July 3, 2019

User Test Plan

Version 1.0.0

OpenML

Members
Andrei Danila | 1034559
Bogdan Enache | 1035066
Gergana Goncheva | 1037010
Loïc Alexander Hijl | 1002745
Adrian-Stefan Mares | 0993873
Veselin Minev | 1014541
Thanh-Dat Mathieu Nguyen | 1036672
Antoine Labasse Lutou Nijhuis | 1016657
Claudiu-Teodor Nohai | 1038442
Dragos Mihai Serban | 1033859
Tsvetan Zahariev | 1035269
Sonya Zarkova | 1034611

Project managers
Yuxuan Zhang
Stefan Tanja

Supervisors
Erik Luit
Ion Barosan

Customer
Joaquin Vanschoren

Abstract

This document contains the description of the unit test plan for the Deep Learning (DL) Extension Library, which is part of the project "Sharing deep learning models" developed by the team OpenML Support Squad. The unit tests are made in correspondence to the requirements in the Software Requirements Document (SRD) [1]. This Unit Test Plan (UTP) document complies with the European Space Agency (ESA) [2] software standard.

Contents

Document Status Sheet

Document Title: User Test Plan
Identification: UTP/1.0.0
Version: 1.0.0
Authors: A.Danila, B.Enache, G.Goncheva, L.A.Hijl, A.Mares, V.Minev, T.M.Nguyen, A.L.L.Nijhuis, C.Nohai, D.M.Serban, T.Zahariev, S.Zarkova

Document History

Version	Date	Author(s)	Summary
0.0.1	25-04-2019	T. Zahariev	Created initial document
0.0.2	13-05-2019	L.A. Hijl, T.Zahariev, A.Danila	Setup Layout
0.0.3	21-05-2019	L.A. Hijl & A.L.L. Nijhuis	First draft of Test Plan and Test Procedure
0.0.4	22-05-2019	A.Danilla, B.Enache, S.Zarkova	First draft of Abstract and all sections of the Introduction
0.0.5	30-05-2019	A.Danilla, B.Enache, S.Zarkova	First complete draft
0.1.0	09-06-2019	All authors	Fixed first feedback from supervisor
0.1.1	09-06-2019	All authors	Fixed internal feedback
1.0.0	02-07-2019	All authors	Final edit after approval of supervisor

1 | Introduction

1.1 Purpose

This document contains the unit test plan for the project "Sharing deep learning models", which is an extension for the OpenML platform. More specifically, the test cases are documented, as well as their description, expected result and actual outcome, which will all be supported by diagrams or pictures.
This UTP is written by OpenML Support Squad in accordance to Joaquin Vanschoren's preferences. Any further changes to this document require the full consent of both parties, aforementioned.

1.2 Overview

The remainder of this UTP document contains four additional chapters. First in order is chapter 2 - "Test Plan" with the corresponding Sections: 2.1 containing the test items, Section 2.2 with the features to be tested, Section 2.3 describing the test deliverables, Section 2.4 with the concrete testing tasks, Section 2.5 outlining the environmental needs and lastly Section 2.6 with the definition for the pass / fail criteria of tests.
Chapter 3 - "Test Case Specification" follows, containing a section for each extension with the test cases and a unique test case identifier. Moreover, one can find the items to be tested, the input specification for each test case, the information in regard to the output specifications, and lastly the environmental needs of the tests.
Next is chapter 4 - "Test Procedure" which describes the procedure to run the tests.
The document finishes with chapter 5 - "Test reports", depicting the test runs, the identification of the test procedure, the time when the test was performed and by whom it was performed . For each test case in the procedure, it is determined whether it passes.

1.3 Definitions and Abbreviations

1.3.1 Definitions

DL Extension Library	The product to be created in order to convert the deep learning models and the model transfer specifications (i.e. Open Neural Network eXchange (ONNX) and MLflow) into a format supported by the OpenML platform.
OpenML Python API	"A connector to the collaborative machine learning platform OpenML.org. The OpenML Python package allows to use datasets and tasks from OpenML together with scikit-learn and share the results online" [3].
OpenML flow	The representation of untrained machine learning models in the OpenML platform.

Apache MXNet	"Apache MXNet is an open-source deep learning software framework, used to train, and deploy deep neural networks. It is scalable, allowing for fast model training, and supports multiple programming languages (C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl, and Wolfram Language)" [4].
Deep learning model	A neural network with more than three layers. Deep learning is a subset of machine learning.
Fold	A subset of the training data.
Keras	"Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano" [5].
Neural Network	"Computing systems vaguely inspired by the biological neural networks that constitute animal brains." It is "a framework for many different machine learning algorithms" [6].
ONNX specification	"An open format to represent represent deep learning models" [7]. The representation does not change between different deep learning frameworks.
OpenML	An online machine learning platform for sharing and organizing data, machine learning algorithms and data experiments.
OpenML Support Squad	Name of the development team.
Project "Sharing deep learning models"	The project on which the OpenML Support Squad is working on.
PyTorch	"An open source deep learning platform that provides a seamless path from research prototyping to product deployment" [8].
Sklearn	A Python library that provides machine learning algorithms.
Travis Continuous Integration	"Travis CI is a hosted continuous integration service used to build and test software projects hosted at GitHub" [9].

1.3.2 Abbreviations

API	Application Programming Interface.
DL	Deep Learning.
ESA	European Space Agency.
MXNet	Apache MXNet.
ONNX	Open Neural Network eXchange.
SRD	Software Requirements Document.
URD	User Requirements Document.
UTP	Unit Test Plan.

1.4 List of references

[1] OpenML Support Squad. *DL Extension Library, SRD Software Requirement Document version 1.0.0.*

[2] Software Standardisation and Control. *ESA software engineering standards. 1991.*

[3] OpenML. *OpenML Documentation.* URL: `https://docs.openml.org/` (visited on 09/05/2019).

[4] Wikipedia. *Apache MXNet.* URL: `https://en.wikipedia.org/wiki/Apache_MXNet` (visited on 04/26/2019).

[5] Keras. *Keras homepage.* URL: `https://keras.io/` (visited on 09/05/2019).

[6] Wikipedia. *Artificial Neural Network.* URL: `https://en.wikipedia.org/wiki/Artificial_neural_network` (visited on 04/30/2019).

[7] ONNX. *ONNX homepage.* URL: `https://onnx.ai/` (visited on 04/26/2019).

[8] PyTorch. *PyTorch homepage.* URL: `https://pytorch.org/` (visited on 09/05/2019).

[9] *Travis CI.* URL: `https://en.wikipedia.org/wiki/Travis_CI` (visited on 05/20/2019).

2 | Test plan

2.1 Test items

The software to be tested is the DL Extension Library. It expands on the already existing OpenML Python API to accommodate for Deep Learning models. Testing of the DL Extension Library entails testing whether the desired OpenML procedures on models for all Deep learning libraries covered by the DL Extension Library are functioning correctly.

2.2 Features to be tested

The features to be tested consist of all implemented features of the DL Extension Library. This implies passing of the units tests for serialization, deserialization and running on a fold of the models built on the following Deep Learning libraries: Keras, PyTorch, ONNX, MXNet. All of which are described by the "*must have* functional requirements" chapter 3 of the SRD. Additionally, visualization is implemented in a completely separate module from the DL Extension Library and is accounted for during acceptance testing.

The DL Extension Library is part of the OpenML Python API that will be imported by the users in their Python project. The OpenML Python API handles all interactions with the OpenML platform. Whenever necessary, the OpenML Python API calls the DL Extension Library in order to convert the deep learning model to an OpenML flow, the format supported by the OpenML platform. The OpenML Support Squad is not responsible for the development and maintenance of the OpenML Python API. The OpenML Support Squad has not modified the OpenML Python API since it is outside of the scope of the project "Sharing deep learning models".

The contribution of OpenML Support Squad consists of two parts: creating the DL Extension Library and the visualization module.

The DL Extension Library was built to provide support for the following deep learning libraries and specifications on the OpenML platform: Keras, PyTorch, Apache MXNet (MXNet) and ONNX. These libraries have already been created by their respective developers and, as such, are not part of this project and are not under the responsibility of OpenML Support Squad. Because of this, the OpenML Support Squad will not create unit tests for the functions that belong to these libraries.

The visualization module was built to serve as a prototype which can be integrated into the website. It allows users to visualize their models and model results.

2.3 Test deliverables

Mandatory deliverables before testing:

- UTP sections 1 through 4
- Unit test input

- Finalized version of the DL Extension Library

Mandatory deliverables after testing has concluded:

- UTP section 5
- Unit test output data
- Problem reports

2.4 Testing tasks

Tasks to be tested:

- Designing of unit tests for implemented features to be tested
- Defining and instantiating input data for the unit tests
- Ensuring that environmental needs are met for unit test

2.5 Environmental needs

To perform unit testing on the DL Extension Library, the following environmental requirements are required:

- Built-in Unit testing framework of python, inspired by JUnit testing
- A system capable of utilizing the DL Extension Library effectively requires at least the following specifications:

CPU	$\geq$ 2.0 GHz x64 or equivalent
RAM	$\geq$ 8GB
Disk space	$\geq$ 10GB
Operating system	Windows 7 or later, macOS, and Linux

- Python version 3.5

- Internet connection needed for downloading and uploading OpenML flows as well as publishing results on

2.6 Test case pass/fail criteria

In the next chapter, we describe the individual test cases. Their pass conditions are described in the corresponding contracts (expected outcome). If the output of a unit test differs from the contract, it fails. The unit test plan as a whole passes only if every individual unit test passes.

3 | Test case specification

3.1 Test structure

Test case identifier: The name and number of the test
Description: A brief description of the test
Test item: The function/code to be tested
Input: Input parameters used for the test
Output: Expected result of each function
Environmental needs: Special needs for this test

3.2 Keras Extension tests

All the tests from this section are specific to the OpenML Keras extension.

UTK - 01
The test consists of whether `can_handle_model()` distinguishes between a Keras model and models made in other frameworks.

Test item:	`can_handle_model()`
Input:	a Keras model and/or other models which are not Keras
Output:	True if Keras model, False otherwise
Environmental needs:	-

UTK - 02
Checks whether the version of required libraries by Keras are returned in the form of a list.

Test item:	`get_version_information()`
Input:	-
Output:	a non-empty list of versions of required libraries by Keras
Environmental needs:	-

UTK - 03
Checks whether the returned value of `create_setup_string()` is a string.

Test item:	`create_setup_string()`
Input:	a Keras model
Output:	a string with the versions of the libraries of Keras and the name of the current project
Environmental needs:	`can_handle_model(model)` = True

UTK - 04

The test consists of whether `is_estimator()` distinguishes between a Keras model and models made in other frameworks.

Test item:	`is_estimator()`
Input:	a Keras model and/or other models which are not Keras
Output:	True if Keras model, False otherwise
Environmental needs:	-

UTK - 05

The test consists of whether `_is_keras_flow()` distinguishes between a flow that represents a Keras model and flows representing other frameworks.

Test item:	`_is_keras_flow()`
Input:	a OpenML flow
Output:	True if Keras flow, False otherwise
Environmental needs:	`can_handle_model(model)` = True

UTK - 06

The test checks whether the current version of Keras installed on the machine fulfills the dependencies required by the deserialization of the Keras flow.

Test item:	`_check_dependencies()`
Input:	a string representing the required dependencies, some of which may not be satisfied
Output:	a `ValueError` if a required dependency is not satisfied, nothing otherwise
Environmental needs:	-

UTK - 07

Checks if the given paramateres do not correspond with the given flow.

Test item:	`_openml_param_name_to_keras()`
Input:	OpenML parameters and flows representing Keras models, some of which are not pair-wise correlated, i.e. parameter does not correspond to flow
Output:	a string containing the name of the flow and of the parameter or error if the parameter does not correspond to the flow
Environmental needs:	`can_handle_model(model)` = True

UTK - 08

Checks whether the sequential Keras model is serialized into a flow. The test consists of verifying whether the obtained flow and the original model have the same: name, class name, description, parameters, dependencies, structure.

Test item:	`model_to_flow()`
Input:	a Keras model
Output:	a flow representing a Keras model
Environmental needs:	`can_handle_model(model)` = True

UTK - 09

Checks whether the functional Keras model is serialized into a flow. The test consists of verifying whether the obtained flow and the original model have the same: name, class name, description, parameters, dependencies, structure.

Test item:	`model_to_flow()`
Input:	a Keras model
Output:	a flow representing a Keras model
Environmental needs:	`can_handle_model(model)` = True

UTK - 10

Checks whether the sequential Keras flow is deserialized into a model. The test consists of verifying whether the obtained configuration of the model from the flow is the same as the original model.

Test item:	`flow_to_model()`
Input:	a flow representing a Keras model
Output:	a Keras model
Environmental needs:	`can_handle_model(model)` = True

UTK - 11

Checks whether the functional Keras flow is deserialized into a model. The test consists of verifying whether the obtained configuration of the model from the flow is the same as the original model.

Test item:	`flow_to_model()`
Input:	a flow representing a Keras model
Output:	a Keras model
Environmental needs:	`can_handle_model(model)` = True

UTK - 12

Checks whether the model created from the given parameters is the same as the original one (the one from which the given parameters were retrieved).

Test item:	`_from_parameters()`
Input:	the parameters of an existing Keras model
Output:	a Keras model constructed from the received parameters of an existing Keras model
Environmental needs:	-

UTK - 13

Checks if the compilation of the Keras model is kept after deserialization.

Test item:	`model_to_flow()` followed by `flow_to_model()`
Input:	a Keras model
Output:	the same Keras model
Environmental needs:	`can_handle_model(model)` = True

UTK - 14

Checks whether the results of running several classification tasks on a sequential Keras model are similar in shape and approximate value to the the expected results of the tasks.

Test item:	`run_model_on_fold()`
Input:	a sequential Keras model and several classification tasks
Output:	results of running the tasks on the model
Environmental needs:	`can_handle_model(model)` = True

UTK - 15

Checks whether the results of running several classification tasks on a functional Keras model are similar in shape and approximate value to the the expected results of the tasks.

Test item:	`run_model_on_fold()`
Input:	a functional Keras model and several classification tasks
Output:	results of running the tasks on the model
Environmental needs:	`can_handle_model(model)` = True

UTK - 16

Checks whether the results of running several regression tasks on a sequential Keras model are similar in shape and approximate value to the the expected results of the tasks.

Test item:	`run_model_on_fold()`
Input:	a sequential Keras model and several regression tasks
Output:	results of running the tasks on the model
Environmental needs:	`can_handle_model(model)` = True

UTK - 17

Checks whether the results of running several regression tasks on a functional Keras model are similar in shape and approximate value to the the expected results of the tasks.

Test item:	`run_model_on_fold()`
Input:	a functional Keras model and several regression tasks
Output:	results of running the tasks on the model
Environmental needs:	`can_handle_model(model)` = True

3.3 PyTorch Extension tests

All the tests from this section are specific to the OpenML PyTorch extension.

UTP - 01

The test consists of whether `can_handle_model()` distinguishes between a PyTorch model and models created in other frameworks.

Test item:	`can_handle_model()`
Input:	a PyTorch model and/or other models which are not PyTorch
Output:	True if PyTorch model, False otherwise
Environmental needs:	-

UTP - 02

Checks whether the version of required libraries by Pytorch are returned in the form of a list.

Test item:	`get_version_information()`
Input:	-
Output:	a non-empty list of versions of required libraries by Pytorch
Environmental needs:	-

UTP - 03

Checks whether the returned value of `create_setup_string()` is a string.

Test item:	`create_setup_string()`
Input:	a Pytorch model
Output:	a string with the versions of the libraries of Pytorch and the name of the current project
Environmental needs:	`can_handle_model(model)` = True

UTP - 04

The test consists of whether `is_estimator()` distinguishes between a PyTorch model and models made in other frameworks.

Test item:	`is_estimator()`
Input:	a Pytorch model and/or other models which are not PyTorch
Output:	True if PyTorch model, False otherwise
Environmental needs:	-

UTP - 05

The test consists of whether `_is_pytorch_flow()` distinguishes between a flow that represents a PyTorch model and flows representing other frameworks.

Test item:	`_is_pytorch_flow()`
Input:	a OpenML flow PyTorch
Output:	True if PyTorch flow, False otherwise
Environmental needs:	`can_handle_model(model)` = True

UTP - 06

The test checks whether the current version of Pytorch installed on the machine fulfills the dependencies required by the deserialization of the Pytorch flow.

Test item:	`_check_dependencies()`
Input:	a string representing the required dependencies, some of which may not be satisfied
Output:	a `ValueError` if a required dependency is not satisfied, nothing otherwise
Environmental needs:	-

UTP - 07

Checks if the given paramateres do not correspond with the given flow.

Test item:	`_openml_param_name_to_torch()`
Input:	OpenML parameters and flows representing Pytorch models, some of which are not pair-wise correlated, i.e. parameter does not correspond to flow
Output:	a string containing the name of the flow and of the parameter or error if the parameter does not correspond to the flow
Environmental needs:	`can_handle_model(model)` = True

UTP - 08

Checks whether the sequential Pytorch model is serialized into a flow. The test consists of verifying whether the obtained flow and the original model have the same: name, class name, description, parameters, dependencies, structure.

Test item:	`model_to_flow()`
Input:	a Pytorch model
Output:	a flow representing a Pytorch model
Environmental needs:	`can_handle_model(model)` = True

UTP - 09

Checks whether the sequential Pytorch flow is deserialized into a model. The test consists of verifying whether the obtained configuration of the model from the flow is the same as the original model.

Test item:	`flow_to_model()`
Input:	a flow representing a Pytorch model
Output:	a Pytorch model
Environmental needs:	`can_handle_model(model)` = True

UTP - 10

Checks whether the sequential nested Pytorch flow is deserialized into a model. The test consists of verifying whether the obtained configuration of the model from the flow is the same as the original model.

Test item:	`flow_to_model()`
Input:	a flow representing a sequential nested Pytorch model
Output:	a Pytorch model
Environmental needs:	`can_handle_model(model)` = True

UTP - 11

Checks whether the results of running several classification tasks on a sequential Pytorch model are similar in shape and approximate value to the the expected results of the tasks.

Test item:	`run_model_on_fold()`
Input:	a sequential Pytorch model and several classification tasks
Output:	results of running the tasks on the model
Environmental needs:	`can_handle_model(model)` = True

UTP - 12

Checks whether the results of running several regression tasks on a sequential Pytorch model are similar in shape and approximate value to the the expected results of the tasks.

Test item:	`run_model_on_fold()`
Input:	a sequential Pytorch model and several regression tasks
Output:	results of running the tasks on the model
Environmental needs:	`can_handle_model(model)` = True

3.4 MXNet Extension tests

All the tests from this section are specific to the OpenML MXNet extension.

UTM - 01

The test consists of whether `can_handle_model()` distinguishes between a MXNet model and models made in other frameworks.

Test item:	`can_handle_model()`
Input:	a MXNet model and/or other models which are not MXNet
Output:	True if MXNet model, False otherwise
Environmental needs:	-

UTM - 02

Checks whether the version of required libraries by MXNet are returned in the form of a list.

Test item:	`get_version_information()`
Input:	-
Output:	a non-empty list of versions of required libraries by Keras
Environmental needs:	-

UTM - 03

Checks whether the returned value of `create_setup_string()` is a string.

Test item:	`create_setup_string()`
Input:	a MXNet model
Output:	a string with the versions of the libraries of MXNet and the name of the current project
Environmental needs:	`can_handle_model(model)` = True

UTM - 04

The test consists of whether `is_estimator()` distinguishes between a MXNet model and models made in other frameworks.

Test item:	`is_estimator()`
Input:	a MXNet model and/or other models which are not MXNet
Output:	True if MXNet model, False otherwise
Environmental needs:	-

UTM - 05

The test consists of whether `_is_mxnet_flow()` distinguishes between a flow that represents a MXNet model and flows representing other frameworks.

Test item:	`_is_mxent_flow()`
Input:	a OpenML flow MXNet
Output:	True if MXNet flow, False otherwise
Environmental needs:	`can_handle_model(model)` = True

UTM - 06

The test checks whether the current version of MXNet installed on the machine fulfills the dependencies required by the deserialization of the MXNet flow.

Test item:	`_check_dependencies()`
Input:	a string representing the required dependencies, some of which may not be satisfied
Output:	a `ValueError` if a required dependency is not satisfied, nothing otherwise
Environmental needs:	-

UTM - 07

Checks whether the sequential MXNet model is serialized into a flow. The test consists of verifying whether the obtained flow and the original model have the same: name, class name, description, parameters, dependencies, structure.

Test item:	`model_to_flow()`
Input:	a MXNet model
Output:	a flow representing a MXNet model
Environmental needs:	`can_handle_model(model)` = True

UTM - 08

Checks whether the sequential MXNet flow is deserialized into a model. The test consists of verifying whether the obtained configuration of the model from the flow is the same as the original model.

Test item:	`flow_to_model()`
Input:	a flow representing a MXNet model
Output:	a MXNet model
Environmental needs:	`can_handle_model(model)` = True

UTM - 09

Checks whether the results of running several classification tasks on a sequential MXNet model are similar in shape and approximate value to the the expected results of the tasks.

Test item:	`run_model_on_fold()`
Input:	a sequential MXNet model and several classification tasks
Output:	results of running the tasks on the model
Environmental needs:	`can_handle_model(model)` = True

UTM - 10

Checks whether the results of running several regression tasks on a sequential MXNet model are similar in shape and approximate value to the the expected results of the tasks.

Test item:	`run_model_on_fold()`
Input:	a sequential MXNet model and several regression tasks
Output:	results of running the tasks on the model
Environmental needs:	`can_handle_model(model)` = True

3.5 ONNX Extension tests

All the tests from this section are specific for to the OpenML ONNX extension.

UTO - 01

The test consists of whether `can_handle_model()` distinguishes between an ONNX specification and models made in other frameworks.

Test item:	`can_handle_model()`
Input:	an ONNX specification and/or other models which are not ONNX
Output:	True if ONNX specification, False otherwise
Environmental needs:	-

UTO - 02

Checks whether the version of required libraries by ONNX are returned in the form of a list.

Test item:	`get_version_information()`
Input:	-
Output:	a non-empty list of versions of required libraries by ONNX
Environmental needs:	-

UTO - 03

Checks whether the returned value of `create_setup_string()` is a string.

Test item:	`create_setup_string()`
Input:	an ONNX specification
Output:	a string with the versions of the libraries of ONNX and the name of the current project
Environmental needs:	`can_handle_model(model)` = True

UTO - 04

The test consists of whether `is_estimator()` distinguishes between an ONNX specification and models made in other frameworks.

Test item:	`is_estimator()`
Input:	an ONNX specification and/or other models which are not ONNX
Output:	True if ONNX specification, False otherwise
Environmental needs:	-

UTO - 05

The test consists of whether `_is_onnx_flow()` distinguishes between a flow that represents an ONNX specification and flows representing other frameworks.

Test item:	`_is_onnx_flow()`
Input:	an OpenML flow ONNX
Output:	True if ONNX flow, False otherwise
Environmental needs:	`can_handle_model(model)` = True

UTO - 06

Checks whether the ONNX specification is serialized into a flow. The test consists of verifying whether the obtained flow and the original specification have the same: name, class name, description, parameters, dependencies, structure. For this test we have used an ONNX specification representing an Apache MXNet, because it is the only which is fully supported.

Test item:	`model_to_flow()`
Input:	an ONNX specification
Output:	a flow representing the ONNX specification
Environmental needs:	`can_handle_model(model)` = True

UTO - 07

Checks whether the ONNX flow is deserialized into a specification. The test consists of verifying whether the obtained specification from the flow is the same as the original ONNX specification.

Test item:	`flow_to_model()`
Input:	a flow representing an ONNX specification
Output:	an ONNX specification
Environmental needs:	`can_handle_model(model)` = True

UTO - 08

Checks whether the results of running several classification tasks on an MXNet model, that represents the ONNX specification, are similar in shape and approximate value to the the expected results of the tasks.

Test item:	`run_model_on_fold()`
Input:	an ONNX specification and several classification tasks
Output:	results of running the tasks
Environmental needs:	`can_handle_model(model)` = True

UTO - 09

Checks whether the results of running several regression tasks on an MXNet model, that represents the ONNX specification, are similar in shape and approximate value to the the expected results of the tasks.

Test item:	`run_model_on_fold()`
Input:	an ONNX specification and several regression tasks
Output:	results of running the tasks
Environmental needs:	`can_handle_model(model)` = True

3.6 Visualization tests

All the tests from this section are specific to the visualization module.

UTV - 01

Checks whether the text objects responsible for the flow information are visible or invisible based on the existence of errors and flow loading status.

Test item:	`update_flow_info_texts_visibility()`
Input:	flow compatible with visualizer
Output:	visibility status of flow info objects
Environmental needs:	-

UTV - 02

Checks whether the correct text is displayed when the flow or graph information is changed.

Test item:	`update_flow_graph_text()`
Input:	flow compatible with visualizer
Output:	loading text displayed
Environmental needs:	-

UTV - 03

Checks whether information is passed correctly during loading. Output parameters should be the same as input parameters.

Test item:	`init_flow_loading()`
Input:	clicks to update the visualizer, flow compatible with visualizer
Output:	True iff input = output, False otherwise
Environmental needs:	-

UTV - 04

Checks whether flow loading information is updated correctly based on the presence of a flow id and load button clicks.

Test item:	`update_flow_loading_info()`
Input:	clicks to update the visualizer, flow compatible with visualizer
Output:	information loaded iff flow id present after before click, nothing loaded otherwise
Environmental needs:	-

UTV - 05

Checks whether valid flows can be loaded while invalid inputs return errors.

Test item:	`load_flow()`
Input:	compatible flows, incompatible inputs or flows of unsupported libraries, such as Sklearn flow ids
Output:	data loaded iff valid flow, error otherwise
Environmental needs:	-

UTV - 06

Checks whether error messages are returned in case of erroneous flow data input.

Test item:	`update_flow_error_text()`
Input:	valid flow data, invalid flow data
Output:	no errors iff valid flow data, error otherwise
Environmental needs:	-

UTV - 07

Checks whether the graph is visible based on the correctness of flow data and click updates.

Test item:	`update_flow_graph_visibility()`
Input:	flow data, clicks to update visualizer
Output:	visible iff correct flow data is loaded and load button was clicked, invisible otherwise
Environmental needs:	-

UTV - 08

Checks whether the HTML component hosting the graph is correctly created.

Test item:	`update_flow_graph()`
Input:	flow data, clicks to update visualizer
Output:	iframe with graph iff correct flow data, error otherwise
Environmental needs:	-

UTV - 09

Checks whether the text objects responsible for the run information are visible or invisible based on the existence of errors and run loading status.

Test item:	`update_run_info_texts_visibility()`
Input:	run compatible with visualizer
Output:	visibility status of run info objects
Environmental needs:	-

UTV - 10

Checks whether the correct text is displayed when the run or metric information is changed.

Test item:	`update_run_graph_text()`
Input:	run compatible with visualizer
Output:	loading text displayed
Environmental needs:	-

UTV - 11

Checks whether information is passed correctly during loading. Output parameters should be the same as input parameters.

Test item:	`init_run_loading()`
Input:	clicks to update the visualizer, run compatible with visualizer
Output:	True iff input = output, False otherwise
Environmental needs:	-

UTV - 12

Checks whether run loading information is updated correctly based on the presence of a run id and load button clicks.

Test item:	`update_run_loading_info()`
Input:	clicks to update the visualizer, run compatible with visualizer
Output:	information loaded iff run id present after before click, nothing loaded otherwise
Environmental needs:	-

UTV - 13

Checks whether valid runs can be loaded while invalid inputs return errors.

Test item:	`load_run()`
Input:	compatible runs, incompatible inputs or runs of unsupported libraries, such as Sklearn run ids
Output:	data loaded and populated drop down menus iff valid run, error and empty drop down menus otherwise
Environmental needs:	-

UTV - 14

Checks whether error messages are returned in case of erroneous run data input.

Test item:	`update_run_error_text()`
Input:	valid run data, invalid run data
Output:	no errors iff valid run data, error otherwise
Environmental needs:	-

UTV - 15

Checks whether the metric graphs are visible based on the correctness of run data and click updates.

Test item:	`update_run_graph_visibility()`
Input:	run data, clicks to update visualizer
Output:	visible and populated metrics drop-down menu iff correct run data is loaded and load button was clicked, invisible and empty metric drop-down menu otherwise
Environmental needs:	-

UTV - 16

Checks whether the HTML components/figures hosting the metric visualizations are correctly created.

Test item:	`update_run_graph()`
Input:	run data, clicks to update visualizer
Output:	components/figures with metrics iff correct run data, empty components and error otherwise
Environmental needs:	-

UTV - 17

Checks whether the system can discern between error or loading states.

Test item:	`has_error_or_is_loading()`
Input:	clicks to update visualizer, correct data, incorrect data
Output:	error iff incorrect number of clicks or incorrect data, loading iff correct data loading
Environmental needs:	-

UTV - 18

Checks whether the loading text and style are visible or invisible during data loading or updating.

Test item:	`get_info_text_style()`
Input:	correct data, clicks to update visualizer
Output:	error text invisible iff error present, loading text visible iff data loading
Environmental needs:	-

UTV - 19

Checks whether loading starts once run/flow id present in id box and button clicked.

Test item:	`get_loading_info()`
Input:	compatible run/flow, clicks to update visualizer
Output:	no loading iff no id present, loading iff id present and click
Environmental needs:	-

UTV - 20

Checks whether the system returns the correct error message in case of error.

Test item:	`get_error_text()`
Input:	correct data, incorrect data
Output:	error string empty iff no error, same string as error string from data loading otherwise
Environmental needs:	-

UTV - 21

Checks whether the information text is correctly displayed in the different states of loading.

Test item:	`get_visibility_style()`
Input:	correct data, incorrect data, clicks to update visualizer
Output:	text invisible iff error present, no data loaded or data still loading, visible otherwise
Environmental needs:	-

UTV - 22

Checks whether the loading text and style are visible or invisible during data loading or updating.

Test item:	`get_info_text_style()`
Input:	correct data, clicks to update visualizer
Output:	error text invisible iff error present, loading text visible iff data loading
Environmental needs:	-

UTV - 23

Checks whether the system can create a figure from predefined data.

Test item:	`create_figure()`
Input:	predefined data
Output:	dictionary filled with data and layout values
Environmental needs:	-

UTV - 24

Checks whether the system can create a graph from predefined data.

Test item:	`extract_run_graph_data()`
Input:	predefined data
Output:	correctly filled value lists, lists filled with instances of go.Scatter
Environmental needs:	-

UTV - 25

Checks whether the system can correctly concatenate lists.

Test item:	`add_lists_element_wise()`
Input:	2 lists filled with data elements
Output:	a list with all elements from origin lists
Environmental needs:	-

UTV - 26

Checks whether the system can retrieve training data.

Test item:	`get_training_data()`
Input:	training data URL and task id
Output:	retrieved .csv file if data and id exist, error otherwise
Environmental needs:	-

UTV - 27

Checks whether the system can get the ONNX model protobuf from extension.

Test item:	`get_onnx_model()`
Input:	run/flow id with ONNX model attached to flow
Output:	ONNX model protobuf corresponding to origin library
Environmental needs:	-

---

4 | Test procedure

This chapter describes the procedure that should be followed to correctly run the unit tests. The unit tests which the OpenML Support Squad implemented are strictly related to the extension packages made by the team in project "Sharing deep learning models" and the visualization package. These extension packages include: Keras, PyTorch, ONNX, MXNet extensions. Since each extension uses the same testing framework, namely unittest, the procedures are only described once. Firstly, the primary test procedure for testing each of the extension packages implemented by OpenML Support Squad is described. Afterwards an alternative procedure is provided which simply runs all of the OpenML extension unit tests. Note that this includes the Sklearn tests that are not in the scope of this project and as such OpenML Support Squad is not responsible for passing these tests. Since the visualization is implemented separately, it also has a separate testing procedure using unittest. The difference is small compared to the extension testing. Only the tests described in sections 4.1 and 4.3 are within the scope.

4.1 Extensions test procedure

This section provides the procedure to run the tests that are in the scope of this project. The procedure should be followed for all extensions described in the User Requirements Document (URD).

4.1.1 Test procedure identifier

UT - Number of the unit test

4.1.2 Purpose

This procedure executes all unit tests that are implemented by OpenML Support Squad. These check whether all implemented extensions adhere to the requirements. If all unit tests pass, the requirements will be fulfilled and everything will be in working order. This procedure will not run unit tests that are outside of the scope of this project. Code coverage and duration are not provided.

4.1.3 Procedure steps

1. Open a terminal in the root folder of the OpenML Python API.
2. Type "`python -m pytest tests/test_extensions/[...]/`" in the terminal and fill in the `[...]` with one of the extension folders listed below:
    - `test_keras_extension`
    - `test_pytorch_extension`
    - `test_onnx_extension`
    - `test_mxnet_extension`

3. Press enter to run the tests.
4. Results are shown in the terminal.

4.2 OpenML complete extension test procedure

In this section the procedure to run all extension library unit tests found in OpenML is provided. This includes tests that fall outside the scope of this project.

4.2.1 Test procedure identifier

UT-0

4.2.2 Purpose

This procedure is used when one wants to run all unit tests in OpenML before deployment or outside to confirm that the whole system is working, or if there is enough time and a simple command is required.

4.2.3 Procedure steps

1. Open a terminal in the root folder of the OpenML Python API.
2. Type "`python -m pytest tests/test_extensions/`" in the terminal.
3. Press enter to run the tests.
4. Results are shown in the terminal.

4.3 Visualization

In this section the procedure to run visualization unit tests is provided.

4.3.1 Test procedure identifier

UT-6

4.3.2 Purpose

This procedure is used to test the visualization portion of the project.

4.3.3 Procedure steps

1. Open a terminal in the root folder of the OpenML Python API.
2. Type "`python -m pytest visualization/tests`" in the terminal.
3. Press enter to run the tests.
4. Results are shown in the terminal.

5 | Test reports

5.1 KerasExtension test report

5.1.1 Serialization

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✔ *+model_to_flow() - Sequential/Functional*

5.1.2 Deserialization

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✔ *+flow_to_model() Sequential/Functional*
- ✔ *+from_paremeters()*
- ✔ *+compile()*

5.1.3 Additional functions

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✔ *+can_handle_model()*
- ✔ *+get_version_information()*
- ✔ *+create_setup_string()*
- ✔ *+is_estimator()*
- ✔ *+is_keras_flow()*
- ✔ *+check_dependencies()*
- ✔ *+openml_param_name_to_keras()*

5.1.4 Model run

Date: 13.06.2019
Tester: Travis Continuous Integration

Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✓ *+run_model_on_fold()*

5.2 PyTorchExtension test report

5.2.1 Serialization

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✓ *+model_to_flow()*

5.2.2 Deserialization

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✓ *+flow_to_model() - Nested/Non-nested*

5.2.3 Additional functions

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✓ *+can_handle_model()*
- ✓ *+get_version_information()*
- ✓ *+create_setup_string()*
- ✓ *+is_estimator()*
- ✓ *+is_pytorch_flow()*
- ✓ *+check_dependencies()*
- ✓ *+openml_param_name_to_torch()*

5.2.4 Model run

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✓ *+run_model_on_fold() - Classification/Regression*

5.3 ONNX Extension test report

5.3.1 Serialization

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Veselin Minev & Claudiu-Teodor Nohai
Verdict: Pass

- ✓ *+model_to_flow() - Sequential/Functional*

5.3.2 Deserialization

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Veselin Minev & Claudiu-Teodor Nohai
Verdict: Pass

- ✓ *+flow_to_model()*

5.3.3 Additional functions

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Veselin Minev & Claudiu-Teodor Nohai
Verdict: Pass

- ✓ *+can_handle_model()*
- ✓ *+get_version_information()*
- ✓ *+create_setup_string()*
- ✓ *+is_estimator()*
- ✓ *+is_onnx_flow()*

5.3.4 Model run

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Veselin Minev & Claudiu-Teodor Nohai
Verdict: Pass

- ✓ *+run_model_on_fold() - Classification/Regression*

5.4 MXNetExtension test report

5.4.1 Serialization

Date: 13.06.2019

Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✓ *+model_to_flow() - Only Sequential*

5.4.2 Deserialization

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✓ *+flow_to_model()*

5.4.3 Additional functions

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✓ *+can_handle_model()*
- ✓ *+get_version_information()*
- ✓ *+create_setup_string()*
- ✓ *+is_estimator()*
- ✓ *+is_mxnet_flow()*
- ✓ *+check_dependencies()*

5.4.4 Model run

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Adrian-Stefan Mares & Dragos Mihai Serban
Verdict: Pass

- ✓ *+run_model_on_fold()*

5.5 Visualization test report

5.5.1 Flow Visualization

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Veselin Minev & Claudiu-Teodor Nohai
Verdict: Pass

- ✓ *+update_flow_info_texts_visibility()*
- ✓ *+update_flow_graph_text()*
- ✓ *+init_flow_loading()*
- ✓ *+update_flow_loading_info()*
- ✓ *+load_flow()*
- ✓ *+update_flow_error_text()*
- ✓ *+update_flow_graph_visibility()*
- ✓ *+update_flow_graph()*

5.5.2 Run Visualization

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Veselin Minev & Claudiu-Teodor Nohai
Verdict: Pass

- ✓ *+update_run_info_texts_visibility()*
- ✓ *+update_run_graph_text()*
- ✓ *+init_run_loading()*
- ✓ *+update_run_loading_info()*
- ✓ *+load_run()*
- ✓ *+update_run_error_text()*
- ✓ *+update_run_graph_visibility()*
- ✓ *+update_run_graph()*

5.5.3 Utilities Visualization

Date: 13.06.2019
Tester: Travis Continuous Integration
Witness: Veselin Minev & Claudiu-Teodor Nohai
Verdict: Pass

- ✓ *+has_error_or_is_loading()*
- ✓ *+get_info_text_styles()*
- ✓ *+get_loading_info()*
- ✓ *+get_error_text()*
- ✓ *+get_visibility_style()*
- ✓ *+create_figure()*
- ✓ *+extract_run_graph_data()*
- ✓ *+add_lists_element_wise()*
- ✓ *+get_training_data()*
- ✓ *+get_onnx_model()*

Bibliography

[1] OpenML Support Squad. *DL Extension Library, SRD Software Requirement Document version 1.0.0.*

[2] Software Standardisation and Control. *ESA software engineering standards. 1991.*

[3] OpenML. *OpenML Documentation.* URL: `https://docs.openml.org/` (visited on 09/05/2019).

[4] Wikipedia. *Apache MXNet.* URL: `https://en.wikipedia.org/wiki/Apache_MXNet` (visited on 04/26/2019).

[5] Keras. *Keras homepage.* URL: `https://keras.io/` (visited on 09/05/2019).

[6] Wikipedia. *Artificial Neural Network.* URL: `https://en.wikipedia.org/wiki/Artificial_neural_network` (visited on 04/30/2019).

[7] ONNX. *ONNX homepage.* URL: `https://onnx.ai/` (visited on 04/26/2019).

[8] PyTorch. *PyTorch homepage.* URL: `https://pytorch.org/` (visited on 09/05/2019).

[9] *Travis CI.* URL: `https://en.wikipedia.org/wiki/Travis_CI` (visited on 05/20/2019).