parenttoc: True

Parallelism, resource management, and configuration

Parallelism

Some scikit-learn estimators and utilities can parallelize costly operations using multiple CPU cores, thanks to the following components:

- via the joblib library. In this case the number of threads or processes can be controlled with the `n_jobs` parameter.
- via OpenMP, used in C or Cython code.

In addition, some of the numpy routines that are used internally by scikit-learn may also be parallelized if numpy is installed with specific numerical libraries such as MKL, OpenBLAS, or BLIS.

We describe these 3 scenarios in the following subsections.

Joblib-based parallelism

When the underlying implementation uses joblib, the number of workers (threads or processes) that are spawned in parallel can be controlled via the `n_jobs` parameter.

> **Note**
>
> Where (and how) parallelization happens in the estimators is currently poorly documented. Please help us by improving our docs and tackle issue 14228!

Joblib is able to support both multi-processing and multi-threading. Whether joblib chooses to spawn a thread or a process depends on the **backend** that it's using.

Scikit-learn generally relies on the `loky` backend, which is joblib's default backend. Loky is a multi-processing backend. When doing multi-processing, in order to avoid duplicating the memory in each process (which isn't reasonable with big datasets), joblib will create a memmap that all processes can share, when the data is bigger than 1MB.

In some specific cases (when the code that is run in parallel releases the GIL), scikit-learn will indicate to `joblib` that a multi-threading backend is preferable.

As a user, you may control the backend that joblib will use (regardless of what scikit-learn recommends) by using a context manager:

```
from joblib import parallel_backend

with parallel_backend('threading', n_jobs=2):
    # Your scikit-learn code here
```

Please refer to the joblib's docs for more details.

In practice, whether parallelism is helpful at improving runtime depends on many factors. It is usually a good idea to experiment rather than assuming that increasing the number of workers is always a good thing. In some cases it can be highly detrimental to performance to run multiple copies of some estimators or functions in parallel (see oversubscription below).

OpenMP-based parallelism

OpenMP is used to parallelize code written in Cython or C, relying on multi-threading exclusively. By default (and unless joblib is trying to avoid oversubscription), the implementation will use as many threads as possible.

You can control the exact number of threads that are used via the `OMP_NUM_THREADS` environment variable:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\computing\(scikit-learn-main)(doc)(computing)parallelism.rst, line 84)**
>
> Unknown directive type "prompt".
>
> ```
> .. prompt:: bash $
>
>     OMP_NUM_THREADS=4 python my_script.py
> ```

Parallel Numpy routines from numerical libraries

Scikit-learn relies heavily on NumPy and SciPy, which internally call multi-threaded linear algebra routines implemented in libraries such as MKL, OpenBLAS or BLIS.

The number of threads used by the OpenBLAS, MKL or BLIS libraries can be set via the `MKL_NUM_THREADS`, `OPENBLAS_NUM_THREADS`, and `BLIS_NUM_THREADS` environment variables.

Please note that scikit-learn has no direct control over these implementations. Scikit-learn solely relies on Numpy and Scipy.

> **Note**
>
> At the time of writing (2019), NumPy and SciPy packages distributed on pypi.org (used by `pip`) and on the conda-forge channel are linked with OpenBLAS, while conda packages shipped on the "defaults" channel from anaconda.org are linked by default with MKL.

Oversubscription: spawning too many threads

It is generally recommended to avoid using significantly more processes or threads than the number of CPUs on a machine. Over-subscription happens when a program is running too many threads at the same time.

Suppose you have a machine with 8 CPUs. Consider a case where you're running a :class:`~sklearn.model_selection.GridSearchCV` (parallelized with joblib) with `n_jobs=8` over a :class:`~sklearn.ensemble.HistGradientBoostingClassifier` (parallelized with OpenMP). Each instance of :class:`~sklearn.ensemble.HistGradientBoostingClassifier` will spawn 8 threads (since you have 8 CPUs). That's a total of `8 * 8 = 64` threads, which leads to oversubscription of physical CPU resources and to scheduling overhead.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\computing\(scikit-learn-main)(doc)(computing)parallelism.rst`, **line 116**); ***backlink***
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\computing\(scikit-learn-main)(doc)(computing)parallelism.rst`, **line 116**); ***backlink***
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\computing\(scikit-learn-main)(doc)(computing)parallelism.rst`, **line 116**); ***backlink***
>
> Unknown interpreted text role "class".

Oversubscription can arise in the exact same fashion with parallelized routines from MKL, OpenBLAS or BLIS that are nested in joblib calls.

Starting from `joblib >= 0.14`, when the `loky` backend is used (which is the default), joblib will tell its child **processes** to limit the number of threads they can use, so as to avoid oversubscription. In practice the heuristic that joblib uses is to tell the processes to use `max_threads = n_cpus // n_jobs`, via their corresponding environment variable. Back to our example from above, since the joblib backend of :class:`~sklearn.model_selection.GridSearchCV` is `loky`, each process will only be able to use 1 thread instead of 8, thus mitigating the oversubscription issue.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\computing\(scikit-learn-main)(doc)(computing)parallelism.rst`, **line 129**); ***backlink***
>
> Unknown interpreted text role "class".

Note that:

- Manually setting one of the environment variables (`OMP_NUM_THREADS`, `MKL_NUM_THREADS`, `OPENBLAS_NUM_THREADS`, or `BLIS_NUM_THREADS`) will take precedence over what joblib tries to do. The total number of threads will be `n_jobs * <LIB>_NUM_THREADS`. Note that setting this limit will also impact your computations in the main process, which will only use `<LIB>_NUM_THREADS`. Joblib exposes a context manager for finer control over the number of threads in its workers (see joblib docs linked below).
- Joblib is currently unable to avoid oversubscription in a multi-threading context. It can only do so with the `loky` backend (which spawns processes).

You will find additional details about joblib mitigation of oversubscription in joblib documentation.

Configuration switches

Python runtime

:func:`sklearn.set_config` controls the following behaviors:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\computing\(scikit-learn-main)(doc)(computing)parallelism.rst, line 164);** ***backlink***
>
> Unknown interpreted text role "func".

assume_finite

Used to skip validation, which enables faster computations but may lead to segmentation faults if the data contains NaNs.

working_memory

The optimal size of temporary arrays used by some algorithms.

Environment variables

These environment variables should be set before importing scikit-learn.

SKLEARN_ASSUME_FINITE

Sets the default value for the *assume_finite* argument of :func:`sklearn.set_config`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\computing\(scikit-learn-main)(doc)(computing)parallelism.rst, line 187);** ***backlink***
>
> Unknown interpreted text role "func".

SKLEARN_WORKING_MEMORY

Sets the default value for the *working_memory* argument of :func:`sklearn.set_config`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\computing\(scikit-learn-main)(doc)(computing)parallelism.rst, line 193);** ***backlink***
>
> Unknown interpreted text role "func".

SKLEARN_SEED

Sets the seed of the global random generator when running the tests, for reproducibility.

Note that scikit-learn tests are expected to run deterministically with explicit seeding of their own independent RNG instances instead of relying on the numpy or Python standard library RNG singletons to make sure that test results are independent of the test execution order. However some tests might forget to use explicit seeding and this variable is a way to control the initial state of the aforementioned singletons.

SKLEARN_TESTS_GLOBAL_RANDOM_SEED

Controls the seeding of the random number generator used in tests that rely on the *global_random_seed*` fixture.

All tests that use this fixture accept the contract that they should deterministically pass for any seed value from 0 to 99 included.

If the *SKLEARN_TESTS_GLOBAL_RANDOM_SEED* environment variable is set to *"any"* (which should be the case on nightly builds on the CI), the fixture will choose an arbitrary seed in the above range (based on the BUILD_NUMBER or the current day) and all fixtured tests will run for that specific seed. The goal is to ensure that, over time, our CI will run all tests with different seeds while keeping the test duration of a single run of the full test suite limited. This will check that the assertions of tests written to use this fixture are not dependent on a specific seed value.

The range of admissible seed values is limited to [0, 99] because it is often not possible to write a test that can work for any possible seed and we want to avoid having tests that randomly fail on the CI.

Valid values for *SKLEARN_TESTS_GLOBAL_RANDOM_SEED*:

- *SKLEARN_TESTS_GLOBAL_RANDOM_SEED="42"*: run tests with a fixed seed of 42
- *SKLEARN_TESTS_GLOBAL_RANDOM_SEED="40-42"*: run the tests with all seeds between 40 and 42 included
- *SKLEARN_TESTS_GLOBAL_RANDOM_SEED="any"*: run the tests with an arbitrary seed selected between 0 and 99 included
- *SKLEARN_TESTS_GLOBAL_RANDOM_SEED="all"*: run the tests with all seeds between 0 and 99 included. This can take a long time: only use for individual tests, not the full test suite!

If the variable is not set, then 42 is used as the global seed in a deterministic manner. This ensures that, by default, the scikit-learn test

suite is as deterministic as possible to avoid disrupting our friendly third-party package maintainers. Similarly, this variable should not be set in the CI config of pull-requests to make sure that our friendly contributors are not the first people to encounter a seed-sensitivity regression in a test unrelated to the changes of their own PR. Only the scikit-learn maintainers who watch the results of the nightly builds are expected to be annoyed by this.

When writing a new test function that uses this fixture, please use the following command to make sure that it passes deterministically for all admissible seeds on your local machine:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\computing\(scikit-learn-main)(doc)(computing)parallelism.rst, line 255)

Unknown directive type "prompt".

```
.. prompt:: bash $

    SKLEARN_TESTS_GLOBAL_RANDOM_SEED="all" pytest -v -k test_your_test_name
```

SKLEARN_SKIP_NETWORK_TESTS

When this environment variable is set to a non zero value, the tests that need network access are skipped. When this environment variable is not set then network tests are skipped.

SKLEARN_RUN_FLOAT32_TESTS

When this environment variable is set to '1', the tests using the *global_dtype* fixture are also run on float32 data. When this environment variable is not set, the tests are only run on float64 data.

SKLEARN_ENABLE_DEBUG_CYTHON_DIRECTIVES

When this environment variable is set to a non zero value, the *Cython* derivative, *boundscheck* is set to *True*. This is useful for finding segfaults.