Documentation regarding Flow Visualization

This document provides an overview of the current “flow visualization” deliverables. During my research, I uncovered many bottlenecks and things that require further research. Each of these Next Steps will also be elaborated upon.

In case of questions, feel free to reach out to Thomas Boot (t.a.boot@outlook.com)!

1. Deliverable

I have been working on an initial trial version on how we can visualize our openML flows. After some research on different visualization tools, we landed on Netron for this purpose. Some quick notes:

- Netron allows you to load a model either from file of via URL.
- It provides a comprehensive visualization, including data shapes and model parameters.
- It accept pretty much any model file format. However, you might sometimes still encounter some bugs (as I did when loading torch models). Therefore, converting flows to an onnx format is a safe way to go, this format is very well supported with Netron.

I have set out to write a first trial version which enables the conversion of *sklearn* flows to onnx format. In order to do so, I have been focused on developing an approach which:

- Converts an openML flow to sklearn pipeline.
- Converts the sklearn pipeline to onnx format.
- Loads the onnx model via Netron

Quite quickly into this process, we stumbled upon issues with package versions (i.e., every flow is built with a specific sklearn version and requires exactly this package version in order to load the openML flow). Hence, my actual deliverable has been to develop dockerfiles, one for each distinct sklearn configuration I could find among sklearn flows. This resulted in 59 dockerfiles. While some of these are successful in converting the pipelines to onnx format, there are quite a few (fundamental) issues I ran into. In the next chapter, I will elaborate on some of the main issues I discovered (and did not manage to resolve), as well as some recommendations.

1. Next Steps

Below a recap of the issues I encountered when building and running the dockerfiles. I also briefly elaborate on some recommendations how to tackle these issues. Some next steps regarding the visualization and integration of Netron are also provided.

1.1 Issues with Dockerfiles

Below an overview of the issues:

Docker	Sklearn	python	Issues
0 – 2	< 0.17	3.6	Not compatible with openML Python version too old
4 – 9, 11 – 13, 39 – 41	0.18 - 0.19	3.6	Python version too old Not compatible with skl2onnx
16 – 19, 21 – 24, 26, 27, 29 – 35, 37, 38, 42 – 45, 47, 49, 52	0.20 - 0.24	3.7	Successful build (even though many depreciation warnings) Issues with script
46, 48, 50, 53 – 59	> 1.0	3.8	Successful build Issues with script
3, 10, 14, 15 , 20, 25, 28, 36, 51	.dev0, .rc1 or .post1 in name	various	Error during flow retrieval

1.1.1 Non-compatibility with openML

Dockers that require an sklearn version older than 0.18, cannot build properly. The issue I keep receiving is that openML requires sklearn >= 0.18. I have not been successful in finding an older openML version.

1.1.2 Too old python version

Related to the above, I get errors whenever using a python version older than 3.7. It appears that some underlying function in the openML package require imports that are only supported for python 3.7 of newer. One of the error I commonly got was “Cannot import module *dataclasses*”.

1.1.3 Non-compatibility with skl2onnx

For dockers that require sklearn older than 0.20, I received errors during build regarding incompatibility with skl2onnx (the package we need for onnx conversion). I did some research and there should be older skl2onnx versions out there that are compatible, but I have not been successful in doing so.

1.1.4 Issues within the script

Good news – dockers with sklearn 0.20 or newer build successfully! This is also the majority of the openML flows. However, when running these dockers, I got some errors when executing the python script itself. Two errors I have encountered:

- Due to some bugs in the openML-python functions, some flows cannot be reinstated. Particularly pipelines that contain a (MinMax)Scaler. The error I keep getting is that the parameter *feature_range*, which should be a tuple, is stored as a list. This returns an error whenever trying to fit the pipeline. I did some digging, and this appears to be an issue in how flow parameters are being stored in openML.

- In order to convert a pipeline to onnx, you need to fit the pipeline with some dummy data input. I had hoped some generic dummy input would be sufficient. However, some flows require the exact data shapes as the ones on which they have been trained (i.e., think of pipelines that contain a ColumnTransformer). Depending on the flow, I get an error that the input shapes do not match.

1.1.5 Error during flow retrieval

Any docker for which a sklearn version is required that contains *.dev, .rc* or *.post* fails to run properly. First of all, I can't seem to find these package versions in pip. When I try to run the docker with a similar sklearn version (e.g., 0.20.0 instead of 0.20.dev0), I get an error when running. This is particularly because the function which reinstates the flow, checks if the sklearn version matches with the required version. Since the sklearn version is read as a string, I get a TypeError: '<' not supported between instances of 'str' and 'int'.

1.2 Recommendations

Given the issues I encountered, I recommend three things:

- Do not invest in visualising flows that are too old. Trying to reinstate a flow that has been developed with old sklearn versions, is like opening a can of worms.
- Fix some of the smaller bugs found in the openML python code (such as parameters being stored as lists instead of tuples).
- Improve the script by automatically retrieving some run for that particular flow. Based on the run_id, retrieve a dataset on which we know the flow can be trained. Then, fit the pipeline on a small portion of that dataset to ensure the correct data shapes are being used.

1.3 Visualization and integration

Once the above issues have been dealt with, convert all flows to onnx format and store those files somewhere. From that point on, it should be (hopefully) fairly easy to visualize the flows. I have already provided some small app to demonstrate how easy to use Netron is. All that remains is integrating some button in the openML flow webpage. Upon clicking this button, you should retrieve the onnx file for that flow and open it in Netron.