9. Model persistence

Summary of model persistence methods

Persistence method	Pros	Risks / Cons
ONNX	• Serve models without a Python environment • Serving and training environments independent of one another • Most secure option	• Not all scikit-learn models are supported • Custom estimators require more work to support • Original Python object is lost and cannot be reconstructed
skops.io	• More secure than `pickle` based formats • Contents can be partly validated without loading	• Not as fast as `pickle` based formats • Supports less types than `pickle` based formats • Requires the same environment as the training environment
`pickle`	• Native to Python • Can serialize most Python objects • Efficient memory usage with `protocol=5`	• Loading can execute arbitrary code • Requires the same environment as the training environment
`joblib`	• Efficient memory usage • Supports memory mapping • Easy shortcuts for compression and decompression	• Pickle based format • Loading can execute arbitrary code • Requires the same environment as the training environment
cloudpickle	• Can serialize non-packaged, custom Python code	• Pickle based format • Loading can execute arbitrary code

Persistence method	Pros	Risks / Cons
	• Comparable loading efficiency as `pickle` with `protocol=5`	• No forward compatibility guarantees • Requires the same environment as the training environment

After training a scikit-learn model, it is desirable to have a way to persist the model for future use without having to retrain. Based on your use-case, there are a few different ways to persist a scikit-learn model, and here we help you decide which one suits you best. In order to make a decision, you need to answer the following questions:

1. Do you need the Python object after persistence, or do you only need to persist in order to serve the model and get predictions out of it?

If you only need to serve the model and no further investigation on the Python object itself is required, then ONNX might be the best fit for you. Note that not all models are supported by ONNX.

In case ONNX is not suitable for your use-case, the next question is:

2. Do you absolutely trust the source of the model, or are there any security concerns regarding where the persisted model comes from?

If you have security concerns, then you should consider using skops.io which gives you back the Python object, but unlike `pickle` based persistence solutions, loading the persisted model doesn't automatically allow arbitrary code execution. Note that this requires manual investigation of the persisted file, which `skops.io` allows you to do.

The other solutions assume you absolutely trust the source of the file to be loaded, as they are al susceptible to arbitrary code execution upon loading the persisted file since they all use the pickle protocol under the hood.

3. Do you care about the performance of loading the model, and sharing it between processes where a memory mapped object on disk is beneficial?

If yes, then you can consider using joblib. If this is not a major concern for you, then you can use the built-in `pickle` module.

4. Did you try `pickle` or `joblib` and found that the model cannot be persisted? It can happen for instance when you have user defined functions in your model.

If yes, then you can use cloudpickle which can serialize certain objects which cannot be serialized by `pickle` or `joblib`.

9.1. Workflow Overview

In a typical workflow, the first step is to train the model using scikit-learn and scikit-learn compatible libraries. Note that support for scikit-learn and third party estimators varies across the different persistence methods.

9.1.1. Train and Persist the Model

Creating an appropriate model depends on your use-case. As an example, here we train a `sklearn.ensemble.HistGradientBoostingClassifier` on the iris dataset:

```
>>> from sklearn import ensemble
>>> from sklearn import datasets
>>> clf = ensemble.HistGradientBoostingClassifier()
>>> X, y = datasets.load_iris(return_X_y=True)
>>> clf.fit(X, y)
HistGradientBoostingClassifier()
```

Once the model is trained, you can persist it using your desired method, and then you can load the model in a separate environment and get predictions from it given input data. Here there are two major paths depending on how you persist and plan to serve the model:

- ONNX: You need an `ONNX` runtime and an environment with appropriate dependencies installed to load the model and use the runtime to get predictions. This environment can be minimal and does not necessarily even require Python to be installed to load the model and compute predictions. Also note that `onnxruntime` typically requires much less RAM than Python to compute predictions from small models.
- `skops.io`, `pickle`, `joblib`, cloudpickle: You need a Python environment with the appropriate dependencies installed to load the model and get predictions from it. This environment should have the same **packages** and the same **versions** as the environment where the model was trained. Note that none of these methods support loading a model trained with a different version of scikit-learn, and possibly different versions of other dependencies such as `numpy` and `scipy`. Another concern would be running the persisted model on a different hardware, and in most cases you should be able to load your persisted model on a different hardware.

9.2. ONNX

`ONNX`, or Open Neural Network Exchange format is best suitable in use-cases where one needs to persist the model and then use the persisted artifact to get predictions without the need to load the Python object itself. It is also useful in cases where the serving environment needs to be lean and minimal, since the `ONNX` runtime does not require `python`.

`ONNX` is a binary serialization of the model. It has been developed to improve the usability of the interoperable representation of data models. It aims to facilitate the conversion of the data models between different machine learning frameworks, and to improve their portability on different computing architectures. More details are available from the ONNX tutorial. To convert scikit-learn model to `ONNX` sklearn-onnx has been developed. However, not all scikit-learn models are supported, and it is limited to the core scikit-learn and does not support most third party estimators. One can write a custom converter for third party or custom estimators, but the documentation to do that is sparse and it might be challenging to do so.

Using ONNX

9.3. `skops.io`

`skops.io` avoids using `pickle` and only loads files which have types and references to functions which are trusted either by default or by the user. Therefore it provides a more secure format than `pickle`, `joblib`, and cloudpickle.

Using skops

9.4. `pickle`, `joblib`, and `cloudpickle`

These three modules / packages, use the `pickle` protocol under the hood, but come with slight variations:

- `pickle` is a module from the Python Standard Library. It can serialize and deserialize any Python object, including custom Python classes and objects.
- `joblib` is more efficient than `pickle` when working with large machine learning models or large numpy arrays.
- cloudpickle can serialize certain objects which cannot be serialized by `pickle` or `joblib`, such as user defined functions and lambda functions. This can happen for instance, when

using a `FunctionTransformer` and using a custom function to transform the data.

Using `pickle`, `joblib`, or `cloudpickle`

9.5. Security & Maintainability Limitations

`pickle` (and `joblib` and `clouldpickle` by extension), has many documented security vulnerabilities by design and should only be used if the artifact, i.e. the pickle-file, is coming from a trusted and verified source. You should never load a pickle file from an untrusted source, similarly to how you should never execute code from an untrusted source.

Also note that arbitrary computations can be represented using the `ONNX` format, and it is therefore recommended to serve models using `ONNX` in a sandboxed environment to safeguard against computational and memory exploits.

Also note that there are no supported ways to load a model trained with a different version of scikit-learn. While using `skops.io`, `joblib`, `pickle`, or cloudpickle, models saved using one version of scikit-learn might load in other versions, however, this is entirely unsupported and inadvisable. It should also be kept in mind that operations performed on such data could give different and unexpected results, or even crash your Python process.

In order to rebuild a similar model with future versions of scikit-learn, additional metadata should be saved along the pickled model:

- The training data, e.g. a reference to an immutable snapshot
- The Python source code used to generate the model
- The versions of scikit-learn and its dependencies
- The cross validation score obtained on the training data

This should make it possible to check that the cross-validation score is in the same range as before.

Aside for a few exceptions, persisted models should be portable across operating systems and hardware architectures assuming the same versions of dependencies and Python are used. If you encounter an estimator that is not portable, please open an issue on GitHub. Persisted models are often deployed in production using containers like Docker, in order to freeze the environment and dependencies.

If you want to know more about these issues, please refer to these talks:

- Adrin Jalali: Let's exploit pickle, and skops to the rescue! | PyData Amsterdam 2023.
- Alex Gaynor: Pickles are for Delis, not Software - PyCon 2014.

9.5.1. Replicating the training environment in production

If the versions of the dependencies used may differ from training to production, it may result in unexpected behaviour and errors while using the trained model. To prevent such situations it is recommended to use the same dependencies and versions in both the training and production environment. These transitive dependencies can be pinned with the help of package management tools like `pip` , `mamba` , `conda` , `poetry` , `conda-lock` , `pixi` , etc.

It is not always possible to load an model trained with older versions of the scikit-learn library and its dependencies in an updated software environment. Instead, you might need to retrain the model with the new versions of the all the libraries. So when training a model, it is important to record the training recipe (e.g. a Python script) and training set information, and metadata about all the dependencies to be able to automatically reconstruct the same training environment for the updated software.

InconsistentVersionWarning

9.5.2. Serving the model artifact

The last step after training a scikit-learn model is serving the model. Once the trained model is successfully loaded, it can be served to manage different prediction requests. This can involve deploying the model as a web service using containerization, or other model deployment strategies, according to the specifications.

9.6. Summarizing the key points

Based on the different approaches for model persistence, the key points for each approach can be summarized as follows:

- `ONNX` : It provides a uniform format for persisting any machine learning or deep learning model (other than scikit-learn) and is useful for model inference (predictions). It can however, result in compatibility issues with different frameworks.

- `skops.io` : Trained scikit-learn models can be easily shared and put into production using `skops.io` . It is more secure compared to alternate approaches based on `pickle` because it does not load arbitrary code unless explicitly asked for by the user. Such code needs to be packaged and importable in the target Python environment.
- `joblib` : Efficient memory mapping techniques make it faster when using the same persisted model in multiple Python processes when using `mmap_mode="r"` . It also gives easy shortcuts to compress and decompress the persisted object without the need for extra code. However, it may trigger the execution of malicious code when loading a model from an untrusted source as any other pickle-based persistence mechanism.
- `pickle` : It is native to Python and most Python objects can be serialized and deserialized using `pickle` , including custom Python classes and functions as long as they are defined in a package that can be imported in the target environment. While `pickle` can be used to easily save and load scikit-learn models, it may trigger the execution of malicious code while loading a model from an untrusted source. `pickle` can also be very efficient memorywise if the model was persisted with `protocol=5` but it does not support memory mapping.
- cloudpickle: It has comparable loading efficiency as `pickle` and `joblib` (without memory mapping), but offers additional flexibility to serialize custom Python code such as lambda expressions and interactively defined functions and classes. It might be a last resort to persis pipelines with custom Python components such as a `sklearn.preprocessing.FunctionTransformer` that wraps a function defined in the training script itself or more generally outside of any importable Python package. Note that cloudpickle offers no forward compatibility guarantees and you might need the same version of cloudpickle to load the persisted model along with the same version of all the libraries used to define the model. As the other pickle-based persistence mechanisms, it may