Pickle

Introduction

Pickle is an invaluable tool for saving objects. In this lesson you will learn how to use it on various different Python data types.

Objectives

You will be able to:

- Describe the circumstances in which you would want to use a pickle file
- Write a pickle file
- Read a pickle file
- Use the `joblib` library to pickle and load a scikit-learn class

Data Serialization

Think about the importance of being able to save data files to CSV, or another format. For example, you start with a raw dataset which you may have downloaded from the web. Then you painstakingly take hours preprocessing the data, cleaning it, constructing features, aggregates, and other views. In order to avoid having to rerun your entire process, you are apt to save the current final cleaned version of the dataset into a serialized format like CSV.

`pickle` allows you to go beyond just storing data in a format like CSV, and save any object that is currently loaded into your Python interpreter. Literally anything. You could save data stored in a `dict`, `list`, or `set` as a pickle file. You can also save functions or class instances as pickle files. Saving models is one of the important use cases of this technique.

Pickling Base Python Objects

Let's say we have this nested data structure example (from the Python docs (https://docs.python.org/3/library/pickle.html#examples)), which would not be suitable for storage as a CSV because the values are different data types:

In [1]:
```
data_object = {
    'a': [1, 2.0, 3, 4+6j],
    'b': ('character string', b'byte string'),
    'c': {None, True, False}
}
```

Importing `pickle`

`pickle` is a module built in to Python that is suitable for pickling base Python objects. You can import it like this:

In [2]:
```
import pickle
```

Writing Objects to Pickle

Let's store this object as a file on disk called `data.pickle`.

1. Open a file called `'data.pickle'`.
   A. The `.pickle` file extension is conventional for Python 3 objects. You can use a different file name or extension and it won't make a difference as far as Python is concerned, but we recommend using `.pickle` so it's clear what the file is
2. We'll need to open the file using mode `'wb'`.
   A. `w` because we want to write data to the file (and automatically create the file if it doesn't exist yet)
   B. `b` because we specifically want to write binary data. `pickle` uses a binary protocol, so it won't work if you don't include the `b`
3. Then we can use the `dump` function from the `pickle` module to write our data object.

In [3]:
```
with open('data.pickle', 'wb') as f:
    pickle.dump(data_object, f)
```

Importing Objects from Pickle Files

Go ahead and restart the kernel. Now if we try to access the original `data_object` it won't work:

In [1]:
```
try:
    print(data_object)
except NameError as e:
    print(type(e), e)
```

```
<class 'NameError'> name 'data_object' is not defined
```

But we can use `pickle` to load it back into memory with a new name, `data_object2`.

1. Open the file `data.pickle` since that is the file name we saved prior to restarting the kernel.
2. We'll need to open the file using mode `'rb'`.
    A. `r` for read
    B. `b` for binary
3. Then we can use the `load` function from the `pickle` module to read our data object.

In [2]:
```
import pickle
with open('data.pickle', 'rb') as f:
    data_object2 = pickle.load(f)
data_object2
```

Out[2]:
```
{'a': [1, 2.0, 3, (4+6j)],
 'b': ('character string', b'byte string'),
 'c': {False, None, True}}
```

Important reminder: DO NOT open pickle files unless you trust the source (e.g. you created them yourself). They can contain malicious code and there are not any built-in security constraints on them.

Pickle with scikit-learn

So far, your process has typically been to instantiate a model, train it, evaluate it, maybe make some predictions, then shut down the notebook. This means that the time and computational resources used to train the model are lost, and would need to be repeated if you ever wanted to use the model again.

If you pickle your fitted model instead, then all you will need to do is load it back into memory, then it will be all ready to make predictions!

Instantiating and Fitting a Model

Below we fit a simple linear regression model:

In [3]:
```
from sklearn.linear_model import LinearRegression

# y = x + 1
X = [[1],[2],[3],[4],[5]]
y = [2, 3, 4, 5, 6]

model = LinearRegression()
model.fit(X, y)

print(f"Fitted model is y = {model.coef_[0]}x + {model.intercept_}")
```

```
Fitted model is y = 1.0x + 1.0
```

We can now use the model to make predictions:

In [4]:
```
model.predict([[7], [8], [9]])
```

Out[4]:
```
array([ 8.,  9., 10.])
```

Importing `joblib`

For scikit-learn models, it is possible to use the `pickle` module but not recommended because it is less efficient. (See documentation here (https://scikit-learn.org/stable/modules/model_persistence.html).) Instead, we'll use the `joblib` library.

(Note that we still often use the language of "pickle file" and "pickling" even if we are using a different library such as `joblib`.)

In [5]:
```
import joblib
```

Writing Objects with `joblib`

Let's save `model` using `joblib`.

1. Once again we need to open a file to store the model in.
    A. Instead of `'data'`, we'll call this `'regression_model'` so it's clear what the file contains
    B. Instead of the `.pickle` file extension, we'll use `.pkl`

   a. This is the conventional file ending for scikit-learn models, and used to be the standard for all Python objects in Python 2
   b. Neither Python nor `joblib` nor scikit-learn will enforce this file ending. So you also might see examples with `.pickle` or `.joblib` file extensions, or some other ending, even though it was serialized using this technique
   c. If you see a serialized model ending with `.zip`, `.pb`, or `.h5`, that means it is likely not a scikit-learn model and probably was not serialized using `joblib` or `pickle`
2. The mode (`'wb'`) is the same as with `pickle`.
3. The function name (`dump`) is also the same as with `pickle`.

In [6]:
```
with open('regression_model.pkl', 'wb') as f:
    joblib.dump(model, f)
```

Importing Objects with `joblib`

Now, restart the kernel. Once again, we would expect an error if we tried to use the `model` variable:

In [1]:
```
try:
    print(model.predict([[10], [11], [12]]))
except NameError as e:
    print(type(e), e)
```

```
<class 'NameError'> name 'model' is not defined
```

But we can load the model from the pickled file:

1. Open file `'regression_model.pkl'`.
2. Use mode `'rb'`.
3. Use the `load` function (this time from `joblib`).

In [2]:
```
import joblib
with open('regression_model.pkl', 'rb') as f:
    model2 = joblib.load(f)

print(f"Loaded model is y = {model2.coef_[0]}x + {model2.intercept_}")
```

```
Loaded model is y = 1.0x + 1.0
```

Note that the coefficient and intercept are the same as the original model. While this would have been a simple model to re-fit, you can imagine how this would save significant time with a more-complex model or with large production datasets.

Now we can make predictions again!

In [3]:
```
model2.predict([[10], [11], [12]])
```

Out[3]:
```
array([11., 12., 13.])
```

Additional Resources

- Pickle Documentation (https://docs.python.org/3/library/pickle.html)
- scikit-learn Persistence Documentation (using `joblib`) (https://scikit-learn.org/stable/modules/model_persistence.html)

Summary

In this brief lesson you saw how to both save objects to pickle files and import objects from pickle files. This can be particularly useful for saving models that are non deterministic and would otherwise be difficult or impossible to reproduce exact replicas of.