases, but can be natural language texts, images or other), which will form the input data table for the model training. Some typical features widely used in learning tasks with various inputs are word2vec [12] embeddings for natural language text, source code metrics for program code, pixel values for images, etc. Currently, this module supports an AST-based source code embedding algorithm and a simple CSV based file input, but it is designed to be easily extensible with new feature extraction algorithms. To add a new feature assembler, users need to implement the appropriate interfaces and create a plug-in following the tool documentation to assemble a CSV with the desired features for learning.
- *DeepBugHunter* — the machine learning module [13] that can apply many well-known machine learning algorithms to the input data in order to build prediction models. It wraps *scikit-learn*[5] and *Tensorflow* [14] to provide the following learning methods: Naive Bayes; Support Vector Machine; K-nearest Neighbors; Logistic Regression; Linear Regression; Decision Tree; Random Forest; Simple Deep Neural Network; Custom Deep Neural Network; ZeroR.

3.2. Design decisions

The server components (DWF-server, elasticsearch and Kibana) on the manager node are running in a docker stack. This makes it easier to distribute, deploy, and manage all the services. In contrast, workers run on bare metal. The rationale for this is that learning might require GPU access and special CPU features, which would be hard to provide in a virtualized environment. Nonetheless, scaling is still possible by running multiple DWF-clients on the same worker node. The DWF-server is implemented in Python, using the Flask web-framework.[6]

As for the data storage, we selected elasticsearch because it is great at indexing and searching the data. Moreover, elasticsearch

[3] https://www.elastic.co.
[4] https://www.elastic.co/products/kibana.
[5] http://scikit-learn.org/stable/.
[6] https://www.fullstackpython.com/flask.html.

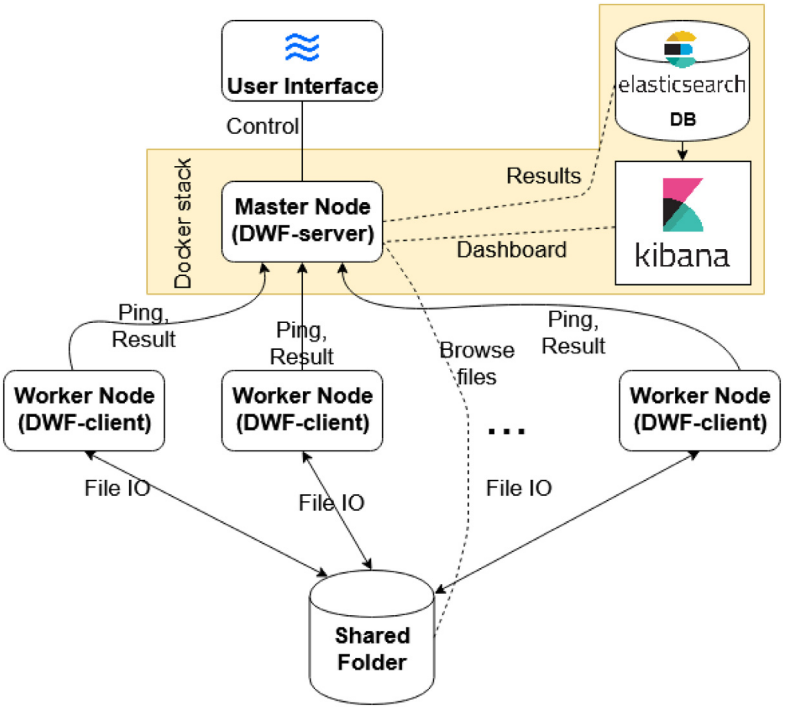


Fig. 1. Architecture of the Deep-Water Framework.

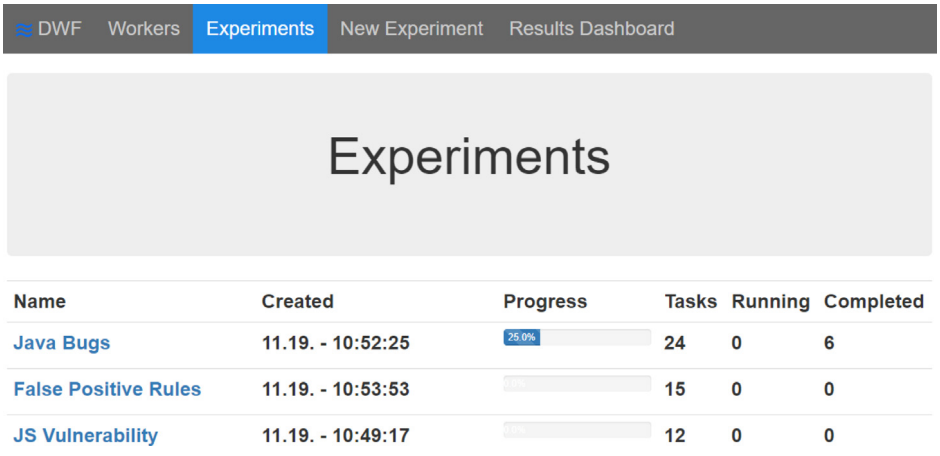


Fig. 2. Experiment list screen.

is a document based NoSQL database, which makes it really convenient to integrate with the existing JSON data we had. Additionally, Kibana is tightly coupled with elasticsearch, which provides a great way to define dashboards on top of the data without the need for programming.

To have a common storage space that we can use to share large data files (csv data files created by feature extractors and used by learning algorithms, saved model files, etc.) among the worker nodes and the master node, we use a Samba share for the time being. However, as a final solution, we plan to set up a distributed file system for this purpose.

3.3. Recorded data

The framework records data that can be categorized into three groups: task related data, architecture related data, and environmental data.

Task related data covers information about the task: input parameters, the used algorithms, status information, and results. Results may vary based on the executed task, but mostly these are performance measures, such as accuracy, precision, recall, F-measure, completeness, confusion matrix, etc.

Architecture related data incorporate information related to the client–server components and their communication: the client (i.e., worker node) hardware information, execution logs, package versions, API endpoints, etc.

Lastly, the environmental data stored by the framework are settings that are essential for the reproducibility of the experiments. These are, for instance, environmental variables, random seeds or timestamps.

4. Usage and showcases of the framework

4.1. Starting and running the framework

To start the server component first time, one needs to build the docker image, then start the whole server stack described in

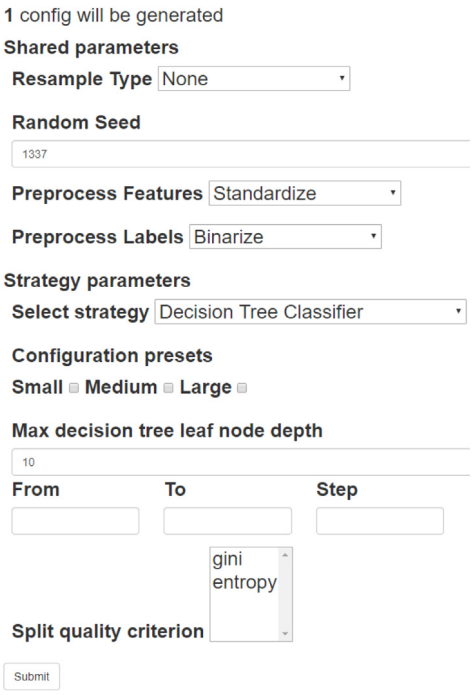
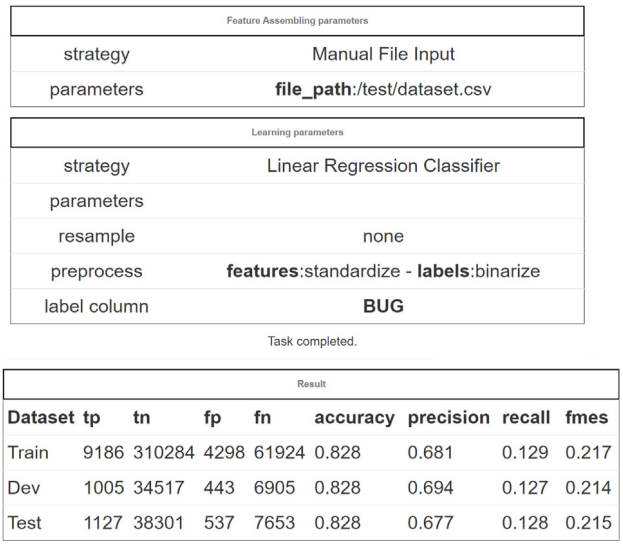


Fig. 3. Add new learning configuration form.



Feature Assembling parameters

strategy	Manual File Input
parameters	**file_path**:/test/dataset.csv

Learning parameters

strategy	Linear Regression Classifier
parameters	
resample	none
preprocess	**features**:standardize - **labels**:binarize
label column	**BUG**

Task completed.

Result

Dataset	tp	tn	fp	fn	accuracy	precision	recall	fmes
Train	9186	310284	4298	61924	0.828	0.681	0.129	0.217
Dev	1005	34517	443	6905	0.828	0.694	0.127	0.214
Test	1127	38301	537	7653	0.828	0.677	0.128	0.215

Fig. 5. Task details screen.

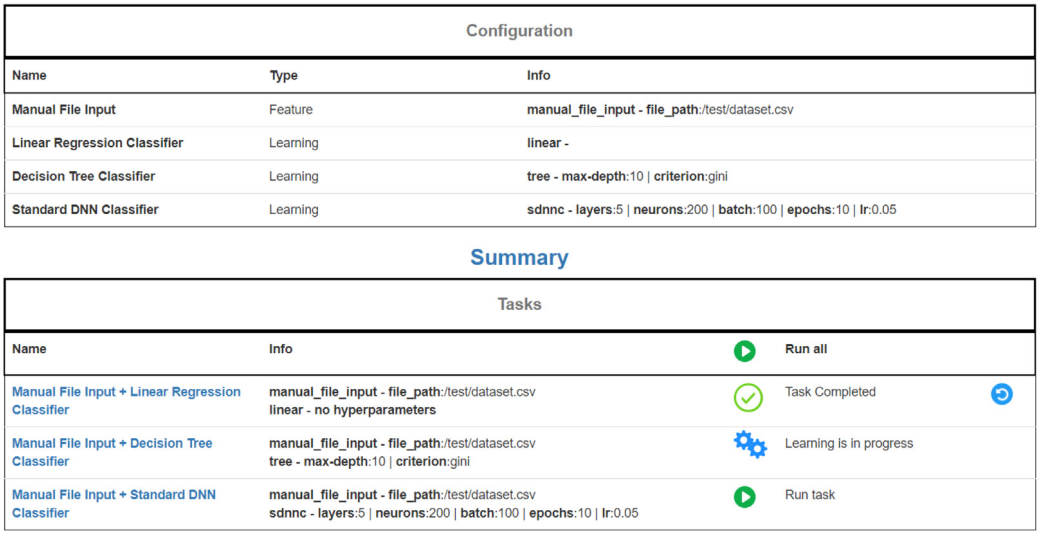
the *docker-compose.yml* file. Starting scripts that execute the necessary sequence of actions in Windows and Linux environments are provided.

To run the client, first the required packages (listed in *requirements.txt*) need to be installed. Then, the related configuration files (*client_params.json, config.json*) should be modified to fit the local environment. Finally, the client can be run by executing the *client.py* Python file. For more technical details, please refer to the README.md file provided in the GitHub repository.

4.2. The Deep-Water Web interface

DWF groups the learning tasks into so-called "experiments". One experiment contains various learning settings belonging to the same research task (e.g., bug prediction of Java classes). The main page of the application (see Fig. 2) lists existing experiments, where the users can track their overall progress (i.e., the number of running and completed tasks). When a user selects an experiment from the list or creates a new one, the framework navigates to the particular experiment's screen (see Fig. 4) that works as a configuration panel until the tasks are generated. In configuration mode, the users can add various feature extraction methods and learning algorithms with specific data pre-processing steps and hyper-parameters to the experiment. Feature extraction algorithms produce the data table for learning methods (e.g. applying source code embedding), which means that the result of the extraction must be a table of numbers in CSV format (i.e. the features of the instances and their class label). The framework allows the users to extend feature extraction algorithms by implementing a simple interface.

Adding a new learning configuration (see Fig. 3) consists of three steps. At the top of the page one can find data preprocessing parameters. Re-sampling upwards means that in case of binary classification, a specified percent of the less populated class' instances are being duplicated, making the distribution of the two classes more even. Re-sampling downwards means



Configuration

Name	Type	Info
Manual File Input	Feature	manual_file_input - file_path:/test/dataset.csv
Linear Regression Classifier	Learning	linear -
Decision Tree Classifier	Learning	tree - max-depth:10 | criterion:gini
Standard DNN Classifier	Learning	sdnnc - layers:5 | neurons:200 | batch:100 | epochs:10 | lr:0.05

Summary

Tasks

Name	Info	Run all	
Manual File Input + Linear Regression Classifier	manual_file_input - file_path:/test/dataset.csv linear - no hyperparameters	Task Completed	
Manual File Input + Decision Tree Classifier	manual_file_input - file_path:/test/dataset.csv tree - max-depth:10 | criterion:gini	Learning is in progress	
Manual File Input + Standard DNN Classifier	manual_file_input - file_path:/test/dataset.csv sdnnc - layers:5 | neurons:200 | batch:100 | epochs:10 | lr:0.05	Run task	

Fig. 4. Experiment configuration screen.

Task results

task	tp	tn	fp	fn	accuracy	precision	recall	fmes
Manual File Input + Linear Regression Classifier	1127	38301	537	7653	82.801%	67.728%	12.836%	21.582%
Manual File Input + Decision Tree Classifier	2527	37070	1768	6253	83.156%	58.836%	28.781%	38.654%
Manual File Input + Standard DNN Classifier	2901.0	37062.0	1776.0	5879.0	83.924%	62.027%	33.041%	43.115%

Fig. 6. Experiment summary screen.

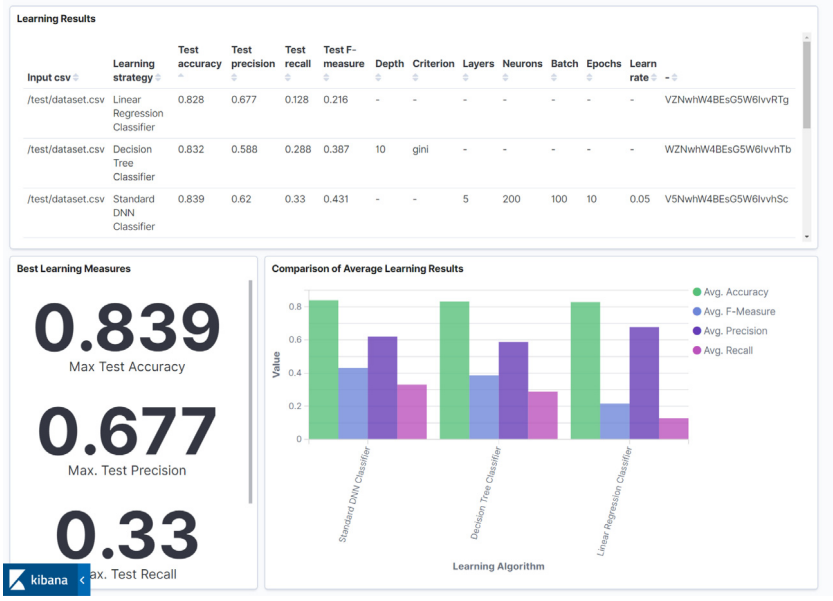


Fig. 7. Kibana dashboard of the results.

the opposite, some of the more populated class' instances are being deleted for the same reason. The user can add an extra pre-processing step to the extracted numeric feature or label columns; the available options are binarize, standardize, and normalize. The second step is the selection of the learning algorithm. After that, the user can define parameters by hand, or can select from predefined configurations (a set of hyper-parameters for the learning algorithm). The numeric values can be added by range, making it easier to generate grid-search for the parameters.

Now that feature extracting and learning configurations are defined, the next step is the generation of tasks. Tasks are simply the combination of exactly one feature extraction and a learning step. Users can add/delete/modify configurations until the executable tasks are generated. After the generation step, all the tasks become runnable and can be executed individually or all at once. The framework then sends the parameters of the tasks to the active worker nodes for execution. The tasks consisting of already executed configurations will not be run unless the user explicitly initiates it (see Fig. 4). Clicking on the name of a task, the user can get details of it (see Fig. 5), like execution progress or task results.

The results of the tasks within one experiment can be compared through the summary view (see Fig. 6). The summary view is a table of results, where one row contains the result of one task. In every column, the background of the best value is green. Clicking on the name of a task navigates the user to the above mentioned task details screen (see Fig. 5).

The "Results Dashboard" menu redirects to the detailed results Kibana dashboard (see Fig. 7). This dashboard is defined by us, but it can be easily modified, extended or replaced using the Kibana web interface.

For further details, see the tutorial[7] contained in the DWF repository, demonstrating how to apply the framework on a public sample data from the UCI Machine Learning Repository [15].

4.3. Showcases of the framework

Showcase 1. *Vulnerability prediction of javascript functions.* We successfully used the framework to find the best data preprocessing method and machine learning algorithm for predicting vulnerable JavaScript functions [16]. We ran an extensive hyperparameter search on a dataset mined from public vulnerability entries with a fixed set of static source code metrics as predictor features. In this case, we fixed the feature extraction mechanism and used a static CSV file as the input for the learning. DWF helped in recording and comparing all the different results to find out that K-nearest Neighbors with a 25% down-sampling

[7] https://github.com/sed-inf-u-szeged/DeepWaterFramework/blob/master/GUIDE.md.

works best in terms of F-measure to predict vulnerable JavaScript functions.

Showcase 2. *Bug prediction of java classes.* Since there is a significant correlation among software metrics and faults in the source code, many bug prediction techniques are based on these metrics. However, there are more and more solutions where other automatic feature extractions (i.e., embeddings) from the source code are investigated. The framework provides an opportunity to compare embeddings produced in different ways with the same learning parameters. In our ongoing research we compare learning efficiency of different embeddings on the unified Java Bug Database [17] using the framework.

5. Conclusions and future work

In this paper, we presented a tool called Deep-Water Framework, which supports researchers and practitioners working with machine learning algorithms and creating models. We have only validated the tool "in-house", but feedback so far is quite positive.

Based on user responses, the main benefits of the tool is the single interface for experiment setup from data pre-processing through feature extraction to model training and hyper-parameter search. Not having to save and search results in excel sheets is also a very big plus, not to mention that the tool does not run the same configuration twice, eliminating unnecessary work. Scalability with a worker node cluster that runs the model training is also very helpful.

The tool received a warm response from the users who already tried it. Despite the fact that the current version of the framework is fully operational, we continue its development and will extend it in many ways. We plan to add a more complex and customizable dashboard to overview learning results, support recommendations about which training algorithm one should use based on the input data characteristics, improve the task scheduling algorithm, extend the available feature extraction methods and learning algorithms, visualize the model training process, support Hadoop for storing shared assets, add new hyper-parameter search strategies, etc.

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.

Acknowledgments

The presented work was carried out within the SETIT Project (2018-1.2.1-NKP-2018-00004)[8] and partially supported by grant TUDFO/47138-1/2019-ITM of the Ministry for Innovation and Technology, Hungary. Furthermore, Péter Hegedűs was supported by the Bolyai János Scholarship of the Hungarian Academy of Sciences and the ÚNKP-19-4-SZTE-20 New National Excellence Program of the Ministry for Innovation and Technology, Hungary.

References

[1] Polyaxon an Open-source Platform for Reproducible Machine Learning at Scale. 2020, https://polyaxon.com website (accessed at 16/03/2020).
[2] Studio.ml. 2020, http://docs.studio.ml website (accessed at 16/03/2020).
[3] Azure Machine Learning Studio. 2020, https://studio.azureml.net website (accessed at 16/03/2020).
[4] MLflow an Open-source Platform to Manage the ML Lifecycle. 2020, https://mlflow.org website (accessed at 16/03/2020).
[5] Greff K, Klein A, Chovanec M, Hutter F, Schmidhuber J. The Sacred Infrastructure for Computational Research. In: Proceedings of the Python in Science Conferences-SciPy Conferences. 2018, http://dx.doi.org/10.6084/m9.figshare.5813412.v1.
[6] Comet a Meta Machine Learning Platform. 2020, https://www.comet.ml website (accessed at 16/03/2020).
[7] Amazon SageMaker Studio. 2020, https://docs.aws.amazon.com/sagemaker/latest/dg/gs-studio.html website (accessed at 16/03/2020).
[8] Alberti M, Pondenkandath V, Würsch M, Ingold R, Liwicki M. DeepDIVA: A Highly-Functional Python Framework for Reproducible Experiments. In: 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR). IEEE; 2018, p. 423–8.
[9] Data Version Control git extension for data scientists, https://dvc.org/.
[10] Weights & Biases Developer Tools for Deep Learning. 2020, https://www.wandb.com website (accessed at 16/03/2020).
[11] CodaLab a Collaborative Platform for Reproducible Research. 2020, http://codalab.org website (accessed at 16/03/2020).
[12] Mikolov T, Chen K, Corrado G, Dean J. Efficient Estimation of Word Representations in Vector Space. 2013, arXiv preprint arXiv:1301.3781.
[13] DeepBugHunter: an Experimental Python Framework for Deep Learning Research, https://github.com/sed-inf-u-szeged/.
[14] Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, et al. Tensorflow: a System for Large-scale Machine Learning. In: OSDI, Vol. 16. 2016, p. 265–83.
[15] UCI Machine Learning Repository. 2020, https://archive.ics.uci.edu/ml/datasets.php website (accessed at 16/03/2020).
[16] Ferenc R, Hegedűs P, Gyimesi P, Antal G, Bán D, Gyimóthy T. Challenging Machine Learning Algorithms in Predicting Vulnerable JavaScript Functions. In: Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. IEEE Press; 2019, p. 8–14.
[17] Ferenc R, Tóth Z, Ladányi G, Siket I, Gyimóthy T. A Public Unified Bug Dataset for Java. In: Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering. ACM; 2018, p. 12–21, doi: https://dl.acm.org/citation.cfm?doid=3273934.3273936.

[8] Project no. 2018-1.2.1-NKP-2018-00004 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.