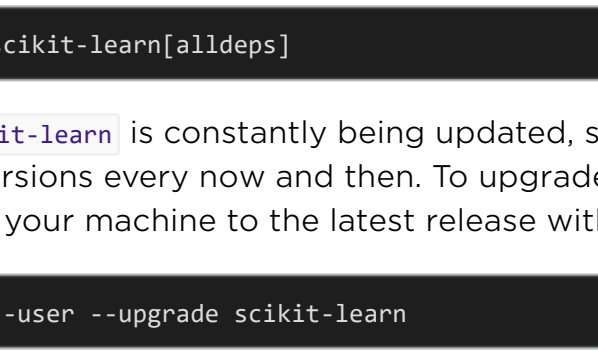
Theory: Introduction to sklearn

7 minutes 0 / 4 problems solved

This topic is an introduction to `scikit-learn`, arguably the most popular ML library in Python.



§1. Installing scikit-learn

Since `scikit-learn` is not a part of the Python standard library, you might need to install it on your machine before you can start using it. One way to do so is by using `pip`. Run the following command in the command line:

```
pip install scikit-learn
```

Note that the `scikit-learn` requires such Python libraries as `numpy` and `scipy`. If you have not installed them yet, depending on your operating system, these libraries will be either automatically installed along with `scikit-learn`, or will need to be installed manually beforehand. This can be easily done with `pip` as well. Alternatively, you can automatically install `scikit-learn` along with all its dependencies by running the following command:

```
pip install scikit-learn[alldeps]
```

Also, note that `scikit-learn` is constantly being updated, so it is a good idea to check for new versions every now and then. To upgrade the current version installed on your machine to the latest release with `pip`, run

```
pip install --user --upgrade scikit-learn
```

Once the library is installed, you can import it in your code and start using it. Importing the whole `scikit-learn` is typically overkill. Normally, you will be importing only the modules you are intending to use. The typical import statement will look like this:

```
from sklearn.module_of_interest import functionality_of_interest
```

To import the whole module, use * :

```
from sklearn.module_of_interest import *
```

You might have noticed that we use the name `sklearn`, not `scikit-learn`, in the import statements above. Well, the former is just a conventional abbreviation of the name of the package. You can use either to install the package, but only the abbreviated version can be used for importing it. In our text, we will use the short `sklearn` from now on, since this is how most users refer to the library in practice.

§2. What's good about sklearn?

There are a number of things that make `sklearn` this popular. Here are some of them.

To begin with, `sklearn` can be used for various tasks within the data mining process, such as data pre-processing, training machine learning models, and model selection. It is really convenient for machine learning practitioners to have this diverse functionality in one place. Besides, `sklearn` integrates well

Current topic:

Introduction to sklearn

Topic depends on:

- Typical ML pipeline
- Pip (Stage 1)

Table of contents:

with other popular Python libraries useful for data analysis, for example, `pandas` for storing and manipulating data, `numpy` for vector computations and `matplotlib` and `seaborn` for data visualization.

Another advantage of `sklearn` is that this library arguably features all widely-used algorithms for the most common machine learning problems such as classification, regression, clustering, etc. Whichever algorithm you want to use with your data, most likely, you will find it inside `sklearn`. Convenient, right?

Part of its success is also due to a simple intuitive interface. Fitting any ML model in `sklearn` is very simple, even if you know almost nothing about machine learning. Moreover, the interface is consistent, so your code will look more or less the same, whether you are solving a classification, regression, or a clustering problem. The same goes for data preprocessing techniques.

Apart from that, `sklearn` has an excellent user guide abound with explanations and examples.

And, finally, `sklearn` is an open-source library, meaning that everyone can view the source code and contribute to it. It also means that `sklearn` is free to use for both personal and commercial purposes, which is why many companies chose it over other expensive data analysis software.

§3. Is sklearn really so perfect?

By now you are probably already convinced that `sklearn` is a wonderful tool. Well, you should keep in mind that the library unfortunately also has some considerable weaknesses.

The major downside is speed. The `sklearn` implementation of the machine learning algorithms cannot compete in speed with their lower-level implementations (for example, in C++). So, if speed is an important factor for your project, or if you are working with really big datasets, you may want to use other tools. However, `sklearn` could still be useful for quick proof of concept on a smaller portion of the data.

Besides, you should be aware of the fact that `sklearn` code might contain bugs. That is natural. After all, this library is created by people, not robots. Be critical about the results you get and make sure everything works as expected. A good way to check this is to come up with some toy examples where the results of the analysis are known to you, and run your code on them. If you think you found a bug, you can report it using the Bug Tracker, so that `sklearn` contributors could fix it, or even fix it yourself.

Also, always study the documentation carefully. The default behavior of some methods in `sklearn` implementation can be different from what you expect.

§4. Conclusions

- `scikit-learn` is a Python library for data mining and machine learning.
- `sklearn` is short for `scikit-learn` and should be used to import the package.
- It has rich functionality and is quite easy to use.
- It's a free, open-source library.
- The `scikit-learn` implementation may contain bugs.

Report a typo

19 users liked this theory. **0** didn't like it. **What about you?**

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion