# Test environment

| Hardware Information | SG2042 |

| :------------------: | :-----------------------: |

| \*\*Architecture\*\* | \*\*RISC-V64\*\* |

| \*\*Operating system\*\* | \*\*openEuler 24.03 (LTS)\*\* |

| \*\*Python version\*\* | \*\*3.12.2\*\* |

| \*\*GCC version\*\* | \*\*12.3.1\*\* |

| \*\*G++ version\*\* | \*\*12.3.1\*\* |

| \*\*Scikit-learn version\*\* | \*\*1.5.1\*\* |

| \*\*Kernel version\*\* | \*\*6.6.0\*\* |

| \*\*Perf version\*\* | \*\*6.6.0\*\* |

scikit-learn installation dependency version:

| Required Dependencies | Versions |

| :---------------: | :----------: |

| \*\*numpy\*\* | \*\*2.0.0\*\* |

| \*\*scipy\*\* | \*\*1.14.0\*\* |

| \*\*cython\*\* | \*\*3.0.10\*\* |

| \*\*joblib\*\* | \*\*1.4.2\*\* |

| \*\*threadpoolctl\*\* | \*\*3.5.0\*\* |

| \*\*pandas\*\* | \*\*2.2.2\*\* |

| \*\*ninja\*\* | \*\*1.11.1.1\*\* |

# asv\_benchmarks

> Use asv tool to perform Benchmarks test on scikit-learn

\*\*Enter directory:\*\*

```

cd scikit-learn/asv\_benchmarks

```

## Test Preparation

You need to create a virtualenv/conda test environment (automatically created after running the asv command) and perform benchmarks in it. Installing conda is cumbersome, so use virtualenv for testing (virtualenv is a dependent software package of asv and does not need to be installed separately).

Install test dependencies:

```

sudo yum install ccache -y

pip install pandas==2.2.2

pip install git+https://github.com/airspeed-velocity/asv

```

1. Go to the `asv\_benchmarks` directory and patch `asv.conf.json` using the provided patch file `asv\_conf.patch`:

```

cd scikit-learn/asv\_benchmarks

patch asv.conf.json < asv\_conf.patch

```

Patch file content description: Modify `asv\_benchmarks/asv.conf.json`:

- Delete the comments before "install\_command", "uninstall\_command", and "build\_command", and replace the content after "build\_command":

```

"install\_command": ["python -mpip install {wheel\_file}"],

"uninstall\_command": ["return-code=any python -mpip uninstall -y {project}"],

"build\_command": ["python -m build --wheel -o {build\_cache\_dir} {build\_dir}"],

```

- Replace "environment\_type" with "virtualenv":

```

"environment\_type": "virtualenv",

```

- Adjust `"matrix"` dependency version:

```

"matrix": {

"numpy": ["2.0.0"],

"scipy": ["1.14.0"],

"cython": ["3.0.10"],

"joblib": ["1.3.2"],

"threadpoolctl": ["3.2.0"],

"pandas": ["2.2.2"]

},

```

!!! Note that the above modifications have been synchronized in the latest scikit-learn github repository (2024-7-17). If you clone the latest repository, you only need to change the environment creation method `"environment\_type"` to `virtualenv`

2. After patching `asv.conf.json`, you can test the modules in the `asv\_benchmark/benchmarks` directory:

```

//Configure global variables, SKLBENCH\_NJOBS is the number of cores

export SKLBENCH\_NJOBS=[1,4]

// Taking linear\_model as an example, you need to confirm the system information and press Enter for the first time.

asv run -b linear\_model HEAD^!

```

In addition, you can configure the test in `config.json` in the `asv\_benchmark/benchmarks` directory.

3. After the test is completed, use the `asv show` command to view the test results:

```

// Display the test result hash

asv show

// Fill in the obtained hash value to view the test results

asv show <hash>

```

\*\*The test data will be saved in the directory `asv\_benchmarks/results/`\*\*

## Error Reporting

1. When executing the `asv run` command, the following error is reported:

```

LibMambaUnsatisfiableError: Encountered problems while solving: - nothing provides \_python\_rc needed by python-3.12.0rc3-rc3\_hab00c5b\_1\_cpython

```

Reason: This benchmarks test does not currently support Python 3.12

Solution: Make sure 3.9 <= Python <= 3.11

2. When executing the `asv run` command, the following error is reported:

```

Creating environments.

·· Error running /home/--/venv/bin/python3.11 -mvirtualenv --wheel=bundle --setuptools=bundle -p /home/--/venv/bin/python3.11 /home/--/scikit-learn/asv\_benchmarks/env/768c56271ce36447f2f605d6575e1318 (exit status 1)

STDOUT -------->

RuntimeError: failed to build image pip because:

```

Failed to create virtualenv environment

Solution: Reinstall virtualenv

```

pip3 uninstall virtualenv

rm -rf ~/.local/share/virtualenv

pip3 install virtualenv

```

3. When running the asv run command to test the linear\_model module, an error is reported

```

[69.44%]...linear\_model.RidgeBenchmark.peakmem\_fit skipped (setup\_cache failed)

[72.22%]...linear\_model.RidgeBenchmark.peakmem\_predict skipped (setup\_cache failed)

[75.00%1...linear\_model.RidgeBenchmark.time\_fit skipped (setup\_cache failed)

[77.78%]...linear\_model.RidgeBenchmark.time\_predict skipped(setup\_cache failed)

[80.56%]...linear\_model.RidgeBenchmark.track\_test\_score skipped(setup\_cache failed)

[83.33%]...linear\_model.RidgeBenchmark.track\_train\_score skipped (setup\_cache failed)

```

Unable to download Ridge test normally

Solution: Install ccache

```

sudo yum install ccache -y

```

## Test items

\*\*Enter directory:\*\*

```

cd scikit-learn/asv\_benchmarks

```

1. Test case 1: cluster

Run the use case:

```

asv run -b cluster HEAD^!

```

Operation results:

```

cluster.KMeansBenchmark.peakmem\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

================ =========== ========= ============

--init

---------------------------- --------------------

representation algorithm random k-means++

================ =========== ========= ============

dense lloyd 107M 116M

dense elkan 141M 141M

sparse lloyd 138M 140M

sparse elkan 145M 147M

================ =========== ========= ============

started: 2024-07-19 23:31:41, duration: 1.88m

cluster.KMeansBenchmark.peakmem\_predict [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

================ =========== ========= ============

--init

---------------------------- --------------------

representation algorithm random k-means++

================ =========== ========= ============

dense lloyd 93.6M 93.6M

dense elkan 93.6M 93.6M

sparse lloyd 101M 101M

sparse elkan 101M 101M

================ =========== ========= ============

started: 2024-07-19 23:33:33, duration: 6.90s

```

2. Test case 2: decomposition

Run the use case:

```

asv run -b decomposition HEAD^!

```

Operation results:

```

decomposition.DictionaryLearningBenchmark.peakmem\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

=============== ====== ======

--n\_jobs

--------------- -------------

fit\_algorithm 1 4

=============== ====== ======

lars 112M 126M

cd 106M 126M

=============== ====== ======

started: 2024-07-19 22:39:55, duration: 2.47m

decomposition.DictionaryLearningBenchmark.peakmem\_transform [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

=============== ======= =======

--n\_jobs

--------------- ---------------

fit\_algorithm 1 4

=============== ======= =======

lars 88.4M 90.4M

cd 88.4M 90.4M

=============== ======= =======

started: 2024-07-19 22:42:23, duration: 17.0s

```

3. Test case three: ensemble

Run the use case:

```

asv run -b ensemble HEAD^!

```

Operation results:

```

ensemble.GradientBoostingClassifierBenchmark.peakmem\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

================ =======

representation

---------------- -------

dense 94.8M

sparse 121M

================ =======

started: 2024-07-19 21:25:49, duration: 24.4s

ensemble.GradientBoostingClassifierBenchmark.peakmem\_predict [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

================ =======

representation

---------------- -------

dense 92.8M

sparse 102M

================ =======

started: 2024-07-19 21:26:14, duration: 1.27s

```

4. Test case 4: linear\_model

Run the use case:

```

asv run -b linear\_model HEAD^!

```

Operation results:

```

linear\_model.ElasticNetBenchmark.peakmem\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

================ ====== =======

--precompute

---------------- --------------

representation True False

================ ====== =======

dense 856M 1.21G

sparse 128M n/a

================ ====== =======

started: 2024-09-12 17:27:52, duration: 16.0s

linear\_model.ElasticNetBenchmark.peakmem\_predict [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

================ ====== =======

--precompute

---------------- --------------

representation True False

================ ====== =======

dense 496M 496M

sparse 101M n/a

================ ====== =======

started: 2024-09-12 17:28:08, duration: 1.63s

```

5. Test case 5: manifold

Run the use case:

```

asv run -b manifold HEAD^!

```

Operation results:

```

manifold.TSNEBenchmark.peakmem\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

============ =======

method

------------ -------

exact 92.7M

barnes\_hut 96M

============ =======

started: 2024-07-19 12:12:48, duration: 59.8s

manifold.TSNEBenchmark.time\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

============ ===========

method

------------ -----------

exact 34.2±0.2s

barnes\_hut 24.7±0.1s

============ ===========

started: 2024-07-19 12:13:48, duration: 3.94m

manifold.TSNEBenchmark.track\_test\_score [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

============ =====================

method

------------ ---------------------

exact 0.32347547445369784

barnes\_hut 0.7240576148033142

============ =====================

started: 2024-07-19 12:17:45, duration: 160ms

manifold.TSNEBenchmark.track\_train\_score [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

============ =====================

method

------------ ---------------------

exact 0.32347547445369784

barnes\_hut 0.7240576148033142

============ =====================

started: 2024-07-19 12:17:45, duration: 148ms

```

6. Test case 6: metrics

Run the use case:

```

asv run -b metrics HEAD^!

```

Operation results:

```

metrics.PairwiseDistancesBenchmark.peakmem\_pairwise\_distances [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

================= ============= ======== =======

-- n\_jobs

---------------------------- ---------------

representation metric 1 4

================= ============= ======== =======

dense cosine 666M 852M

dense euclidean 788M 871M

dense manhattan 258M 338M

dense correlation 251M 484M

sparse cosine 1.42G 1.33G

sparse euclidean 573M 793M

sparse manhattan 189M 210M

sparse correlation n/a n/a

================= ============= ======== =======

started: 2024-07-19 20:53:05, duration: 4.36m

metrics.PairwiseDistancesBenchmark.time\_pairwise\_distances [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

================ ============== ============= ============

-- n\_jobs

---------------------------- ------------------------

representation metric 1 4

================ ============== ============= ============

dense cosine 36.4±0.01s 20.3±0.2s

dense euclidean 1.02±0.01m 26.5±0.4s

dense manhattan 25.8±0.2s 8.16±0.3s

dense correlation 22.6±5s 9.66±0.5s

sparse cosine 33.0±3s 20.3±0.5s

sparse euclidean 10.2±0.01s 7.33±0.3s

spare manhattan 17.9±2s 4.25±0.5s

sparse correlation n/a n/a

================ ============== ============= ============

started: 2024-07-19 20:57:26, duration: 20.9m

```

7. Test case 7: model\_selection

Run the use case:

```

SKLBENCH\_NJOBS=[1,4] asv run -b model\_selection HEAD^!

```

Operation results:

```

model\_selection.CrossValidationBenchmark.peakmem\_crossval [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

======== ======

n\_jobs

-------- ------

1 212M

4 122M

======== ======

started: 2024-07-19 13:00:56, duration: 5.50m

model\_selection.CrossValidationBenchmark.time\_crossval [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

======== ============

n\_jobs

-------- ------------

1 4.01±0.01m

4 1.26±0.02m

======== ============

started: 2024-07-19 13:06:26, duration: 21.2m

model\_selection.CrossValidationBenchmark.track\_crossval [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

======== ====================

n\_jobs

-------- --------------------

1 0.9001555555555555

4 0.9001555555555555

======== ====================

started: 2024-07-19 13:27:40, duration: 5.92m

model\_selection.GridSearchBenchmark.peakmem\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

======== =======

n\_jobs

-------- -------

1 98.5M

4 96.5M

======== =======

started: 2024-07-19 14:12:45, duration: 30.5m

model\_selection.GridSearchBenchmark.peakmem\_predict [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

======== =======

n\_jobs

-------- -------

1 92.3M

4 92.3M

======== =======

started: 2024-07-19 14:43:13, duration: 1.02s

model\_selection.GridSearchBenchmark.time\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

======== ===========

n\_jobs

-------- -----------

1 26.7±1m

4 6.69±0.4m

======== ===========

started: 2024-07-19 14:43:14, duration: 2.21h

```

8. Test case eight: neighbors

Run the use case:

```

asv run -b neighbors HEAD^!

```

Operation results:

```

neighbors.KNeighborsClassifierBenchmark.peakmem\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

=========== ========== ========== ========== ===========

--dimension/n\_jobs

----------- -----------------------------------------

algorithm low / 1 low / 4 high / 1 high / 4

=========== ========== ========== ========== ===========

brute 81.2M 81.2M 84.6M 84.6M

kd\_tree 83.8M 83.8M 92.5M 92.5M

ball\_tree 83.8M 83.8M 92.2M 92.2M

=========== ========== ========== ========== ===========

started: 2024-07-19 12:23:30, duration: 3.80s

neighbors.KNeighborsClassifierBenchmark.peakmem\_predict [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

=========== ========== ========== ========== ===========

--dimension/n\_jobs

----------- -----------------------------------------

algorithm low / 1 low / 4 high / 1 high / 4

=========== ========== ========== ========== ===========

brute 87.7M 87.7M 91.4M 91.4M

kd\_tree 85M 83.9M 95.1M 95.1M

ball\_tree 84.8M 83.9M 94.7M 94.9M

=========== ========== ========== ========== ===========

started: 2024-07-19 12:23:34, duration: 4.96m

```

9. Test case nine: svm

Run the use case:

```

asv run -b svm HEAD^!

```

Operation results:

```

svm.SVCBenchmark.peakmem\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

========= ======

kernel

--------- ------

linear 281M

poly 281M

rbf 281M

sigmoid 281M

========= ======

started: 2024-07-20 13:06:57, duration: 20.4s

svm.SVCBenchmark.peakmem\_predict [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

========= ======

kernel

--------- ------

linear 209M

poly 209M

rbf 209M

sigmoid 209M

========= ======

started: 2024-07-20 13:07:18, duration: 20.9s

svm.SVCBenchmark.time\_fit [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

========= ============

kernel

--------- ------------

linear 5.36±2s

poly 3.89±0.01s

rbf 3.80±0s

sigmoid 4.32±1s

========= ============

started: 2024-07-20 13:07:39, duration: 2.25m

svm.SVCBenchmark.time\_predict [openeuler-riscv-4-2/virtualenv-py3.11-cython3.0.10-joblib1.3.2-numpy2.0.0-pandas2.2.2-scipy1.14.0-threadpoolctl3.2.0]

OK

========= ============

kernel

--------- ------------

linear 3.53±0.01s

poly 3.90±0.01s

rbf 7.95±0.03s

Sigmoid 3.63±0.01s

========= ============

started: 2024-07-20 13:09:54, duration: 2.23m

```

# benchmarks

> Test the use cases in the `scikit-learn/benchmarks` directory

## Install test dependencies

```

cd scikit-learn/benchmarks

pip install pytest matplotlib pandas memory\_profiler pooch seaborn plotly jinja2 pip install scikit-image

```

Special test cases require input parameters:

- bench\_20newsgroups.py

```

python bench\_20newsgroups.py -e {dummy,random\_forest,extra\_trees,logistic\_regression,naive\_bayes,adaboost}

```

- bench\_covertype.py

```

python bench\_covertype.py --classifiers {GBRT,ExtraTrees,RandomForest,CART,SGD,GaussianNB,liblinear,SAG}

```

- bench\_isotonic.py

```

python bench\_isotonic.py --iterations 100 --log\_min\_problem\_size 2 --log\_max\_problem\_size 10 --dataset perturbed\_logarithm

```

## Test items

1. Test case 1: bench\_20newsgroups\_dummy

Run the use case:

```

python bench\_20newsgroups.py -e dummy

```

Operation results:

```

Classifier train-time test-time Accuracy

--------------------------------------------

dummy 0.0051s 0.0033s 0.0530

```

2. Test case 2: bench\_20newsgroups\_random\_forest

Run the use case:

```

python bench\_20newsgroups.py -e random\_forest

```

Operation results:

```

Classifier train-time test-time Accuracy

--------------------------------------------

random\_forest 347.6202s 9.7250s 0.7641

```

3. Test case 3: bench\_20newsgroups\_extra\_trees

Run the use case:

```

python bench\_20newsgroups.py -e extra\_trees

```

Operation results:

```

Classifier train-time test-time Accuracy

--------------------------------------------

extra\_trees 235.4406s 3.0372s 0.7967

```

4. Test case 4: bench\_20newsgroups\_logistic\_regression

Run the use case:

```

python bench\_20newsgroups.py -e logistic\_regression

```

Operation results:

```

Classifier train-time test-time Accuracy

--------------------------------------------

logistic\_regression 139.3669s 0.2912s 0.7257

```

5. Test case 5: bench\_20newsgroups\_naive\_bayes

Run the use case:

```

python bench\_20newsgroups.py -e naive\_bayes

```

Operation results:

```

Classifier train-time test-time Accuracy

--------------------------------------------

naive\_bayes 0.7633s 0.2938s 0.7053

```

6. Test case 6: bench\_20newsgroups\_adaboost

Run the use case:

```

python bench\_20newsgroups.py -e adaboost

```

Operation results:

```

Classifier train-time test-time Accuracy

--------------------------------------------

adaboost 15.2203s 0.2506s 0.1216

```

7. Test case 7: bench\_covertype\_GBRT

Run the use case:

```

python bench\_covertype.py --classifiers GBRT

```

Operation results:

```

Classifier train-time test-time error-rate

--------------------------------------------

GBRT 1235.468s 0.9651s 0.3333

```

8. Test case eight: bench\_covertype\_ExtraTrees

Run the use case:

```

python bench\_covertype.py --classifiers ExtraTrees

```

Operation results:

```

Classifier train-time test-time error-rate

--------------------------------------------

ExtraTrees 94.6251s 2.6508s 0.0371

```

9. Test case nine: bench\_covertype\_RandomForest

Run the use case:

```

python bench\_covertype.py --classifiers RandomForest

```

Operation results:

```

Classifier train-time test-time error-rate

--------------------------------------------

RandomForest 85.9106s 1.7945s 0.0340

```

10. Test case 10: bench\_covertype\_CART

Run the use case:

```

python bench\_covertype.py --classifiers CART

```

Operation results:

```

Classifier train-time test-time error-rate

--------------------------------------------

CART 53.4808s 0.1557s 0.0426

```

11. Test case 11: bench\_covertype\_SGD

Run the use case:

```

python bench\_covertype.py --classifiers SGD

```

Operation results:

```

Classifier train-time test-time error-rate

--------------------------------------------

SGD 10.4110s 0.0429s 0.2315

```

12. Test case 12: bench\_covertype\_GaussianNB

Run the use case:

```

python bench\_covertype.py --classifiers GaussianNB

```

Operation results:

```

Classifier train-time test-time error-rate

--------------------------------------------

GaussianNB 1.0584s 0.1994s 0.4841

```

13. Test case 13: bench\_covertype\_liblinear

Run the use case:

```

python bench\_covertype.py --classifiers liblinear

```

Operation results:

```

sklearn.utils.\_param\_validation.InvalidParameterError: The 'loss' parameter of LinearSVC must be a str among {'hinge', 'squared\_hinge'}. Got 'l2' instead.

```

14. Test case 14: bench\_covertype\_SAG

Run the use case:

```

python bench\_covertype.py --classifiers SAG

```

Operation results:

```

Classifier train-time test-time error-rate

--------------------------------------------

SAG 3.5478s 0.0865s 0.2293

```

15. Test case 15: bench\_feature\_expansions

Run the use case:

```

python bench\_feature\_expansions.py

```

Operation results:

```

NULL

```

16. Test case 16: bench\_glm

Run the use case:

```

python bench\_glm.py

```

Operation results:

```

NULL

```

17. Test case 17: bench\_glmnet

Run the use case:

```

python bench\_glmnet.py

```

Operation results:

```

ModuleNotFoundError: No module named 'glmnet.elastic\_net'

ImportError: numpy.core.multiarray failed to import

```

18. Test case 18: bench\_hist\_gradient\_boosting

Run the use case:

```

python bench\_hist\_gradient\_boosting.py

```

Operation results:

```

Data size: 1000 samples train, 1000 samples test.

Fitting a sklearn model...

score: 0.9730

fit duration: 0.374s,

score duration: 0.013s,

Data size: 10000 samples train, 10000 samples test.

Fitting a sklearn model...

score: 0.9840

fit duration: 0.569s,

score duration: 0.097s,

Data size: 100000 samples train, 100000 samples test.

Fitting a sklearn model...

score: 0.9860

fit duration: 2.307s,

score duration: 0.731s,

Data size: 500000 samples train, 500000 samples test.

Fitting a sklearn model...

score: 0.9869

fit duration: 8.889s,

score duration: 3.662s,

Data size: 1000000 samples train, 1000000 samples test.

Fitting a sklearn model...

score: 0.9869

fit duration: 16.444s,

score duration: 7.456s,

```

19. Test case 19: bench\_hist\_gradient\_boosting\_adult

Run the use case:

```

python bench\_hist\_gradient\_boosting\_adult.py

```

Operation results:

```

Number of features: 14

Number of categorical features: 12

Number of numerical features: 2

Fitting a sklearn model...

fitted in 8.650s

predicted in 1.666s, ROC AUC: 0.9173, ACC: 0.8626

```

20. Test case 20: bench\_hist\_gradient\_boosting\_categorical\_only

Run the use case:

```

python bench\_hist\_gradient\_boosting\_categorical\_only.py

```

Operation results:

```

Number of features: 20

Number of samples: 10000

Fitting a sklearn model...

fitted in 7.081s

Predicted in 0.847s

```

21. Test case 21: bench\_hist\_gradient\_boosting\_higgsboson

Run the use case:

```

python bench\_hist\_gradient\_boosting\_higgsboson.py

```

Operation results:

```

dataset is too large

```

22. Test case 22: bench\_hist\_gradient\_boosting\_threading

Run the use case:

```

python bench\_hist\_gradient\_boosting\_threading.py

```

Operation results:

```

Data size: 1000000 samples train, 1000000 samples test.

n\_features: 100

n\_threads: 1

Fitting a sklearn model...

score: 0.9328

fit duration: 136.338s,

score duration: 11.568s,

n\_threads: 2

Fitting a sklearn model...

score: 0.9328

fit duration: 135.002s,

score duration: 11.490s,

n\_threads: 4

Fitting a sklearn model...

score: 0.9328

fit duration: 135.381s,

score duration: 11.509s,

n\_threads: 8

Fitting a sklearn model...

score: 0.9328

fit duration: 144.006s,

score duration: 13.891s,

n\_threads: 16

Fitting a sklearn model...

score: 0.9328

fit duration: 147.124s,

score duration: 12.931s,

n\_threads: 32

Fitting a sklearn model...

score: 0.9328

fit duration: 141.200s,

score duration: 12.112s,

n\_threads: 64

Fitting a sklearn model...

score: 0.9328

fit duration: 139.791s,

score duration: 19.330s,

n\_threads: 127

Fitting a sklearn model...

score: 0.9328

fit duration: 199.193s,

score duration: 18.410s,

```

23. Test case 23: bench\_isolation\_forest

Run the use case:

```

python bench\_isolation\_forest.py

```

Operation results:

```

http (AUC: 0.991, train\_time= 2.76s, test\_time= 0.44s)

smtp (AUC: 0.666, train\_time= 2.17s, test\_time= 0.26s)

SA (AUC: 0.965, train\_time= 3.72s, test\_time= 2.76s)

SF (AUC: 0.942, train\_time= 3.01s, test\_time= 1.94s)

shuttle (AUC: 0.997, train\_time= 2.63s, test\_time= 0.53s)

forestcover (AUC: 0.978, train\_time= 2.84s, test\_time= 4.97s)

```

24. Test case 24: bench\_isotonic.py

Run the use case:

```

python bench\_isotonic.py

```

Operation results:

```

100 0.00242953

1000 0.0035535200000000005

10000 0.00461494

100000 0.01059438

1000000 0.160536579999999998

10000000 0.7837066500000002

100000000 4.5084064

1000000000 68.08233364

```

25. Test case 25: bench\_kernel\_pca\_solvers\_time\_vs\_n\_components

Run the use case:

```

python bench\_kernel\_pca\_solvers\_time\_vs\_n\_components.py

```

Operation results:

```

NULL

```

26. Test case 26: bench\_kernel\_pca\_solvers\_time\_vs\_n\_samples

Run the use case:

```

python bench\_kernel\_pca\_solvers\_time\_vs\_n\_samples.py

```

Operation results:

```

NULL

```

27. Test case 27: bench\_lasso

Run the use case

```

python bench\_lasso.py

```

Operation results:

```

NULL

```

28. Test case 28: bench\_lof

Run the use case:

```

python bench\_lof.py

```

Operation results:

```

NULL

```

29. Test case 29: bench\_mnist

Run the use case:

```

python bench\_mnist.py

```

Operation results:

```

Classifier train-time test-time error-rate

------------------------------------------------------------

Nystroem-SVM 353.20s 13.18s 0.0207

ExtraTrees 199.62s 2.35s 0.0306

```

30. Test case 30: bench\_multilabel\_metrics

Run the use case:

```

python bench\_multilabel\_metrics.py

```

Operation results:

```

ImportError: cannot import name 'jaccard\_similarity\_score' from 'sklearn.metrics'

```

31. Test case 31: bench\_online\_ocsvm

Run the use case:

```

python bench\_online\_ocsvm.py

```

Operation results:

```

NULL

```

32. Test case 32: bench\_pca\_solvers

Run the use case:

```

python bench\_pca\_solvers.py

```

Operation results:

```

NULL

```

33. Test case 33: bench\_plot\_fastkmeans

Run the use case:

```

python bench\_plot\_fastkmeans.py

```

Operation results:

```

==============================

Iteration 025 of 025

==============================

K-Means

Speed: 0.783s

Inertia: 5944963513.01540

Fast K-Means

Speed: 3.423s

Inertia: 5970223605.933099

```

34. Test case 34: bench\_plot\_hierarchical

Run the use case:

```

python bench\_plot\_hierarchical.py

```

Operation results:

```

==============================

Iteration 032 of 032

n\_samples 15000; n\_features 50

==============================

Single

Speed: 15.138s

Average

Speed: 155.986s

Complete

Speed: 112.557s

Ward

Speed: 124.940s

```

35. Test case 35: bench\_plot\_incremental\_pca

Run the use case:

```

python bench\_plot\_incremental\_pca.py

```

Operation results:

```

NULL

```

36. Test case thirty-six: bench\_plot\_lasso\_path

Run the use case:

```

python bench\_plot\_lasso\_path.py

```

Operation results:

```

sklearn.utils.\_param\_validation.InvalidParameterError: The 'effective\_rank' parameter of make\_regression must be an int in the range [1, inf) or None. Got np.float64(1.0) instead.

```

37. Test case thirty-seven: bench\_plot\_neighbors

Run the use case:

```

bench\_plot\_neighbors.py

```

Operation results:

```

TypeError: legend() takes 0-2 positional arguments but 3 were given

```

38. Test case 38: bench\_plot\_nmf

Run the use case:

```

python bench\_plot\_nmf.py

```

Operation results:

```

TypeError: Memory.\_\_init\_\_() got an unexpected keyword argument 'cachedir'

```

39. Test case thirty-nine: bench\_plot\_omp\_lars

Run the use case:

```

python bench\_plot\_omp\_lars.py

```

Operation results:

```

====================

Iteration 025 of 025

====================

n\_samples: 5000

n\_features: 5000

benchmarking lars\_path (with Gram):9.553s

benchmarking lars\_path (without Gram):5.722s

benchmarking orthogonal\_mp (with Gram):7.963s

benchmarking orthogonal\_mp (without Gram):4.657s

```

40. Test case 40: bench\_plot\_parallel\_pairwise

Run the use case:

```

python bench\_plot\_parallel\_pairwise.py

```

Operation results:

```

NULL

```

41. Test case 41: bench\_plot\_polynomial\_kernel\_approximation

Run the use case:

```

python bench\_plot\_polynomial\_kernel\_approximation.py

```

Operation results:

```

NULL

```

42. Test case 42: bench\_plot\_randomized\_svd

Run the use case:

```

python bench\_plot\_randomized\_svd.py

```

Operation results:

```

ValueError: array must not contain infs or NaNs

```

43. Test case 43: bench\_plot\_svd

Run the use case:

```

python bench\_plot\_svd.py

```

Operation results:

```

TypeError: FigureBase.gca() got an unexpected keyword argument 'projection'

```

44. Test case 44: bench\_plot\_ward

Run the use case:

```

python bench\_plot\_ward.py

```

Operation results:

```

TypeError: 'numpy.float64' object cannot be interpreted as an integer

```

45. Test case 45: bench\_random\_projections

Run the use case:

```

python bench\_random\_projections.py

```

Operation results:

```

Transformer | fit | transform

-------------------------------|--------------|--------------

GaussianRandomProjection | 5.2222s | 21.2746s

SparseRandomProjection | 0.3937s | 1.4192s

```

46. Test case 46: bench\_rcv1\_logreg\_convergence

Run the use case:

```

python bench\_rcv1\_logreg\_convergence.py

```

Operation results:

```

TypeError: Memory.\_\_init\_\_() got an unexpected keyword argument 'cachedir'

```

47. Test case 47: bench\_saga

Run the use case:

```

python bench\_saga.py

```

Operation results:

```

AttributeError: module 'time' has no attribute 'clock'

```

48. Test case 48: bench\_sample\_without\_replacement

Run the use case:

```

python bench\_sample\_without\_replacement.py

```

Operation results:

```

NULL

```

49. Test case 49: bench\_sgd\_regression

Run the use case:

```

python bench\_sgd\_regression.py

```

Operation results:

```

NULL

```

50. Test case 50: bench\_sparsify

Run the use case:

```

python bench\_sparsify.py

```

Operation results:

```

Input data sparsity: 0.050000

True coef sparsity: 0.000100

Test data sparsity: 0.027400

Model sparsity: 0.000024

r^2 on test data (dense model): 0.232742

r^2 on test data (sparse model): 0.232742

```

51. Test case 51: bench\_text\_vectorizers

Run the use case:

```

python bench\_text\_vectorizers.py

```

Operation results:

```

========== Run time performance (sec) ===========

Computing the mean and the standard deviation of the run time over 3 runs...

vectorizer CountVectorizer HashingVectorizer TfidfVectorizer

analyzer ngram\_range

char (4, 4) 7.701 (+-0.442) 4.277 (+-0.759) 9.098 (+-1.105)

char\_wb (4, 4) 7.107 (+-0.831) 3.626 (+-0.230) 6.977 (+-0.203)

word (1, 1) 0.975 (+-0.019) 0.730 (+-0.024) 1.537 (+-0.134)

(1, 2) 4.565 (+-0.615) 1.994 (+-0.641) 6.559 (+-1.970)

=============== Memory usage (MB) ===============

vectorizer CountVectorizer HashingVectorizer TfidfVectorizer

analyzer ngram\_range

char (4, 4) 290.0 241.9 290.2

char\_wb (4, 4) 244.7 241.9 249.7

word (1, 1) 176.9 242.8 231.9

(1, 2) 269.1 242.8 269.4

```

52. Test case 52: bench\_tree

Run the use case:

```

python bench\_tree.py

```

Operation results:

```

NULL

```

53. Test case 53: bench\_tsne\_mnist

Run the use case:

```

python bench\_tsne\_mnist.py

```

Operation results:

```

PCA preprocessing down to 50 dimensions took 16.172s

Fitting sklearn TSNE on 100 samples took 1.990s in 849 iterations, nn accuracy: 0.740

Fitting sklearn TSNE on 500 samples took 9.218s in 999 iterations, nn accuracy: 0.670

Fitting sklearn TSNE on 1000 samples took 17.621s in 999 iterations, nn accuracy: 0.622

Fitting sklearn TSNE on 5000 samples took 121.799s in 999 iterations, nn accuracy: 0.547

```

54. Test case 54: plot\_tsne\_mnist

Run the use case:

```

python plot\_tsne\_mnist.py

```

Operation results:

```

ValueError: Object arrays cannot be loaded when allow\_pickle=False

```

# examples

> Test the examples in the `scikit-learn/examples` directory

## Install test dependencies

```

pip install pytest matplotlib pandas memory\_profiler pooch seaborn plotly jinja2 pip install scikit-image

```

The following use cases require special handling:

- approximate\_nearest\_neighbors.py

Need to install nmslib

The Non- Metric Space Library (NMSLIB) is an efficient cross-platform similarity search library and a toolkit for evaluating similarity search methods. The Pypi source currently provided by Python only supports Python 3.9. This library should not be maintained much in the future, but some Python open source projects are still using it.

```

// Clone nmslib source code

git clone https://github.com/nmslib/nmslib.git

cd nmslib/python\_bindings

// Install dependencies

pip install pybind11

pip install sphinx\_rtd\_theme

// Install

python setup.py build\_ext

python setup.py install

```

After installation, execute this use case:

```

// Install test dependencies

pip install pynndescent

// Run the test case

python approximate\_nearest\_neighbors.py

```

!!! nmslib can only be installed on the ARM platform, RISCV will report an error:

```

gcc: error: '-march=native': ISA string must begin with rv32 or rv64

error: command '/usr/lib64/ccache/gcc' failed with exit code 1

```

## Test items

### applications

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/applications

```

1. Test case 1: plot\_cyclical\_feature\_engineering

Run the use case:

```bash

python plot\_cyclical\_feature\_engineering.py

```

Operation results:

```

Categorical features: ['season', 'holiday', 'workingday', 'weather']

Mean model.n\_iter\_ = 100.0

Mean Absolute Error: 0.044 +/- 0.003

Root Mean Squared Error: 0.068 +/- 0.005

Mean model.alpha\_ = 2.7298221281347037

Mean Absolute Error: 0.142 +/- 0.014

Root Mean Squared Error: 0.184 +/- 0.020

Mean Absolute Error: 0.099 +/- 0.011

Root Mean Squared Error: 0.131 +/- 0.011

Mean Absolute Error: 0.125 +/- 0.014

Root Mean Squared Error: 0.166 +/- 0.020

Mean Absolute Error: 0.097 +/- 0.011

Root Mean Squared Error: 0.132 +/- 0.013

Mean Absolute Error: 0.078 +/- 0.009

Root Mean Squared Error: 0.104 +/- 0.009

Mean Absolute Error: 0.053 +/- 0.002

Root Mean Squared Error: 0.076 +/- 0.004

Mean Absolute Error: 0.082 +/- 0.006

Root Mean Squared Error: 0.111 +/- 0.011

```

2. Test case 2: plot\_digits\_denoising

Run the use case:

```bash

python plot\_digits\_denoising.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_face\_recognition

Run the use case:

```bash

python plot\_face\_recognition.py

```

Operation results:

```

Total dataset size:

n\_samples: 1288

n\_features: 1850

n\_classes: 7

Extracting the top 150 eigenfaces from 966 faces

done in 31.526s

Projecting the input data on the eigenfaces orthonormal basis

done in 0.376s

Fitting the classifier to the training set

done in 14.299s

Best estimator found by grid search:

SVC(C=41037.23718774337, class\_weight='balanced', gamma=0.004582101006952281)

Predicting people's names on the test set

done in 0.159s

precision recall f1-score support

Ariel Sharon 0.89 0.62 0.73 13

Colin Powell 0.75 0.90 0.82 60

Donald Rumsfeld 0.90 0.67 0.77 27

George W Bush 0.87 0.97 0.92 146

Gerhard Schroeder 1.00 0.80 0.89 25

Hugo Chavez 0.88 0.47 0.61 15

Tony Blair 0.93 0.78 0.85 36

accuracy 0.86 322

macro avg 0.89 0.74 0.80 322

weighted avg 0.87 0.86 0.86 322

```

4. Test case 4: plot\_model\_complexity\_influence

Run the use case:

```bash

python plot\_model\_complexity\_influence.py

```

Operation results:

```

Benchmarking SGDClassifier(alpha=0.001, l1\_ratio=0.25, loss='modified\_huber',

n\_iter\_no\_change=2, penalty='elasticnet', tol=0.1)

Complexity: 4948 | Hamming Loss (Misclassification Ratio): 0.2675 | Pred. Time: 0.083745s

Benchmarking SGDClassifier(alpha=0.001, l1\_ratio=0.5, loss='modified\_huber',

n\_iter\_no\_change=2, penalty='elasticnet', tol=0.1)

Complexity: 1847 | Hamming Loss (Misclassification Ratio): 0.3264 | Pred. Time: 0.059787s

Benchmarking SGDClassifier(alpha=0.001, l1\_ratio=0.75, loss='modified\_huber',

n\_iter\_no\_change=2, penalty='elasticnet', tol=0.1)

Complexity: 997 | Hamming Loss (Misclassification Ratio): 0.3383 | Pred. Time: 0.048674s

Benchmarking SGDClassifier(alpha=0.001, l1\_ratio=0.9, loss='modified\_huber',

n\_iter\_no\_change=2, penalty='elasticnet', tol=0.1)

Complexity: 802 | Hamming Loss (Misclassification Ratio): 0.3582 | Pred. Time: 0.044596s

Benchmarking NuSVR(C=1000.0, gamma=3.0517578125e-05, nu=0.05)

Complexity: 18 | MSE: 5558.7313 | Pred. Time: 0.000327s

Benchmarking NuSVR(C=1000.0, gamma=3.0517578125e-05, nu=0.1)

Complexity: 36 | MSE: 5289.8022 | Pred. Time: 0.000430s

Benchmarking NuSVR(C=1000.0, gamma=3.0517578125e-05, nu=0.2)

Complexity: 72 | MSE: 5193.8353 | Pred. Time: 0.000656s

Benchmarking NuSVR(C=1000.0, gamma=3.0517578125e-05, nu=0.35)

Complexity: 124 | MSE: 5131.3279 | Pred. Time: 0.000929s

Benchmarking NuSVR(C=1000.0, gamma=3.0517578125e-05)

Complexity: 178 | MSE: 5149.0779 | Pred. Time: 0.001245s

Benchmarking GradientBoostingRegressor(learning\_rate=0.05, max\_depth=2, n\_estimators=10)

Complexity: 10 | MSE: 4066.4812 | Pred. Time: 0.000362s

Benchmarking GradientBoostingRegressor(learning\_rate=0.05, max\_depth=2, n\_estimators=25)

Complexity: 25 | MSE: 3551.1723 | Pred. Time: 0.000379s

Benchmarking GradientBoostingRegressor(learning\_rate=0.05, max\_depth=2, n\_estimators=50)

Complexity: 50 | MSE: 3445.2171 | Pred. Time: 0.000426s

Benchmarking GradientBoostingRegressor(learning\_rate=0.05, max\_depth=2, n\_estimators=75)

Complexity: 75 | MSE: 3433.0358 | Pred. Time: 0.000488s

Benchmarking GradientBoostingRegressor(learning\_rate=0.05, max\_depth=2)

Complexity: 100 | MSE: 3456.0602 | Pred. Time: 0.000528s

```

5. Test case 5: plot\_outlier\_detection\_wine

Run the use case:

```bash

python plot\_outlier\_detection\_wine.py

```

Operation results:

```

NULL

```

6. Test case 6: plot\_out\_of\_core\_classification

Run the use case:

```bash

python plot\_out\_of\_core\_classification.py

```

Operation results:

```

NULL

```

7. Test case 7: plot\_prediction\_latency

Run the use case:

```bash

python plot\_prediction\_latency.py

```

Operation results:

```

NULL

```

8. Test case 8: plot\_species\_distribution\_modeling

Run the use case:

```bash

python plot\_species\_distribution\_modeling.py

```

Operation results:

```

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Modeling distribution of species 'bradypus variegatus'

- fit OneClassSVM ... done.

- plot coastlines from coverage

- predict species distribution

Area under the ROC curve : 0.895437

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Modeling distribution of species 'microryzomys minutus'

- fit OneClassSVM ... done.

- plot coastlines from coverage

- predict species distribution

Area under the ROC curve : 0.994986

time elapsed: 36.89s

```

9. Test case nine: plot\_stock\_market

Run the use case:

```bash

python plot\_stock\_market.py

```

Operation results:

```

NULL

```

10. Test case 10: plot\_time\_series\_lagged\_features

Run the use case:

```bash

python plot\_time\_series\_lagged\_features.py

```

Operation results:

```

NULL

```

11. Test case 11: plot\_tomography\_l1\_reconstruction

Run the use case:

```bash

python plot\_tomography\_l1\_reconstruction.py

```

Operation results:

```

NULL

```

12. Test case 12: plot\_topics\_extraction\_with\_nmf\_lda

Run the use case:

```bash

python plot\_topics\_extraction\_with\_nmf\_lda.py

```

Operation results:

```

Fitting the NMF model (Frobenius norm) with tf-idf features, n\_samples=2000 and n\_features=1000...

done in 0.160s.

Fitting the NMF model (generalized Kullback-Leibler divergence) with tf-idf features, n\_samples=2000 and n\_features=1000...

done in 1.808s.

Fitting the MiniBatchNMF model (Frobenius norm) with tf-idf features, n\_samples=2000 and n\_features=1000, batch\_size=128...

done in 0.208s.

Fitting the MiniBatchNMF model (generalized Kullback-Leibler divergence) with tf-idf features, n\_samples=2000 and n\_features=1000, batch\_size=128...

done in 0.451s.

Fitting LDA models with tf features, n\_samples=2000 and n\_features=1000...

done in 4.873s.

```

13. Test case 13: wikipedia\_principal\_eigenvector

Run the use case:

```bash

python wikipedia\_principal\_eigenvector.py

```

Operation results:

```

Computing the redirect map

Parsing the NT redirect file

[2024-07-31T12:40:46.429635] line: 00000000

[2024-07-31T12:40:53.955002] line: 01000000

[2024-07-31T12:41:01.598782] line: 02000000

[2024-07-31T12:41:09.323773] line: 03000000

[2024-07-31T12:41:16.781378] line: 04000000

Computing the transitive closure of the redirect relation

[2024-07-31T12:41:17.408794] line: 00000000

[2024-07-31T12:41:18.152476] line: 01000000

[2024-07-31T12:41:18.925669] line: 02000000

[2024-07-31T12:41:19.747393] line: 03000000

[2024-07-31T12:41:20.613174] line: 04000000

Computing the integer index map

[2024-07-31T12:41:20.694394] line: 00000000

[2024-07-31T12:41:28.516241] line: 01000000

[2024-07-31T12:41:36.295921] line: 02000000

[2024-07-31T12:41:44.063641] line: 03000000

[2024-07-31T12:41:51.863697] line: 04000000

Computing the adjacency matrix

Converting to CSR representation

CSR conversion done

Computing the principal singular vectors using randomized\_svd

done in 5.299s

Top wikipedia pages according to principal singular vectors

```

### bicluster

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/bicluster

```

1. Test case 1: plot\_bicluster\_newsgroups

Run the use case:

```bash

python plot\_bicluster\_newsgroups.py

```

Operation results:

```

Vectorizing...

Coclustering...

Done in 2.47s. V-measure: 0.4415

MiniBatchKMeans...

Done in 1.72s. V-measure: 0.3015

Best biclusters:

----------------

bicluster 0 : 8 documents, 6 words

categories : 100% talk.politics.mideast

words: cosmo, angmar, alfalfa, alpha, proline, benson

bicluster 1 : 1948 documents, 4325 words

categories : 23% talk.politics.guns, 18% talk.politics.misc, 17% sci.med

words: gun, guns, geb, banks, gordon, clinton, pitt, cdt, surrender, veal

bicluster 2: 1259 documents, 3534 words

categories : 27% soc.religion.christian, 25% talk.politics.mideast, 25% alt.atheism

words: god, jesus, christians, kent, sin, objective, belief, christ, faith, moral

bicluster 3: 775 documents, 1623 words

categories : 30% comp.windows.x, 25% comp.sys.ibm.pc.hardware, 20% comp.graphics

words: scsi, nada, ide, vga, esdi, isa, kth, s3, vlb, bmug

bicluster 4: 2180 documents, 2802 words

categories : 18% comp.sys.mac.hardware, 16% sci.electronics, 16% comp.sys.ibm.pc.hardware

words: voltage, shipping, circuit, receiver, processing, scope, mpce, analog, kolstad, umass

```

2. Test case 2: plot\_spectral\_biclustering

Run the use case:

```bash

python plot\_spectral\_biclustering.py

```

Operation results:

```

consensus score: 1.0

```

3. Test case 3: plot\_spectral\_coclustering

Run the use case:

```bash

python plot\_spectral\_coclustering.py

```

Operation results:

```

consensus score: 1.000

```

calibration

Enter the directory:

```

cd scikit-learn/examples/calibration

```

1. Test case 1: plot\_calibration\_curve

Run the use case:

```bash

python plot\_calibration\_curve.py

```

Operation results:

```

Brier score losses: (the smaller the better)

No calibration: 0.104

With isotonic calibration: 0.084

With sigmoid calibration: 0.109

```

2. Test case 2: plot\_calibration\_multiclass

Run the use case:

```bash

python plot\_calibration\_multiclass.py

```

Operation results:

```

NULL

```

3. Test case 3: plot\_calibration

Run the use case:

```bash

python plot\_calibration.py

```

Operation results:

```

Log-loss of

\* uncalibrated classifier: 1.327

\* calibrated classifier: 0.549

```

4. Test case 4: plot\_compare\_calibration

Run the use case:

```bash

python plot\_compare\_calibration.py

```

Operation results:

```

NULL

```

### classification

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/classification

```

1. Test case 1: plot\_classification\_probability

Run the use case:

```bash

python plot\_classification\_probability.py

```

Operation results:

```

Accuracy (train) for L1 logistic: 82.7%

Accuracy (train) for L2 logistic (Multinomial): 82.7%

Accuracy (train) for L2 logistic (OvR): 79.3%

Accuracy (train) for Linear SVC: 82.0%

Accuracy (train) for GPC: 82.7%

```

2. Test case 2: plot\_classifier\_comparison

Run the use case:

```bash

python plot\_classifier\_comparison.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_digits\_classification

Run the use case:

```bash

python plot\_digits\_classification.py

```

Operation results:

```

Classification report for classifier SVC(gamma=0.001):

precision recall f1-score support

0 1.00 0.99 0.99 88

1 0.99 0.97 0.98 91

2 0.99 0.99 0.99 86

3 0.98 0.87 0.92 91

4 0.99 0.96 0.97 92

5 0.95 0.97 0.96 91

6 0.99 0.99 0.99 91

7 0.96 0.99 0.97 89

8 0.94 1.00 0.97 88

9 0.93 0.98 0.95 92

accuracy 0.97 899

macro avg 0.97 0.97 0.97 899

weighted avg 0.97 0.97 0.97 899

Classification report rebuilt from confusion matrix:

precision recall f1-score support

0 1.00 0.99 0.99 88

1 0.99 0.97 0.98 91

2 0.99 0.99 0.99 86

3 0.98 0.87 0.92 91

4 0.99 0.96 0.97 92

5 0.95 0.97 0.96 91

6 0.99 0.99 0.99 91

7 0.96 0.99 0.97 89

8 0.94 1.00 0.97 88

9 0.93 0.98 0.95 92

accuracy 0.97 899

macro avg 0.97 0.97 0.97 899

weighted avg 0.97 0.97 0.97 899

```

4. Test case 4: plot\_lda

Run the use case:

```bash

python plot\_lda.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_lda\_qda

Run the use case:

```bash

python plot\_lda\_qda.py

```

Operation results:

```

NULL

```

### cluster

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/cluster

```

1. Test case 1: plot\_adjusted\_for\_chance\_measures

Run the use case:

```bash

python plot\_adjusted\_for\_chance\_measures.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_affinity\_propagation

Run the use case:

```bash

python plot\_affinity\_propagation.py

```

Operation results:

```

Estimated number of clusters: 3

Homogeneity: 0.872

Completeness: 0.872

V-measure: 0.872

Adjusted Rand Index: 0.912

Adjusted Mutual Information: 0.871

Silhouette Coefficient: 0.753

```

3. Test case three: plot\_agglomerative\_clustering\_metrics

Run the use case:

```bash

python plot\_agglomerative\_clustering\_metrics.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_agglomerative\_clustering

Run the use case:

```bash

python plot\_agglomerative\_clustering.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_agglomerative\_dendrogram

Run the use case:

```bash

python plot\_agglomerative\_dendrogram.py

```

Operation results:

```

NULL

```

6. Test case 6: plot\_birch\_vs\_minibatchkmeans

Run the use case:

```bash

python plot\_birch\_vs\_minibatchkmeans.py

```

Operation results:

```

BIRCH without global clustering as the final step took 3.33 seconds

n\_clusters : 158

BIRCH with global clustering as the final step took 3.24 seconds

n\_clusters : 100

Time taken to run MiniBatchKMeans 7.20 seconds

```

7. Test case 7: plot\_bisect\_kmeans

Run the use case:

```bash

python plot\_bisect\_kmeans.py

```

Operation results:

```

NULL

```

8. Test case eight: plot\_cluster\_comparison

Run the use case:

```bash

python plot\_cluster\_comparison.py

```

Operation results:

```

NULL

```

9. Test case nine: plot\_cluster\_iris

Run the use case:

```bash

python plot\_cluster\_iris.py

```

Operation results:

```

NULL

```

10. Test case 10: plot\_coin\_segmentation

Run the use case:

```bash

python plot\_coin\_segmentation.py

```

Operation results:

```

Spectral clustering: kmeans, 11.98s

Spectral clustering: discretize, 10.75s

Spectral clustering: cluster\_qr, 10.22s

```

11. Test case 11: plot\_coin\_ward\_segmentation

Run the use case:

```bash

python plot\_coin\_ward\_segmentation.py

```

Operation results:

```

Elapsed time: 1.885s

Number of pixels: 4697

Number of clusters: 27

```

12. Test case 12: plot\_color\_quantization

Run the use case:

```bash

python plot\_color\_quantization.py

```

Operation results:

```

Fitting model on a small sub-sample of the data

done in 0.259s.

Predicting color indices on the full image (k-means)

done in 0.130s.

Predicting color indices on the full image (random)

done in 0.546s.

```

13. Test case 13: plot\_dbscan

Run the use case:

```bash

python plot\_dbscan.py

```

Operation results:

```

Estimated number of clusters: 3

Estimated number of noise points: 18

Homogeneity: 0.953

Completeness: 0.883

V-measure: 0.917

Adjusted Rand Index: 0.952

Adjusted Mutual Information: 0.916

Silhouette Coefficient: 0.626

```

14. Test case 14: plot\_dict\_face\_patches

Run the use case:

```bash

python plot\_dict\_face\_patches.py

```

Operation results:

```

[MiniBatchKMeans] Reassigning 8 cluster centers.

[MiniBatchKMeans] Reassigning 5 cluster centers.

Partial fit of 100 out of 2400

[MiniBatchKMeans] Reassigning 3 cluster centers.

Partial fit of 200 out of 2400

[MiniBatchKMeans] Reassigning 1 cluster centers.

Partial fit of 300 out of 2400

[MiniBatchKMeans] Reassigning 3 cluster centers.

Partial fit of 400 out of 2400

Partial fit of 500 out of 2400

Partial fit of 600 out of 2400

Partial fit of 700 out of 2400

Partial fit of 800 out of 2400

Partial fit of 900 out of 2400

Partial fit of 1000 out of 2400

Partial fit of 1100 out of 2400

Partial fit of 1200 out of 2400

Partial fit of 1300 out of 2400

Partial fit of 1400 out of 2400

Partial fit of 1500 out of 2400

Partial fit of 1600 out of 2400

Partial fit of 1700 out of 2400

Partial fit of 1800 out of 2400

Partial fit of 1900 out of 2400

Partial fit of 2000 out of 2400

Partial fit of 2100 out of 2400

Partial fit of 2200 out of 2400

Partial fit of 2300 out of 2400

Partial fit of 2400 out of 2400

done in 12.83s.

```

15. Test case 15: plot\_digits\_agglomeration

Run the use case:

```bash

python plot\_digits\_agglomeration.py

```

Operation results:

```

NULL

```

16. Test case 16: plot\_digits\_linkage

Run the use case:

```bash

python plot\_digits\_linkage.py

```

Operation results:

```

Computing embedding

Done.

ward : 0.39s

average : 0.30s

complete : 0.29s

single : 0.11s

```

17. Test case 17: plot\_face\_compress

Run the use case:

```bash

python plot\_face\_compress.py

```

Operation results:

```

The dimension of the image is (768, 1024)

The data used to encode the image is of type uint8

The number of bytes taken in RAM is 786432

The number of bytes taken in RAM is 6291456

Compression ratio: 8.0

Type of the compressed image: float64

```

18. Test case 18: plot\_feature\_agglomeration\_vs\_univariate\_selection

Run the use case:

```bash

python plot\_feature\_agglomeration\_vs\_univariate\_selection.py

```

Operation results:

```

NULL

```

19. Test case 19: plot\_hdbscan

Run the use case:

```bash

python plot\_hdbscan.py

```

Operation results:

```

NULL

```

20. Test case 20: plot\_inductive\_clustering

Run the use case:

```bash

python plot\_inductive\_clustering.py

```

Operation results:

```

NULL

```

21. Test case 21: plot\_kmeans\_assumptions

Run the use case:

```bash

python plot\_kmeans\_assumptions.py

```

Operation results:

```

NULL

```

22. Test case 22: plot\_kmeans\_digits

Run the use case:

```bash

python plot\_kmeans\_digits.py

```

Operation results:

```

# digits: 10; # samples: 1797; # features 64

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

init time inertia homo compl v-meas ARI AMI silhouette

k-means++ 0.401s 69545 0.598 0.645 0.621 0.469 0.617 0.163

random 0.326s 69735 0.681 0.723 0.701 0.574 0.698 0.184

PCA-based 0.107s 69513 0.600 0.647 0.622 0.468 0.618 0.146

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

```

23. Test case 23: plot\_kmeans\_plusplus

Run the use case:

```bash

python plot\_kmeans\_plusplus.py

```

Operation results:

```

NULL

```

24. Test case 24: plot\_kmeans\_silhouette\_analysis

Run the use case:

```bash

python plot\_kmeans\_silhouette\_analysis.py

```

Operation results:

```

For n\_clusters = 2 The average silhouette\_score is : 0.7049787496083262

For n\_clusters = 3 The average silhouette\_score is : 0.5882004012129721

For n\_clusters = 4 The average silhouette\_score is : 0.6505186632729437

For n\_clusters = 5 The average silhouette\_score is : 0.561464362648773

For n\_clusters = 6 The average silhouette\_score is : 0.4857596147013469

```

25. Test case 25: plot\_kmeans\_stability\_low\_dim\_dense

Run the use case:

```bash

python plot\_kmeans\_stability\_low\_dim\_dense.py

```

Operation results:

```

NULL

```

26. Test case 26: plot\_linkage\_comparison

Run the use case:

```bash

python plot\_linkage\_comparison.py

```

Operation results:

```

NULL

```

27. Test case 27: plot\_mean\_shift

Run the use case:

```bash

python plot\_mean\_shift.py

```

Operation results:

```

NULL

```

28. Test case 28: plot\_mini\_batch\_kmeans

Run the use case:

```bash

python plot\_mini\_batch\_kmeans.py

```

Operation results:

```

NULL

```

29. Test case 29: plot\_optics

Run the use case:

```bash

python plot\_optics.py

```

Operation results:

```

NULL

```

30. Test case 30: plot\_segmentation\_toy

Run the use case:

```bash

python plot\_segmentation\_toy.py

```

Operation results:

```

NULL

```

31. Test case 31: plot\_ward\_structured\_vs\_unstructured

Run the use case:

```bash

python plot\_ward\_structured\_vs\_unstructured.py

```

Operation results:

```

Compute unstructured hierarchical clustering...

Elapsed time: 0.31s

Number of points: 1500

Compute structured hierarchical clustering...

Elapsed time: 0.42s

Number of points: 1500

```

### compose

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/compose

```

1. Test case 1: plot\_column\_transformer\_mixed\_types

Run the use case:

```bash

python plot\_column\_transformer\_mixed\_types.py

```

Operation results:

```

Model score: 0.798

<class 'pandas.core.frame.DataFrame'>

Index: 1047 entries, 1118 to 684

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 embarked 1045 non-null category

1 sex 1047 non-null category

2 pclass 1047 non-null int64

3 age 841 non-null float64

4 fare 1046 non-null float64

dtypes: category(2), float64(2), int64(1)

memory usage: 35.0 KB

Model score: 0.798

Best params:

{'preprocessor\_\_num\_\_imputer\_\_strategy': 'mean', 'preprocessor\_\_cat\_\_selector\_\_percentile': 30, 'classifier\_\_C': 100}

Internal CV score: 0.786

accuracy of the best model from randomized search: 0.798

```

2. Test case 2: plot\_column\_transformer

Run the use case:

```bash

python plot\_column\_transformer.py

```

Operation results:

```

precision recall f1-score support

0 0.84 0.88 0.86 396

1 0.87 0.84 0.85 394

accuracy 0.86 790

macro avg 0.86 0.86 0.86 790

weighted avg 0.86 0.86 0.86 790

```

3. Test case 3: plot\_compare\_reduction

Run the use case:

```bash

python plot\_compare\_reduction.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_digits\_pipe

Run the use case:

```bash

python plot\_digits\_pipe.py

```

Operation results:

```

No module named 'polars'

```

5. Test case 5: plot\_feature\_union

Run the use case:

```bash

python plot\_feature\_union.py

```

Operation results:

```

NULL

```

6. Test case six: plot\_transformed\_target

Run the use case:

```bash

python plot\_transformed\_target.py

```

Operation results:

```

NULL

```

covariance

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/covariance

```

1. Test case 1: plot\_covariance\_estimation

Run the use case:

```bash

python plot\_covariance\_estimation.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_lw\_vs\_oas

Run the use case:

```bash

python plot\_lw\_vs\_oas.py

```

Operation results:

```

NULL

```

3. Test case 3: plot\_mahalanobis\_distances

Run the use case:

```bash

python plot\_mahalanobis\_distances.py

```

Operation results:

```

Estimated covariance matrix:

MCD (Robust):

[[ 3.26253567e+00 -3.06695631e-03]

[-3.06695631e-03 1.22747343e+00]]

MLE:

[[ 3.23773583 -0.24640578]

[-0.24640578 7.51963999]]

```

4. Test case 4: plot\_robust\_vs\_empirical\_covariance

Run the use case:

```bash

python plot\_robust\_vs\_empirical\_covariance.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_sparse\_cov

Run the use case:

```bash

python plot\_sparse\_cov.py

```

Operation results:

```

NULL

```

### cross\_decomposition

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/cross\_decomposition

```

1. Test case 1: plot\_compare\_cross\_decomposition

Run the use case:

```bash

python plot\_compare\_cross\_decomposition.py

```

Operation results:

```

Corr(X)

[[1. 0.53 0.03 0.08]

[0.53 1. 0.02 0.06]

[0.03 0.02 1. 0.49]

[0.08 0.06 0.49 1. ]]

Corr(Y)

[[ 1. 0.49 0.01 0.02]

[ 0.49 1. -0.01 -0.03 ]

[ 0.01 -0.01 1. 0.48 ]

[ 0.02 -0.03 0.48 1. ]]

True B (such that: Y = XB + Err)

[[1 1 1]

[2 2 2]

[0 0 0]

[0 0 0]

[0 0 0]

[0 0 0]

[0 0 0]

[0 0 0]

[0 0 0]

[0 0 0]]

Estimated B

[[ 1. 2. -0. 0. -0. 0. -0. 0. -0.1 0. ]

[ 1. 2. -0. 0. 0. -0. 0. 0. 0. 0. ]

[ 1. 2. -0. 0.1 -0. -0.1 0.1 0. -0. 0. ]]

Estimated betas

[[ 1. 2. -0. 0. -0. -0.1 0. 0. 0. 0. ]]

```

2. Test case 2: plot\_pcr\_vs\_pls

Run the use case:

```bash

python plot\_pcr\_vs\_pls.py

```

Operation results:

```

PCR r-squared -0.026

PLS r-squared 0.658

PCR r-squared with 2 components 0.673

```

### datasets

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/datasets

```

1. Test case 1: plot\_digits\_last\_image

Run the use case:

```bash

python plot\_digits\_last\_image.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_iris\_dataset

Run the use case:

```bash

python plot\_iris\_dataset.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_random\_dataset

Run the use case:

```bash

python plot\_random\_dataset.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_random\_multilabel\_dataset

Run the use case:

```bash

python plot\_random\_multilabel\_dataset.py

```

Operation results:

```

The data was generated from (random\_state=561):

Class P(C) P(w0|C) P(w1|C)

red 0.48 0.28 0.72

blue 0.47 0.40 0.60

yellow 0.05 0.10 0.90

```

### decomposition

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/decomposition

```

1. Test case 1: plot\_faces\_decomposition

Run the use case:

```bash

python plot\_faces\_decomposition.py

```

Operation results:

```

Dataset consists of 400 faces

```

2. Test case 2: plot\_ica\_blind\_source\_separation

Run the use case:

```bash

python plot\_ica\_blind\_source\_separation.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_ica\_vs\_pca

Run the use case:

```bash

python plot\_ica\_vs\_pca.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_image\_denoising

Run the use case:

```bash

python plot\_image\_denoising.py

```

Operation results:

```

Distorting image...

Extracting reference patches...

22692 patches extracted in 0.08s.

Learning the dictionary...

1.0 iterations / 111 steps in 118.20.

Extracting noisy patches...

done in 0.04s.

Orthogonal Matching Pursuit

1 atom...

done in 4.08s.

Orthogonal Matching Pursuit

2 atoms...

done in 8.15s.

Least-angle regression

4 atoms...

done in 67.30s.

Thresholding

alpha=0.1...

done in 0.58s.

```

5. Test case 5: plot\_incremental\_pca

Run the use case:

```bash

python plot\_incremental\_pca.py

```

Operation results:

```

NULL

```

6. Test case six: plot\_kernel\_pca

Run the use case:

```bash

python plot\_kernel\_pca.py

```

Operation results:

```

NULL

```

7. Test case 7: plot\_pca\_iris

Run the use case:

```bash

python plot\_pca\_iris.py

```

Operation results:

```

NULL

```

8. Test case eight: plot\_pca\_vs\_fa\_model\_selection

Run the use case:

```bash

python plot\_pca\_vs\_fa\_model\_selection.py

```

Operation results:

```

best n\_components by PCA CV = 5

best n\_components by FactorAnalysis CV = 5

best n\_components by PCA MLE = 5

best n\_components by PCA CV = 20

best n\_components by FactorAnalysis CV = 5

best n\_components by PCA MLE = 18

```

9. Test case nine: plot\_pca\_vs\_lda

Run the use case:

```bash

python plot\_pca\_vs\_lda.py

```

Operation results:

```

explained variance ratio (first two components): [0.92461872 0.05306648]

```

10. Test case 10: plot\_sparse\_coding

Run the use case:

```bash

python plot\_sparse\_coding.py

```

Operation results:

```

NULL

```

11. Test case 11: plot\_varimax\_fa

Run the use case:

```bash

python plot\_varimax\_fa.py

```

Operation results:

```

PCA:

[[ 0.52106591 0.37741762]

[-0.26934744 0.92329566]

[ 0.5804131 0.02449161 ]

[ 0.56485654 0.06694199 ] ]

Unrotated FA:

[[ 0.88096009 -0.4472869 ]

[-0.41691605 -0.55390036]

[ 0.99918858 0.01915283 ]

[ 0.96228895 0.05840206 ] ]

Varimax FA:

[[ 0.98633022 -0.05752333]

[-0.16052385 -0.67443065]

[ 0.90809432 0.41726413 ]

[ 0.85857475 0.43847489 ] ]

```

### developing\_estimators

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/developing\_estimators

```

1. Test case 1: sklearn\_is\_fitted

Run the use case:

```bash

python sklearn\_is\_fitted.py

```

Operation results:

```

NULL

```

### ensemble

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/ensemble

```

1. Test case 1: plot\_adaboost\_multiclass

Run the use case:

```bash

python plot\_adaboost\_multiclass.py

```

Operation results:

```

DecisionTreeClassifier's misclassification\_error: 0.475

DummyClassifier's misclassification\_error: 0.692

```

2. Test case 2: plot\_adaboost\_regression

Run the use case:

```bash

python plot\_adaboost\_regression.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_adaboost\_twoclass

Run the use case:

```bash

python plot\_adaboost\_twoclass.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_bias\_variance

Run the use case:

```bash

python plot\_bias\_variance.py

```

Operation results:

```

Tree: 0.0255 (error) = 0.0003 (bias^2) + 0.0152 (var) + 0.0098 (noise)

Bagging(Tree): 0.0196 (error) = 0.0004 (bias^2) + 0.0092 (var) + 0.0098 (noise)

```

5. Test case 5: plot\_ensemble\_oob

Run the use case:

```bash

python plot\_ensemble\_oob.py

```

Operation results:

```

NULL

```

6. Test case 6: plot\_feature\_transformation

Run the use case:

```bash

python plot\_feature\_transformation.py

```

Operation results:

```

NULL

```

7. Test case 7: plot\_forest\_hist\_grad\_boosting\_comparison

Run the use case:

```bash

python plot\_forest\_hist\_grad\_boosting\_comparison.py

```

Operation results:

```

The dataset consists of 20640 samples and 8 features

Number of physical cores: 1

```

8. Test case 8: plot\_forest\_importances\_faces

Run the use case:

```bash

python plot\_forest\_importances\_faces.py

```

Operation results:

```

Elapsed time to compute the importance: 4.661 seconds

```

9. Test case nine: plot\_forest\_importances

Run the use case:

```bash

python plot\_forest\_importances.py

```

Operation results:

```

Elapsed time to compute the importance: 0.361 seconds

Elapsed time to compute the importance: 10.222 seconds

```

10. Test case 10: plot\_forest\_iris

Run the use case:

```bash

python plot\_forest\_iris.py

```

Operation results:

```

DecisionTree with features [0, 1] has a score of 0.9266666666666666

RandomForest with 30 estimators with features [0, 1] has a score of 0.9266666666666666

ExtraTrees with 30 estimators with features [0, 1] has a score of 0.9266666666666666

AdaBoost with 30 estimators with features [0, 1] has a score of 0.82

DecisionTree with features [0, 2] has a score of 0.9933333333333333

RandomForest with 30 estimators with features [0, 2] has a score of 0.9933333333333333

ExtraTrees with 30 estimators with features [0, 2] has a score of 0.9933333333333333

AdaBoost with 30 estimators with features [0, 2] has a score of 0.9933333333333333

DecisionTree with features [2, 3] has a score of 0.9933333333333333

RandomForest with 30 estimators with features [2, 3] has a score of 0.9933333333333333

ExtraTrees with 30 estimators with features [2, 3] has a score of 0.9933333333333333

AdaBoost with 30 estimators with features [2, 3] has a score of 0.9866666666666667

```

11. Test case 11: plot\_gradient\_boosting\_categorical

Run the use case:

```bash

python plot\_gradient\_boosting\_categorical.py

```

Operation results:

```

Number of samples: 1460

Number of features: 20

Number of categorical features: 10

Number of numerical features: 10

```

12. Test case 12: plot\_gradient\_boosting\_early\_stopping

Run the use case:

```bash

python plot\_gradient\_boosting\_early\_stopping.py

```

Operation results:

```

NULL

```

13. Test case 13: plot\_gradient\_boosting\_oob

Run the use case:

```bash

python plot\_gradient\_boosting\_oob.py

```

Operation results:

```

Accuracy: 0.6840

```

14. Test case 14: plot\_gradient\_boosting\_quantile

Run the use case:

```bash

python plot\_gradient\_boosting\_quantile.py

```

Operation results:

```

{'learning\_rate': 0.2,

'max\_depth': 2,

'min\_samples\_leaf': 20,

'min\_samples\_split': 10,

'n\_estimators': 150}

{'learning\_rate': 0.05,

'max\_depth': 2,

'min\_samples\_leaf': 5,

'min\_samples\_split': 20,

'n\_estimators': 150}

```

15. Test case 15: plot\_gradient\_boosting\_regression

Run the use case:

```bash

python plot\_gradient\_boosting\_regression.py

```

Operation results:

```

NULL

```

16. Test case 16: plot\_gradient\_boosting\_regularization

Run the use case:

```bash

python plot\_gradient\_boosting\_regularization.py

```

Operation results:

```

NULL

```

17. Test case 17: plot\_hgbt\_regression

Run the use case:

```bash

python plot\_hgbt\_regression.py

```

Operation results:

```

Training sample size: 16531

Test sample size: 11021

Number of features: 7

RMSE without constraints = 0.103 +/- 0.030

RMSE with constraints = 0.107 +/- 0.034

```

18. Test case 18: plot\_isolation\_forest

Run the use case:

```bash

python plot\_isolation\_forest.py

```

Operation results:

```

Training sample size: 16531

Test sample size: 11021

Number of features: 7

RMSE without constraints = 0.103 +/- 0.030

RMSE with constraints = 0.107 +/- 0.034

```

19. Test case 19: plot\_monotonic\_constraints

Run the use case:

```bash

python plot\_monotonic\_constraints.py

```

Operation results:

```

NULL

```

20. Test case 20: plot\_random\_forest\_embedding

Run the use case:

```bash

python plot\_random\_forest\_embedding.py

```

Operation results:

```

NULL

```

21. Test case 21: plot\_random\_forest\_regression\_multioutput

Run the use case:

```bash

python plot\_random\_forest\_regression\_multioutput.py

```

Operation results:

```

NULL

```

22. Test case 22: plot\_stack\_predictors

Run the use case:

```bash

python plot\_stack\_predictors.py

```

Operation results:

```

NULL

```

23. Test case 23: plot\_voting\_decision\_regions

Run the use case:

```bash

python plot\_voting\_decision\_regions.py

```

Operation results:

```

NULL

```

24. Test case 24: plot\_voting\_probas

Run the use case:

```bash

python plot\_voting\_probas.py

```

Operation results:

```

NULL

```

25. Test case 25: plot\_voting\_regressor

Run the use case:

```bash

python plot\_voting\_regressor.py

```

Operation results:

```

NULL

```

### exercises

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/exercises

```

1. Test case 1: plot\_cv\_diabetes

Run the use case:

```bash

python plot\_cv\_diabetes.py

```

Operation results:

```

subsets of the data:

[fold 0] alpha: 0.05968, score: 0.54209

[fold 1] alpha: 0.04520, score: 0.15521

[fold 2] alpha: 0.07880, score: 0.45192

```

2. Test case 2: plot\_digits\_classification\_exercise

Run the use case:

```bash

python plot\_digits\_classification\_exercise.py

```

Operation results:

```

KNN score: 0.961111

LogisticRegression score: 0.933333

```

3. Test case three: plot\_iris\_exercise

Run the use case:

```bash

python plot\_iris\_exercise.py

```

Operation results:

```

NULL

```

### feature\_selection

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/feature\_selection

```

1. Test case 1: plot\_feature\_selection\_pipeline

Run the use case:

```bash

python plot\_feature\_selection\_pipeline.py

```

Operation results:

```

precision recall f1-score support

0 0.92 0.80 0.86 15

1 0.75 0.90 0.82 10

accuracy 0.84 25

macro avg 0.84 0.85 0.84 25

weighted avg 0.85 0.84 0.84 25

```

2. Test case 2: plot\_feature\_selection

Run the use case:

```bash

python plot\_feature\_selection.py

```

Operation results:

```

Classification accuracy without selecting features: 0.789

Classification accuracy after univariate feature selection: 0.868

```

3. Test case three: plot\_f\_test\_vs\_mi

Run the use case:

```bash

python plot\_f\_test\_vs\_mi.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_rfe\_digits

Run the use case:

```bash

python plot\_rfe\_digits.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_rfe\_with\_cross\_validation

Run the use case:

```bash

python plot\_rfe\_with\_cross\_validation.py

```

Operation results:

```

NULL

```

6. Test case 6: plot\_select\_from\_model\_diabetes

Run the use case:

```bash

python plot\_select\_from\_model\_diabetes.py

```

Operation results:

```

tol: -0.01

Features selected: ['worst perimeter']

ROC AUC score: 0.975

Done in 65.592s

tol: -0.001

Features selected: ['radius error' 'fractal dimension error' 'worst texture'

'worst perimeter' 'worst concave points']

ROC AUC score: 0.997

Done in 59.934s

tol: -0.0001

Features selected: ['mean compactness' 'mean concavity' 'mean concave points' 'radius error'

'area error' 'concave points error' 'symmetry error'

'fractal dimension error' 'worst texture' 'worst perimeter' 'worst area'

'worst concave points' 'worst symmetry']

ROC AUC score: 0.998

Done in 70.645s

```

### gaussian\_process

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/gaussian\_process

```

1. Test case 1: plot\_compare\_gpr\_krr

Run the use case:

```bash

python plot\_compare\_gpr\_krr.py

```

Operation results:

```

Fitting KernelRidge with default kernel: 0.005 seconds

Time for KernelRidge fitting: 23.839 seconds

Time for KernelRidge prediction: 0.009 seconds

Time for GaussianProcessRegressor fitting: 0.178 seconds

Time for GaussianProcessRegressor predict: 0.012 seconds

```

2. Test case 2: plot\_gpc\_iris

Run the use case:

```bash

python plot\_gpc\_iris.py

```

Operation results:

```

NULL

```

3. Test case 3: plot\_gpc\_isoprobability

Run the use case:

```bash

python plot\_gpc\_isoprobability.py

```

Operation results:

```

Learned kernel: 0.0256\*\*2 \* DotProduct(sigma\_0=5.72) \*\* 2

```

4. Test case 4: plot\_gpc

Run the use case:

```bash

python plot\_gpc.py

```

Operation results:

```

Log Marginal Likelihood (initial): -17.598

Log Marginal Likelihood (optimized): -3.875

Accuracy: 1.000 (initial) 1.000 (optimized)

Log-loss: 0.214 (initial) 0.319 (optimized)

```

5. Test case 5: plot\_gpc\_xor

Run the use case:

```bash

python plot\_gpc\_xor.py

```

Operation results:

```

NULL

```

6. Test case six: plot\_gpr\_co2

Run the use case:

```bash

python plot\_gpr\_co2.py

```

Operation results:

```

ModuleNotFoundError: No module named 'polars'

```

7. Test case 7: plot\_gpr\_noisy

Run the use case:

```bash

python plot\_gpr\_noisy.py

```

Operation results:

```

NULL

```

8. Test case eight: plot\_gpr\_noisy\_targets

Run the use case:

```bash

python plot\_gpr\_noisy\_targets.py

```

Operation results:

```

NULL

```

9. Test case nine: plot\_gpr\_on\_structured\_data

Run the use case:

```bash

python plot\_gpr\_on\_structured\_data.py

```

Operation results:

```

NULL

```

10. Test case 10: plot\_gpr\_prior\_posterior

Run the use case:

```bash

python plot\_gpr\_prior\_posterior.py

```

Operation results:

```

Kernel parameters before fit:

1\*\*2 \* RBF(length\_scale=1))

Kernel parameters after fit:

0.594\*\*2 \*RBF(length\_scale=0.279)

Log-likelihood: -0.067

Kernel parameters before fit:

1\*\*2 \* RationalQuadratic(alpha=0.1, length\_scale=1))

Kernel parameters after fit:

0.594\*\*2 \* RationalQuadratic(alpha=2.91e+05, length\_scale=0.279)

Log-likelihood: -0.067

Kernel parameters before fit:

1\*\*2 \* ExpSineSquared(length\_scale=1, periodicity=3))

Kernel parameters after fit:

0.799\*\*2 \* ExpSineSquared(length\_scale=0.791, periodicity=2.87)

Log-likelihood: 3.394

Kernel parameters before fit:

0.316\*\*2 \* DotProduct(sigma\_0=1) \*\* 2)

Kernel parameters after fit:

0.656\*\*2 \* DotProduct(sigma\_0=2.07) \*\* 2

Log-likelihood: -7959003586.436

Kernel parameters before fit:

1\*\*2 \*Matern(length\_scale=1, nu=1.5))

Kernel parameters after fit:

0.609\*\*2 \* Matern(length\_scale=0.484, nu=1.5)

Log-likelihood: -1.185

```

### impute

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/impute

```

1. Test case 1: plot\_iterative\_imputer\_variants\_comparison

Run the use case:

```bash

python plot\_iterative\_imputer\_variants\_comparison.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_missing\_values

Run the use case:

```bash

python plot\_missing\_values.py

```

Operation results:

```

NULL

```

### inspection

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/inspection

```

1. Test case 1: plot\_causal\_interpretation

Run the use case:

```bash

python plot\_causal\_interpretation.py

```

Operation results:

```

R2 score with ability: 0.975

R2 score without ability: 0.968

```

2. Test case 2: plot\_linear\_model\_coefficient\_interpretation

Run the use case:

```bash

python plot\_linear\_model\_coefficient\_interpretation.py

```

Operation results:

```

RangeIndex: 534 entries, 0 to 533

Data columns (total 10 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 EDUCATION 534 non-null int64

1 SOUTH 534 non-null category

2 SEX 534 non-null category

3 EXPERIENCE 534 non-null int64

4 UNION 534 non-null category

5 AGE 534 non-null int64

6 RACE 534 non-null category

7 OCCUPATION 534 non-null categories

8 SECTOR 534 non-null category

9 MARR 534 non-null category

dtypes: category(7), int64(3)

memory usage: 17.3 KB

```

3. Test case three: plot\_partial\_dependence

Run the use case:

```bash

python plot\_partial\_dependence.py

```

Operation results:

```

Index: 1729 entries, 0 to 8640

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 season 1729 non-null category

1 month 1729 non-null int64

2 hours 1729 non-null int64

3 holiday 1729 non-null category

4 weekday 1729 non-null int64

5 workingday 1729 non-null category

6 weather 1729 non-null category

7 temp 1729 non-null float64

8 feel\_temp 1729 non-null float64

9 humidity 1729 non-null float64

10 windspeed 1729 non-null float64

dtypes: category(4), float64(4), int64(3)

memory usage: 115.4 KB

Training MLPRegressor...

done in 10.462s

Test R2 score: 0.61

Computing partial dependence plots...

done in 13.123s

Training HistGradientBoostingRegressor...

done in 2.250s

Test R2 score: 0.62

Computing partial dependence plots...

done in 44.670s

Computing partial dependence plots and individual conditional expectation...

done in 19.306s

Test R2 score: 0.38

Computing partial dependence plots...

done in 307.837s

Computing partial dependence plots...

done in 263.352s

Computing partial dependence plots...

done in 11.717s

```

4. Test case 4: plot\_permutation\_importance\_multicollinear

Run the use case:

```bash

python plot\_permutation\_importance\_multicollinear.py

```

Operation results:

```

Baseline accuracy on test data: 0.97

Baseline accuracy on test data with features removed: 0.97

```

5. Test case 5: plot\_permutation\_importance

Run the use case:

```bash

python plot\_permutation\_importance.py

```

Operation results:

```

RF train accuracy: 1.000

RF test accuracy: 0.814

RF train accuracy: 0.810

RF test accuracy: 0.832

```

### kernel\_approximation

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/kernel\_approximation

```

1. Test case 1: plot\_scalable\_poly\_kernels

Run the use case:

```bash

python plot\_scalable\_poly\_kernels.py

```

Operation results:

```

Linear SVM score on raw features: 75.62%

Linear SVM score on 250 PolynomialCountSketch features: 75.60%

Linear SVM score on 500 PolynomialCountSketch features: 77.58%

Linear SVM score on 1000 PolynomialCountSketch features: 78.03%

Linear SVM score on 2000 PolynomialCountSketch features: 78.26%

Kernel-SVM score on raw features: 79.77%

```

### linear\_model

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/linear\_model

```

1. Test case 1: plot\_ard

Run the use case:

```bash

python plot\_ard.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_bayesian\_ridge\_curvefit

Run the use case:

```bash

python plot\_bayesian\_ridge\_curvefit.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_elastic\_net\_precomputed\_gram\_matrix\_with\_weighted\_samples

Run the use case:

```bash

python plot\_elastic\_net\_precomputed\_gram\_matrix\_with\_weighted\_samples.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_huber\_vs\_ridge

Run the use case:

```bash

python plot\_huber\_vs\_ridge.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_iris\_logistic

Run the use case:

```bash

python plot\_iris\_logistic.py

```

Operation results:

```

NULL

```

6. Test case 6: plot\_lasso\_and\_elasticnet

Run the use case:

```bash

python plot\_lasso\_and\_elasticnet.py

```

Operation results:

```

Lasso fit done in 0.008s

Lasso r^2 on test data: 0.480

ARD fit done in 0.476s

ARD r^2 on test data : 0.542

ElasticNet fit done in 0.007s

ElasticNet r^2 on test data: 0.636

```

7. Test case 7: plot\_lasso\_coordinate\_descent\_path

Run the use case:

```bash

python plot\_lasso\_coordinate\_descent\_path.py

```

Operation results:

```

NULL

```

8. Test case eight: plot\_lasso\_dense\_vs\_sparse\_data

Run the use case:

```bash

python plot\_lasso\_dense\_vs\_sparse\_data.py

```

Operation results:

```

Sparse Lasso done in 0.293s

Dense Lasso done in 0.177s

Distance between coefficients: 1.13e-13

Matrix density: 0.626%

Sparse Lasso done in 0.357s

Dense Lasso done in 6.245s

Distance between coefficients: 2.48e-11

```

9. Test case nine: plot\_lasso\_lars\_ic

Run the use case:

```bash

python plot\_lasso\_lars\_ic.py

```

Operation results:

```

NULL

```

10. Test case 10: plot\_lasso\_lars

Run the use case:

```bash

python plot\_lasso\_lars.py

```

Operation results:

```

NULL

```

11. Test case 11: plot\_lasso\_model\_selection

Run the use case:

```bash

python plot\_lasso\_model\_selection.py

```

Operation results:

```

NULL

```

12. Test case 12: plot\_logistic\_l1\_l2\_sparsity

Run the use case:

```bash

python plot\_logistic\_l1\_l2\_sparsity.py

```

Operation results:

```

C=1.00

Sparsity with L1 penalty: 6.25%

Sparsity with Elastic-Net penalty: 4.69%

Sparsity with L2 penalty: 4.69%

Score with L1 penalty: 0.90

Score with Elastic-Net penalty: 0.90

Score with L2 penalty: 0.90

C=0.10

Sparsity with L1 penalty: 31.25%

Sparsity with Elastic-Net penalty: 12.50%

Sparsity with L2 penalty: 4.69%

Score with L1 penalty: 0.90

Score with Elastic-Net penalty: 0.90

Score with L2 penalty: 0.90

C=0.01

Sparsity with L1 penalty: 84.38%

Sparsity with Elastic-Net penalty: 68.75%

Sparsity with L2 penalty: 4.69%

Score with L1 penalty: 0.86

Score with Elastic-Net penalty: 0.88

Score with L2 penalty: 0.89

```

13. Test case 13: plot\_logistic\_multinomial

Run the use case:

```bash

python plot\_logistic\_multinomial.py

```

Operation results:

```

training score: 0.995 (multinomial)

training score : 0.976 (ovr)

```

14. Test case 14: plot\_logistic\_path

Run the use case:

```bash

python plot\_logistic\_path.py

```

Operation results:

```

NULL

```

15. Test case 15: plot\_logistic

Run the use case:

```bash

python plot\_logistic.py

```

Operation results:

```

```

16. Test case 16: plot\_multi\_task\_lasso\_support

Run the use case:

```bash

python plot\_multi\_task\_lasso\_support.py

```

Operation results:

```

NULL

```

17. Test case 17: plot\_nnls

Run the use case:

```bash

python plot\_nnls.py

```

Operation results:

```

NNLS R2 score 0.8225220806196525

OLS R2 score 0.7436926291700346

```

18. Test case 18: plot\_ols\_3d

Run the use case:

```bash

python plot\_ols\_3d.py

```

Operation results:

```

NULL

```

19. Test case 19: plot\_ols

Run the use case:

```bash

python plot\_ols.py

```

Operation results:

```

Coefficients:

[938.23786125]

Mean squared error: 2548.07

Coefficient of determination: 0.47

```

20. Test case 20: plot\_ols\_ridge\_variance

Run the use case:

```bash

python plot\_ols\_ridge\_variance.py

```

Operation results:

```

NULL

```

21. Test case 21: plot\_omp

Run the use case:

```bash

python plot\_omp.py

```

Operation results:

```

NULL

```

22. Test case 22: plot\_poisson\_regression\_non\_normal\_loss

Run the use case:

```bash

python plot\_poisson\_regression\_non\_normal\_loss.py

```

Operation results:

```

Average Frequency = 0.10070308464041304

Fraction of exposure with zero claims = 93.9%

Constant mean frequency evaluation:

MSE: 0.564

MAE: 0.189

mean Poisson deviance: 0.625

Ridge evaluation:

MSE: 0.560

MAE: 0.186

WARNING: Estimator yields invalid, non-positive predictions for 595 samples out of 223745. These predictions are ignored when computing the Poisson deviance.

mean Poisson deviance: 0.597

PoissonRegressor evaluation:

MSE: 0.560

MAE: 0.186

mean Poisson deviance: 0.594

Poisson Gradient Boosted Trees evaluation:

MSE: 0.559

MAE: 0.183

mean Poisson deviance: 0.574

Actual number of claims: 11935

Predicted number of claims by Ridge(alpha=1e-06): 11933.4

Predicted number of claims by PoissonRegressor(alpha=1e-12, solver='newton-cholesky'): 11932.0

Predicted number of claims by HistGradientBoostingRegressor(loss='poisson', max\_leaf\_nodes=128): 12117.4

Predicted number of claims by DummyRegressor(): 11931.2

```

23. Test case 23: plot\_polynomial\_interpolation

Run the use case:

```bash

python plot\_polynomial\_interpolation.py

```

Operation results:

```

NULL

```

24. Test case 24: plot\_quantile\_regression

Run the use case:

```bash

python plot\_quantile\_regression.py

```

Operation results:

```

Training error (in-sample performance)

LinearRegression:

MAE = 1.805

MSE = 6.486

QuantileRegressor:

MAE = 1.670

MSE = 7.025

Test error (cross-validated performance)

LinearRegression:

MAE = 1.732

MSE = 6.690

QuantileRegressor:

MAE = 1.679

MSE = 7.129

```

25. Test case 25: plot\_ransac

Run the use case:

```bash

python plot\_ransac.py

```

Operation results:

```

Estimated coefficients (true, linear regression, RANSAC):

82.1903908407869 [54.17236387] [82.08533159]

```

26. Test case 26: plot\_ridge\_coeffs

Run the use case:

```bash

python plot\_ridge\_coeffs.py

```

Operation results:

```

The true coefficient of this regression problem are:

[38.32634568 88.49665188 0. 29.75747153 0. 19.08699432

25.44381023 38.69892343 49.28808734 71.75949622]

```

27. Test case 27: plot\_ridge\_path

Run the use case:

```bash

python plot\_ridge\_path.py

```

Operation results:

```

NULL

```

28. Test case 28: plot\_robust\_fit

Run the use case:

```bash

python plot\_robust\_fit.py

```

Operation results:

```

```

29. Test case 29: plot\_sgd\_comparison

Run the use case:

```bash

python plot\_sgd\_comparison.py

```

Operation results:

```

NULL

```

30. Test case 30: plot\_sgd\_early\_stopping

Run the use case:

```bash

python plot\_sgd\_early\_stopping.py

```

Operation results:

```

NULL

```

31. Test case 31: plot\_sgd\_iris

Run the use case:

```bash

python plot\_sgd\_iris.py

```

Operation results:

```

NULL

```

32. Test case 32: plot\_sgd\_loss\_functions

Run the use case:

```bash

python plot\_sgd\_loss\_functions.py

```

Operation results:

```

NULL

```

33. Test case 33: plot\_sgdocsvm\_vs\_ocsvm

Run the use case:

```bash

python plot\_sgdocsvm\_vs\_ocsvm.py

```

Operation results:

```

NULL

```

34. Test case thirty-four: plot\_sgd\_penalties

Run the use case:

```bash

python plot\_sgd\_penalties.py

```

Operation results:

```

NULL

```

35. Test case 35: plot\_sgd\_separating\_hyperplane

Run the use case:

```bash

python plot\_sgd\_separating\_hyperplane.py

```

Operation results:

```

NULL

```

36. Test case thirty-six: plot\_sgd\_weighted\_samples

Run the use case:

```bash

python plot\_sgd\_weighted\_samples.py

```

Operation results:

```

NULL

```

37. Test case 37: plot\_sparse\_logistic\_regression\_20newsgroups

Run the use case:

```bash

python plot\_sparse\_logistic\_regression\_20newsgroups.py

```

Operation results:

```

Dataset 20newsgroup, train\_samples=4500, n\_features=130107, n\_classes=20

[model=One versus Rest, solver=saga] Number of epochs: 1

[model=One versus Rest, solver=saga] Number of epochs: 2

[model=One versus Rest, solver=saga] Number of epochs: 3

Test accuracy for model ovr: 0.5960

% non-zero coefficients for model ovr, per class:

[0.26593496 0.43348936 0.26362917 0.31973683 0.37815029 0.2928359

0.27054655 0.62717609 0.19522393 0.30897646 0.34586917 0.28207552

0.34125758 0.29898468 0.34279478 0.59489497 0.38353048 0.35278655

0.19829832 0.14603365]

Run time (3 epochs) for model ovr:6.91

[model=Multinomial, solver=saga] Number of epochs: 1

[model=Multinomial, solver=saga] Number of epochs: 2

[model=Multinomial, solver=saga] Number of epochs: 5

Test accuracy for model multinomial: 0.6440

% non-zero coefficients for model multinomial, per class:

[0.36047253 0.1268187 0.10606655 0.17985197 0.5395559 0.07993421

0.06686804 0.21443888 0.11528972 0.2075215 0.10914094 0.11144673

0.13988486 0.09684337 0.26286057 0.11682692 0.55800226 0.17370318

0.11452112 0.14603365]

Run time (5 epochs) for model multinomial:16.44

Example runs in 39.882 s

```

38. Test case 38: plot\_sparse\_logistic\_regression\_mnist

Run the use case:

```bash

python plot\_sparse\_logistic\_regression\_mnist.py

```

Operation results:

```

Sparsity with L1 penalty: 79.80%

Test score with L1 penalty: 0.8353

Example run in 60.632 s

```

39. Test case thirty-nine: plot\_theilsen

Run the use case:

```bash

python plot\_theilsen.py

```

Operation results:

```

NULL

```

40. Test case 40: plot\_tweedie\_regression\_insurance\_claims

Run the use case:

```bash

python plot\_tweedie\_regression\_insurance\_claims.py

```

Operation results:

```

ClaimNb Exposure Area VehPower VehAge DrivAge BonusMalus VehBrand \

IDpol

139 1 0.75 F 7 1 61 50 B12

190 1 0.14 B 12 5 50 60 B12

414 1 0.14 E 4 0 36 85 B12

424 2 0.62 F 10 0 51 100 B12

463 1 0.31 A 5 0 45 50 B12

VehGas Density Region ClaimAmount PurePremium Frequency \

IDpol

139 Regular 27000 R11 303.00 404.000000 1.333333

190 Diesel 56 R25 1981.84 14156.000000 7.142857

414 Regular 4792 R11 1456.55 10403.928571 7.142857

424 Regular 27000 R11 10834.00 17474.193548 3.225806

463 Regular 12 R73 3986.67 12860.225806 3.225806

AvgClaimAmount

IDpol

139 303.00

190 1981.84

414 1456.55

424 5417.00

463 3986.67

Evaluation of PoissonRegressor on target Frequency

subset train test

metric

D² explained 0.0448 0.0427

mean abs. error 0.1379 0.1378

mean squared error 0.2441 0.2246

Evaluation of GammaRegressor on target AvgClaimAmount

subset train test

metric

D² explained 3.900000e-03 4.400000e-03

mean abs. error 1.756746e+03 1.744042e+03

mean squared error 5.801770e+07 5.030677e+07

Evaluation of a mean predictor on target AvgClaimAmount

subset train test

metric

D² explained 0.000000e+00 -0.000000e+00

mean abs. error 1.756687e+03 1.744497e+03

mean squared error 5.803882e+07 5.033764e+07

Mean AvgClaim Amount per policy: 71.78

Mean AvgClaim Amount | NbClaim > 0: 1951.21

Predicted Mean AvgClaim Amount | NbClaim > 0: 1940.95

Predicted Mean AvgClaim Amount (dummy) | NbClaim > 0: 1978.59

Evaluation of the Product Model and the Tweedie Regressor on target PurePremium

Product Model TweedieRegressor

subset train test train test

metric

D² explained NaN NaN 1.640000e-02 1.370000e-02

mean Tweedie dev p=1.5000 7.669930e+01 7.617050e+01 7.640770e+01 7.640880e+01

mean Tweedie dev p=1.7000 3.695740e+01 3.683980e+01 3.682880e+01 3.692270e+01

mean Tweedie dev p=1.8000 3.046010e+01 3.040530e+01 3.037600e+01 3.045390e+01

mean Tweedie dev p=1.9000 3.387580e+01 3.385000e+01 3.382120e+01 3.387830e+01

mean Tweedie dev p=1.9900 2.015716e+02 2.015414e+02 2.015347e+02 2.015587e+02

mean Tweedie dev p=1.9990 1.914573e+03 1.914370e+03 1.914538e+03 1.914387e+03

mean Tweedie dev p=1.9999 1.904751e+04 1.904556e+04 1.904747e+04 1.904558e+04

mean abs. error 2.730119e+02 2.722128e+02 2.739865e+02 2.731249e+02

mean squared error 3.295040e+07 3.212197e+07 3.295505e+07 3.213056e+07

subset train test

observed 3.917618e+07 1.299546e+07

predicted, frequency\*severity model 3.916555e+07 1.313276e+07

predicted, tweedie, power=1.90 3.951751e+07 1.325198e+07

```

### manifold

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/manifold

```

1. Test case 1: plot\_compare\_methods

Run the use case:

```bash

python plot\_compare\_methods.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_lle\_digits

Run the use case:

```bash

python plot\_lle\_digits.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_manifold\_sphere

Run the use case:

```bash

python plot\_manifold\_sphere.py

```

Operation results:

```

standard: 0.41 sec

ltsa: 1 sec

hessian: 1.2 sec

modified: 0.91 sec

ISO: 0.51 sec

MDS: 5.2 sec

Spectral Embedding: 0.3 sec

t-SNE: 13 sec

```

4. Test case 4: plot\_mds

Run the use case:

```bash

python plot\_mds.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_swissroll

Run the use case:

```bash

python plot\_swissroll.py

```

Operation results:

```

NULL

```

6. Test case 6: plot\_t\_sne\_perplexity

Run the use case:

```bash

python plot\_t\_sne\_perplexity.py

```

Operation results:

```

circles, perplexity=5 in 0.7 sec

circles, perplexity=30 in 0.96 sec

circles, perplexity=50 in 0.9 sec

circles, perplexity=100 in 1 sec

S-curve, perplexity=5 in 0.59 sec

S-curve, perplexity=30 in 0.82 sec

S-curve, perplexity=50 in 0.95 sec

S-curve, perplexity=100 in 1 sec

uniform grid, perplexity=5 in 0.87 sec

uniform grid, perplexity=30 in 1.1 sec

uniform grid, perplexity=50 in 1.1 sec

uniform grid, perplexity=100 in 1.2 sec

```

### miscellaneous

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/miscellaneous

```

1. Test case 1: plot\_anomaly\_comparison

Run the use case:

```bash

python plot\_anomaly\_comparison.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_display\_object\_visualization

Run the use case:

```bash

python plot\_display\_object\_visualization.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_estimator\_representation

Run the use case:

```bash

python plot\_estimator\_representation.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_isotonic\_regression

Run the use case:

```bash

python plot\_isotonic\_regression.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_johnson\_lindenstrauss\_bound

Run the use case:

```bash

python plot\_johnson\_lindenstrauss\_bound.py

```

Operation results:

```

Embedding 300 samples with dim 130107 using various random projections

Projected 300 samples from 130107 to 300 in 3.016s

Random matrix with size: 1.296 MB

Mean distances rate: 0.98 (0.15)

Projected 300 samples from 130107 to 1000 in 8.763s

Random matrix with size: 4.333 MB

Mean distances rate: 1.03 (0.10)

Projected 300 samples from 130107 to 10000 in 99.421s

Random matrix with size: 43.309 MB

Mean distances rate: 1.01 (0.03)

```

6. Test case 6: plot\_kernel\_approximation

Run the use case:

```bash

python plot\_kernel\_approximation.py

```

Operation results:

```

NULL

```

7. Test case 7: plot\_kernel\_ridge\_regression

Run the use case:

```bash

python plot\_kernel\_ridge\_regression.py

```

Operation results:

```

Best SVR with params: {'C': 1.0, 'gamma': np.float64(0.1)} and R2 score: 0.737

SVR complexity and bandwidth selected and model fitted in 4.259 s

Best KRR with params: {'alpha': 0.1, 'gamma': np.float64(0.1)} and R2 score: 0.723

KRR complexity and bandwidth selected and model fitted in 3.748 s

Support vector ratio: 0.340

SVR prediction for 100000 inputs in 0.727 s

KRR prediction for 100000 inputs in 0.898 s

```

8. Test case eight: plot\_metadata\_routing

Run the use case:

```bash

python plot\_metadata\_routing.py

```

Operation results:

```

NULL

```

9. Test case nine: plot\_multilabel

Run the use case:

```bash

python plot\_multilabel.py

```

Operation results:

```

NULL

```

10. Test case 10: plot\_multioutput\_face\_completion

Run the use case:

```bash

python plot\_multioutput\_face\_completion.py

```

Operation results:

```

NULL

```

11. Test case 11: plot\_outlier\_detection\_bench

Run the use case:

```bash

python plot\_outlier\_detection\_bench.py

```

Operation results:

```

10065 datapoints with 338 anomalies (3.36%)

Duration for LOF: 10.51 s

Duration for IForest: 1.72 s

14302 datapoints with 137 anomalies (0.96%)

Duration for LOF: 12.71 s

Duration for IForest: 1.38 s

2714 datapoints with 30 anomalies (1.11%)

Duration for LOF: 4.99 s

Duration for IForest: 1.30 s

2126 datapoints with 53 anomalies (2.49%)

Duration for LOF: 0.26 s

Duration for IForest: 0.85 s

```

12. Test case 12: plot\_partial\_dependence\_visualization\_api

Run the use case:

```bash

python plot\_partial\_dependence\_visualization\_api.py

```

Operation results:

```

NULL

```

13. Test case 13: plot\_pipeline\_display

Run the use case:

```bash

python plot\_pipeline\_display.py

```

Operation results:

```

NULL

```

14. Test case 14: plot\_roc\_curve\_visualization\_api

Run the use case:

```bash

python plot\_roc\_curve\_visualization\_api.py

```

Operation results:

```

NULL

```

15. Test case 15: plot\_set\_output

Run the use case:

```bash

python plot\_set\_output.py

```

Operation results:

```

NULL

```

### mixture

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/mixture

```

1. Test case 1: plot\_concentration\_prior

Run the use case:

```bash

python plot\_concentration\_prior.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_gmm\_covariances

Run the use case:

```bash

python plot\_gmm\_covariances.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_gmm\_init

Run the use case:

```bash

python plot\_gmm\_init.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_gmm\_pdf

Run the use case:

```bash

python plot\_gmm\_pdf.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_gmm

Run the use case:

```bash

python plot\_gmm.py

```

Operation results:

```

NULL

```

6. Test case six: plot\_gmm\_selection

Run the use case:

```bash

python plot\_gmm\_selection.py

```

Operation results:

```

NULL

```

7. Test case 7: plot\_gmm\_sin

Run the use case:

```bash

python plot\_gmm\_sin.py

```

Operation results:

```

NULL

```

### model\_selection

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/model\_selection

```

1. Test case 1: plot\_confusion\_matrix

Run the use case:

```bash

python plot\_confusion\_matrix.py

```

Operation results:

```

Confusion matrix, without normalization

[[13 0 0]

[ 0 10 6]

[ 0 0 9 ]]

Normalized confusion matrix

[[1. 0. 0. ]

[0. 0.62 0.38]

[0. 0. 1. ]]

```

2. Test case 2: plot\_cost\_sensitive\_learning

Run the use case:

```bash

python plot\_cost\_sensitive\_learning.py

```

Operation results:

```

memory usage: 63.3 MB

Benefit/cost of our easy-going classifier: 221,445.07€

Benefit/cost of our intolerant classifier: -668,903.24€

Balanced accuracy of our easy-going classifier: 0.500

Balanced accuracy of our intolerant classifier: 0.500

Benefit/cost of our logistic regression: 260,787.21€

Balanced accuracy of our logistic regression: 0.815

Benefit/cost of our logistic regression: 268,847.31€

Balanced accuracy of our logistic regression: 0.898

Benefit/cost of our logistic regression: 268,847.31€

Balanced accuracy of our logistic regression: 0.898

```

3. Test case three: plot\_cv\_indices

Run the use case:

```bash

python plot\_cv\_indices.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_cv\_predict

Run the use case:

```bash

python plot\_cv\_predict.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_det

Run the use case:

```bash

python plot\_det.py

```

Operation results:

```

NULL

```

6. Test case six: plot\_grid\_search\_digits

Run the use case:

```bash

python plot\_grid\_search\_digits.py

```

Operation results:

```

The selected final model is the fastest to predict out of the previously

selected subset of best models based on precision and recall.

Its scoring time is:

mean\_score\_time 0.040909

mean\_test\_recall 0.877206

std\_test\_recall 0.069196

mean\_test\_precision 1.0

std\_test\_precision 0.0

rank\_test\_recall 3

rank\_test\_precision 1

params {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}

Name: 2, dtype: object

precision recall f1-score support

False 0.99 1.00 0.99 807

True 1.00 0.87 0.93 92

accuracy 0.99 899

macro avg 0.99 0.93 0.96 899

weighted avg 0.99 0.99 0.99 899

```

7. Test case 7: plot\_grid\_search\_refit\_callable

Run the use case:

```bash

python plot\_grid\_search\_refit\_callable.py

```

Operation results:

```

The best\_index\_ is 2

The n\_components selected is 10

The corresponding accuracy score is 0.88

```

8. Test case eight: plot\_grid\_search\_stats

Run the use case:

```bash

python plot\_grid\_search\_stats.py

```

Operation results:

```

Correlation of models:

kernel rbf linear 3\_poly 2\_poly

kernel

rbf 1.000000 0.882561 0.783392 0.351390

linear 0.882561 1.000000 0.746492 0.298688

3\_poly 0.783392 0.746492 1.000000 0.355440

2\_poly 0.351390 0.298688 0.355440 1.000000

Corrected t-value: 0.750

Corrected p-value: 0.227

Uncorrected t-value: 2.611

Uncorrected p-value: 0.005

Probability of rbf being more accurate than linear: 0.773

Probability of linear being more accurate than rbf: 0.227

Probability of rbf and linear being practically equivalent: 0.432

```

9. Test case nine: plot\_grid\_search\_text\_feature\_extraction

Run the use case:

```bash

python plot\_grid\_search\_text\_feature\_extraction.py

```

Operation results:

```

Loading 20 newsgroups dataset for 2 categories:

['alt.atheism', 'talk.religion.misc']

857 documents

Performing grid search...

Hyperparameters to be evaluated:

{'clf\_\_alpha': array([1.e-06, 1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01,

1.e+02, 1.e+03, 1.e+04, 1.e+05, 1.e+06]),

'vect\_\_max\_df': (0.2, 0.4, 0.6, 0.8, 1.0),

'vect\_\_min\_df': (1, 3, 5, 10),

'vect\_\_ngram\_range': ((1, 1), (1, 2)),

'vect\_\_norm': ('l1', 'l2')}

Fitting 5 folds for each of 40 candidates, totaling 200 fits

Done in 240.264s

Best parameter combination found:

clf\_\_alpha: 0.01

vector\_\_max\_df: 0.2

vector\_\_min\_df: 1

vector\_\_ngram\_range: (1, 1)

vect\_\_norm: l1

Accuracy of the best parameters using the inner CV of the random search: 0.816

Accuracy on test set: 0.709

```

10. Test case 10: plot\_learning\_curve

Run the use case:

```bash

python plot\_learning\_curve.py

```

Operation results:

```

NULL

```

11. Test case 11: plot\_likelihood\_ratios

Run the use case:

```bash

python plot\_likelihood\_ratios.py

```

Operation results:

```

Percentage of people carrying the disease: 10.37%

LR+: 12.617

```

12. Test case 12: plot\_multi\_metric\_evaluation

Run the use case:

```bash

python plot\_multi\_metric\_evaluation.py

```

Operation results:

```

NULL

```

13. Test case 13: plot\_nested\_cross\_validation\_iris

Run the use case:

```bash

python plot\_nested\_cross\_validation\_iris.py

```

Operation results:

```

Average difference of 0.007361 with std. dev. of 0.007760.

```

14. Test case 14: plot\_permutation\_tests\_for\_classification

Run the use case:

```bash

python plot\_permutation\_tests\_for\_classification.py

```

Operation results:

```

NULL

```

15. Test case 15: plot\_precision\_recall

Run the use case:

```bash

python plot\_precision\_recall.py

```

Operation results:

```

NULL

```

16. Test case 16: plot\_randomized\_search

Run the use case:

```bash

python plot\_randomized\_search.py

```

Operation results:

```

RandomizedSearchCV took 4.32 seconds for 15 parameter candidates settings.

Model with rank: 1

Mean validation score: 0.983 (std: 0.012)

Parameters: {'alpha': np.float64(0.014366617976279942), 'average': False, 'l1\_ratio': np.float64(0.18975084484454796)}

Model with rank: 2

Mean validation score: 0.981 (std: 0.016)

Parameters: {'alpha': np.float64(0.01141094466077932), 'average': False, 'l1\_ratio': np.float64(0.3361030835556883)}

Model with rank: 3

Mean validation score: 0.980 (std: 0.015)

Parameters: {'alpha': np.float64(0.04580771832828349), 'average': False, 'l1\_ratio': np.float64(0.8663404432338067)}

GridSearchCV took 14.32 seconds for 60 candidate parameter settings.

Model with rank: 1

Mean validation score: 0.993 (std: 0.007)

Parameters: {'alpha': np.float64(0.01), 'average': False, 'l1\_ratio': np.float64(0.3333333333333333)}

Model with rank: 2

Mean validation score: 0.989 (std: 0.014)

Parameters: {'alpha': np.float64(0.01), 'average': False, 'l1\_ratio': np.float64(0.6666666666666666)}

Model with rank: 3

Mean validation score: 0.989 (std: 0.009)

Parameters: {'alpha': np.float64(0.1), 'average': False, 'l1\_ratio': np.float64(0.0)}

Model with rank: 3

Mean validation score: 0.989 (std: 0.004)

Parameters: {'alpha': np.float64(0.1), 'average': False, 'l1\_ratio': np.float64(0.5555555555555556)}

Model with rank: 3

Mean validation score: 0.989 (std: 0.007)

Parameters: {'alpha': np.float64(1.0), 'average': False, 'l1\_ratio': np.float64(0.0)}

```

17. Test case 17: plot\_roc\_crossval

Run the use case:

```bash

python plot\_roc\_crossval.py

```

Operation results:

```

NULL

```

18. Test case 18: plot\_roc

Run the use case:

```bash

python plot\_roc.py

```

Operation results:

```

y\_score:

[[0.38095776 0.05072909 0.56831315]

[0.07031555 0.27915668 0.65052777]]

y\_score.ravel():

[0.38095776 0.05072909 0.56831315 0.07031555 0.27915668 0.65052777]

Micro-averaged One-vs-Rest ROC AUC score:

0.77

Micro-averaged One-vs-Rest ROC AUC score:

0.77

Macro-averaged One-vs-Rest ROC AUC score:

0.78

Macro-averaged One-vs-Rest ROC AUC score:

0.78

[(np.str\_('setosa'), np.str\_('versicolor')), (np.str\_('setosa'), np.str\_('virginica')), (np.str\_('versicolor'), np.str\_('virginica'))]

Macro-averaged One-vs-One ROC AUC score:

0.78

Macro-averaged One-vs-One ROC AUC score:

0.78

```

19. Test case 19: plot\_successive\_halving\_heatmap

Run the use case:

```bash

python plot\_successive\_halving\_heatmap.py

```

Operation results:

```

NULL

```

20. Test case 20: plot\_successive\_halving\_iterations

Run the use case:

```bash

python plot\_successive\_halving\_iterations.py

```

Operation results:

```

NULL

```

21. Test case 21: plot\_train\_error\_vs\_test\_error

Run the use case:

```bash

python plot\_train\_error\_vs\_test\_error.py

```

Operation results:

```

Optimal regularization parameter: 0.000335292414924956

```

22. Test case 22: plot\_tuned\_decision\_threshold

Run the use case:

```bash

python plot\_tuned\_decision\_threshold.py

```

Operation results:

```

NULL

```

23. Test case 23: plot\_underfitting\_overfitting

Run the use case:

```bash

python plot\_underfitting\_overfitting.py

```

Operation results:

```

NULL

```

24. Test case 24: plot\_validation\_curve

Run the use case:

```bash

python plot\_validation\_curve.py

```

Operation results:

```

NULL

```

### multiclass

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/multiclass

```

1. Test case 1: plot\_multiclass\_overview

Run the use case:

```bash

python plot\_multiclass\_overview.py

```

Operation results:

```

NULL

```

### multioutput

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/multioutput

```

1. Test case 1: plot\_classifier\_chain\_yeast

Run the use case:

```bash

python plot\_classifier\_chain\_yeast.py

```

Operation results:

```

NULL

```

### neighbors

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/neighbors

```

1. Test case 1: approximate\_nearest\_neighbors

Run the use case:

```bash

python approximate\_nearest\_neighbors.py

```

Operation results:

```

Unable to install nmslib

gcc: error: '-march=native': ISA string must begin with rv32 or rv64

error: command '/usr/lib64/ccache/gcc' failed with exit code 1

```

2. Test case 2: plot\_caching\_nearest\_neighbors

Run the use case:

```bash

python plot\_caching\_nearest\_neighbors.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_classification

Run the use case:

```bash

python plot\_classification.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_digits\_kde\_sampling

Run the use case:

```bash

python plot\_digits\_kde\_sampling.py

```

Operation results:

```

best bandwidth: 3.79269019073225

```

5. Test case 5: plot\_kde\_1d

Run the use case:

```bash

python plot\_kde\_1d.py

```

Operation results:

```

NULL

```

6. Test case 6: plot\_lof\_novelty\_detection

Run the use case:

```bash

python plot\_lof\_novelty\_detection.py

```

Operation results:

```

NULL

```

7. Test case 7: plot\_lof\_outlier\_detection

Run the use case:

```bash

python plot\_lof\_outlier\_detection.py

```

Operation results:

```

NULL

```

8. Test case eight: plot\_nca\_classification

Run the use case:

```bash

python plot\_nca\_classification.py

```

Operation results:

```

NULL

```

9. Test case nine: plot\_nca\_dim\_reduction

Run the use case:

```bash

python plot\_nca\_dim\_reduction.py

```

Operation results:

```

NULL

```

10. Test case 10: plot\_nca\_illustration

Run the use case:

```bash

python plot\_nca\_illustration.py

```

Operation results:

```

NULL

```

11. Test case 11: plot\_nearest\_centroid

Run the use case:

```bash

python plot\_nearest\_centroid.py

```

Operation results:

```

None 0.8133333333333334

0.2 0.82

```

12. Test case 12: plot\_regression

Run the use case:

```bash

python plot\_regression.py

```

Operation results:

```

NULL

```

13. Test case 13: plot\_species\_kde

Run the use case:

```bash

python plot\_species\_kde.py

```

Operation results:

```

NULL

```

### neural\_networks

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/neural\_networks

```

1. Test case 1: plot\_mlp\_alpha

Run the use case:

```bash

python plot\_mlp\_alpha.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_mlp\_training\_curves

Run the use case:

```bash

python plot\_mlp\_training\_curves.py

```

Operation results:

```

learning on dataset iris

training: constant learning-rate

Training set score: 0.980000

Training set loss: 0.096950

training: constant with momentum

Training set score: 0.980000

Training set loss: 0.049530

training: constant with Nesterov's momentum

Training set score: 0.980000

Training set loss: 0.049540

training: inv-scaling learning-rate

Training set score: 0.360000

Training set loss: 0.978444

training: inv-scaling with momentum

Training set score: 0.860000

Training set loss: 0.504185

training: inv-scaling with Nesterov's momentum

Training set score: 0.860000

Training set loss: 0.503452

training: adam

Training set score: 0.980000

Training set loss: 0.045311

```

3. Test case three: plot\_mnist\_filters

Run the use case:

```bash

python plot\_mnist\_filters.py

```

Operation results:

```

Iteration 1, loss = 0.44139186

Iteration 2, loss = 0.19174891

Iteration 3, loss = 0.13983521

Iteration 4, loss = 0.11378556

Iteration 5, loss = 0.09443967

Iteration 6, loss = 0.07846529

Iteration 7, loss = 0.06506307

Iteration 8, loss = 0.05534985

Training set score: 0.986429

Test set score: 0.953061

```

4. Test case 4: plot\_rbm\_logistic\_classification

Run the use case:

```bash

python plot\_rbm\_logistic\_classification.py

```

Operation results:

```

[BernoulliRBM] Iteration 1, pseudo-likelihood = -25.57, time = 0.87s

[BernoulliRBM] Iteration 2, pseudo-likelihood = -23.68, time = 1.17s

[BernoulliRBM] Iteration 3, pseudo-likelihood = -22.88, time = 1.17s

[BernoulliRBM] Iteration 4, pseudo-likelihood = -21.91, time = 1.16s

[BernoulliRBM] Iteration 5, pseudo-likelihood = -21.79, time = 1.16s

[BernoulliRBM] Iteration 6, pseudo-likelihood = -20.96, time = 1.19s

[BernoulliRBM] Iteration 7, pseudo-likelihood = -20.88, time = 1.18s

[BernoulliRBM] Iteration 8, pseudo-likelihood = -20.50, time = 1.15s

[BernoulliRBM] Iteration 9, pseudo-likelihood = -20.36, time = 1.18s

[BernoulliRBM] Iteration 10, pseudo-likelihood = -20.07, time = 1.21s

Logistic regression using RBM features:

precision recall f1-score support

0 0.10 1.00 0.18 174

1 0.00 0.00 0.00 184

2 0.00 0.00 0.00 166

3 0.00 0.00 0.00 194

4 0.00 0.00 0.00 186

5 0.00 0.00 0.00 181

6 0.00 0.00 0.00 207

7 0.00 0.00 0.00 154

8 0.00 0.00 0.00 182

9 0.00 0.00 0.00 169

accuracy 0.10 1797

macro avg 0.01 0.10 0.02 1797

weighted avg 0.01 0.10 0.02 1797

Logistic regression using raw pixel features:

precision recall f1-score support

0 0.10 1.00 0.18 174

1 0.00 0.00 0.00 184

2 0.00 0.00 0.00 166

3 0.00 0.00 0.00 194

4 0.00 0.00 0.00 186

5 0.00 0.00 0.00 181

6 0.00 0.00 0.00 207

7 0.00 0.00 0.00 154

8 0.00 0.00 0.00 182

9 0.00 0.00 0.00 169

accuracy 0.10 1797

macro avg 0.01 0.10 0.02 1797

weighted avg 0.01 0.10 0.02 1797

```

### preprocessing

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/preprocessing

```

1. Test case 1: plot\_all\_scaling

Run the use case:

```bash

python plot\_all\_scaling.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_discretization\_classification

Run the use case:

```bash

python plot\_discretization\_classification.py

```

Operation results:

```

dataset 0

---------

LogisticRegression: 0.86

LinearSVC: 0.86

/home/yangchangke/venv/lib/python3.11/site-packages/numpy/ma/core.py:2846: RuntimeWarning: invalid value encountered in cast

\_data = np.array(data, dtype=dtype, copy=copy,

KBinsDiscretizer + LogisticRegression: 0.86

KBinsDiscretizer + LinearSVC: 0.94

GradientBoostingClassifier: 0.90

SVC: 0.94

dataset 1

---------

LogisticRegression: 0.40

LinearSVC: 0.40

KBinsDiscretizer + LogisticRegression: 0.78

KBinsDiscretizer + LinearSVC: 0.80

GradientBoostingClassifier: 0.84

SVC: 0.84

dataset 2

---------

LogisticRegression: 0.98

LinearSVC: 0.96

KBinsDiscretizer + LogisticRegression: 0.94

KBinsDiscretizer + LinearSVC: 0.94

GradientBoostingClassifier: 0.94

SVC: 0.98

```

3. Test case 3: plot\_discretization

Run the use case:

```bash

python plot\_discretization.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_discretization\_strategies

Run the use case:

```bash

python plot\_discretization\_strategies.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_map\_data\_to\_normal

Run the use case:

```bash

python plot\_map\_data\_to\_normal.py

```

Operation results:

```

NULL

```

6. Test case 6: plot\_scaling\_importance

Run the use case:

```bash

python plot\_scaling\_importance.py

```

Operation results:

```

Optimal C for the unscaled PCA: 0.0004

Optimal C for the standardized data with PCA: 20.69

Test accuracy for the unscaled PCA

35.19%

Test accuracy for the standardized data with PCA

96.30%

Log-loss for the unscaled PCA

0.957

Log-loss for the standardized data with PCA

0.0825

```

7. Test case seven: plot\_target\_encoder\_cross\_val

Run the use case:

```bash

python plot\_target\_encoder\_cross\_val.py

```

Operation results:

```

Raw Model score on training set: 0.0049896314219657345

Raw Model score on test set: 0.004577621579882729

Model with CF on train set: 0.8000184677460297

Model with CF on test set: 0.7927845601690889

Model without CF on training set: 0.858486250088675

Model without CF on test set: 0.6338211367106759

```

8. Test case eight: plot\_target\_encoder

Run the use case:

```bash

python plot\_target\_encoder.py

```

Operation results:

```

NULL

```

### release\_highlights

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/release\_highlights

```

1. Test case 1: plot\_release\_highlights\_0\_22\_0

Run the use case:

```bash

python plot\_release\_highlights\_0\_22\_0.py

```

Operation results:

```

ax.boxplot(

[0 0 1 1]

[[1. 2. 4. ]

[3. 4. 3. ]

[5.5 6. 5. ]

[8. 8. 7.]]

Average number of nodes without pruning 22.3

Average number of nodes with pruning 6.4

pclass embarked

0 1 S

1 1 S

2 1 S

3 1 S

4 1 S

0.9837403549382717

```

2. Test case 2: plot\_release\_highlights\_0\_23\_0

Run the use case:

```bash

python plot\_release\_highlights\_0\_23\_0.py

```

Operation results:

```

0.35776189065725794

0.42425183539869415

0.6875046614372226

0.999791942438998

```

3. Test case three: plot\_release\_highlights\_0\_24\_0

Run the use case:

```bash

python plot\_release\_highlights\_0\_24\_0.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_release\_highlights\_1\_0\_0

Run the use case:

```bash

python plot\_release\_highlights\_1\_0\_0.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_release\_highlights\_1\_1\_0

Run the use case:

```bash

python plot\_release\_highlights\_1\_1\_0.py

```

Operation results:

```

relative reconstruction error: 0.00364

```

6. Test case 6: plot\_release\_highlights\_1\_2\_0

Run the use case:

```bash

python plot\_release\_highlights\_1\_2\_0.py

```

Operation results:

```

NULL

```

7. Test case 7: plot\_release\_highlights\_1\_3\_0

Run the use case:

```bash

python plot\_release\_highlights\_1\_3\_0.py

```

Operation results:

```

number of digits: 10

number of clusters found: 10

0.9751818034688537

```

8. Test case 8: plot\_release\_highlights\_1\_4\_0

Run the use case:

```bash

python plot\_release\_highlights\_1\_4\_0.py

```

Operation results:

```

ModuleNotFoundError: No module named 'polars'

```

9. Test case nine: plot\_release\_highlights\_1\_5\_0

Run the use case:

```bash

python plot\_release\_highlights\_1\_5\_0.py

```

Operation results:

```

Untuned decision threshold: 0.5

Custom score: -0.12

Tuned decision threshold: 0.071

Custom score: 0.04

Explained variance: 0.88

Explained variance: 0.13

StandardScaler()

OneHotEncoder()

```

### semi\_supervised

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/semi\_supervised

```

1. Test case 1: plot\_label\_propagation\_digits\_active\_learning

Run the use case:

```bash

python plot\_label\_propagation\_digits\_active\_learning.py

```

Operation results:

```

Iteration 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Label Spreading model: 40 labeled & 290 unlabeled (330 total)

precision recall f1-score support

0 1.00 1.00 1.00 22

1 0.78 0.69 0.73 26

2 0.93 0.93 0.93 29

3 1.00 0.89 0.94 27

4 0.92 0.96 0.94 23

5 0.96 0.70 0.81 33

6 0.97 0.97 0.97 35

7 0.94 0.91 0.92 33

8 0.62 0.89 0.74 28

9 0.73 0.79 0.76 34

accuracy 0.87 290

macro avg 0.89 0.87 0.87 290

weighted avg 0.88 0.87 0.87 290

Confusion matrix

[[22 0 0 0 0 0 0 0 0 0]

[ 0 18 2 0 0 0 1 0 5 0 ]

[ 0 0 27 0 0 0 0 0 2 0]

[ 0 0 0 24 0 0 0 0 3 0]

[ 0 1 0 0 22 0 0 0 0 0 ]

[ 0 0 0 0 0 23 0 0 0 10]

[ 0 1 0 0 0 0 34 0 0 0 ]

[ 0 0 0 0 0 0 0 30 3 0]

[ 0 3 0 0 0 0 0 0 25 0]

[ 0 0 0 0 2 1 0 2 2 27]]

```

2. Test case 2: plot\_label\_propagation\_digits

Run the use case:

```bash

python plot\_label\_propagation\_digits.py

```

Operation results:

```

Label Spreading model: 40 labeled & 300 unlabeled points (340 total)

precision recall f1-score support

0 1.00 1.00 1.00 27

1 0.82 1.00 0.90 37

2 1.00 0.86 0.92 28

3 1.00 0.80 0.89 35

4 0.92 1.00 0.96 24

5 0.74 0.94 0.83 34

6 0.89 0.96 0.92 25

7 0.94 0.89 0.91 35

8 1.00 0.68 0.81 31

9 0.81 0.88 0.84 24

accuracy 0.90 300

macro avg 0.91 0.90 0.90 300

weighted avg 0.91 0.90 0.90 300

```

3. Test case three: plot\_label\_propagation\_structure

Run the use case:

```bash

python plot\_label\_propagation\_structure.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_self\_training\_varying\_threshold

Run the use case:

```bash

python plot\_self\_training\_varying\_threshold.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_semi\_supervised\_newsgroups

Run the use case:

```bash

python plot\_semi\_supervised\_newsgroups.py

```

Operation results:

```

2823 documents

5 categories

Supervised SGDClassifier on 100% of the data:

Number of training samples: 2117

Unlabeled samples in training set: 0

Micro-averaged F1 score on test set: 0.884

----------

Supervised SGDClassifier on 20% of the training data:

Number of training samples: 419

Unlabeled samples in training set: 0

Micro-averaged F1 score on test set: 0.754

----------

SelfTrainingClassifier on 20% of the training data (rest is unlabeled):

Number of training samples: 2117

Unlabeled samples in training set: 1698

End of iteration 1, added 1112 new labels.

End of iteration 2, added 187 new labels.

End of iteration 3, added 67 new labels.

End of iteration 4, added 26 new labels.

End of iteration 5, added 10 new labels.

End of iteration 6, added 8 new labels.

End of iteration 7, added 3 new labels.

End of iteration 8, added 7 new labels.

End of iteration 9, added 3 new labels.

End of iteration 10, added 1 new labels.

Micro-averaged F1 score on test set: 0.819

----------

LabelSpreading on 20% of the data (rest is unlabeled):

Number of training samples: 2117

Unlabeled samples in training set: 1698

Micro-averaged F1 score on test set: 0.623

```

6. Test case 6: plot\_semi\_supervised\_versus\_svm\_iris

Run the use case:

```bash

python plot\_semi\_supervised\_versus\_svm\_iris.py

```

Operation results:

```

NULL

```

### svm

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/svm

```

1. Test case 1: plot\_custom\_kernel

Run the use case:

```bash

python plot\_custom\_kernel.py

```

Operation results:

```

NULL

```

2. Test case 2: plot\_iris\_svc

Run the use case:

```bash

python plot\_iris\_svc.py

```

Operation results:

```

NULL

```

3. Test case three: plot\_linearsvc\_support\_vectors

Run the use case:

```bash

python plot\_linearsvc\_support\_vectors.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_oneclass

Run the use case:

```bash

python plot\_oneclass.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_rbf\_parameters

Run the use case:

```bash

python plot\_rbf\_parameters.py

```

Operation results:

```

The best parameters are {'C': 1.0, 'gamma': 0.1} with a score of 0.97

```

6. Test case 6: plot\_separating\_hyperplane

Run the use case:

```bash

python plot\_separating\_hyperplane.py

```

Operation results:

```

NULL

```

7. Test case 7: plot\_separating\_hyperplane\_unbalanced

Run the use case:

```bash

python plot\_separating\_hyperplane\_unbalanced.py

```

Operation results:

```

NULL

```

8. Test case eight: plot\_svm\_anova

Run the use case:

```bash

python plot\_svm\_anova.py

```

Operation results:

```

NULL

```

9. Test case nine: plot\_svm\_kernels

Run the use case:

```bash

python plot\_svm\_kernels.py

```

Operation results:

```

NULL

```

10. Test case 10: plot\_svm\_margin

Run the use case:

```bash

python plot\_svm\_margin.py

```

Operation results:

```

NULL

```

11. Test case 11: plot\_svm\_regression

Run the use case:

```bash

python plot\_svm\_regression.py

```

Operation results:

```

NULL

```

12. Test case 12: plot\_svm\_scale\_c

Run the use case:

```bash

python plot\_svm\_scale\_c.py

```

Operation results:

```

NULL

```

13. Test case 13: plot\_svm\_tie\_breaking

Run the use case:

```bash

python plot\_svm\_tie\_breaking.py

```

Operation results:

```

NULL

```

14. Test case 14: plot\_weighted\_samples

Run the use case:

```bash

python plot\_weighted\_samples.py

```

Operation results:

```

NULL

```

### text

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/text

```

1. Test case 1: plot\_document\_classification\_20newsgroups

Run the use case:

```bash

python plot\_document\_classification\_20newsgroups.py

```

Operation results:

```

================================================================================

Logistic Regression

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training:

LogisticRegression(C=5, max\_iter=1000)

train time: 0.727s

Test time: 0.00269s

accuracy: 0.772

dimensionality: 5316

density: 1.0

================================================================================

Ridge Classifier

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training:

RidgeClassifier(solver='sparse\_cg')

train time: 0.157s

Test time: 0.00412s

accuracy: 0.76

dimensionality: 5316

density: 1.0

================================================================================

kNN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training:

KNeighborsClassifier(n\_neighbors=100)

train time: 0.00422s

Test time: 0.431s

accuracy: 0.752

================================================================================

Random Forest

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training:

RandomForestClassifier()

train time: 8.08s

Test time: 0.154s

accuracy: 0.706

```

2. Test case 2: plot\_document\_clustering

Run the use case:

```bash

python plot\_document\_clustering.py

```

Operation results:

```

3387 documents - 4 categories

vectorization done in 1.842 s

n\_samples: 3387, n\_features: 7929

0.007

Number of assigned elements to each cluster: [ 481 675 1785 446]

Number of assigned elements to each cluster: [ 575 619 485 1708]

Number of assigned elements to each cluster: [ 1 1 1 3384]

Number of assigned elements to each cluster: [1887 311 332 857]

Number of assigned elements to each cluster: [ 291 673 1771 652]

True number of documents in each category according to the class labels: [799 973 987 628]

Clustering done in 1.44 ± 0.31 s

Homogeneity: 0.349 ± 0.012

Completeness: 0.400 ± 0.006

V-measure: 0.373 ± 0.009

Adjusted Rand-Index: 0.226 ± 0.035

Silhouette coefficient: 0.007 ± 0.000

LSA done in 2.861 s

Explained variance of the SVD step: 18.4%

Clustering done in 0.16 ± 0.02 s

Homogeneity: 0.402 ± 0.011

Completeness: 0.436 ± 0.011

V-measure: 0.418 ± 0.008

Adjusted Rand-Index: 0.320 ± 0.016

Silhouette coefficient: 0.032 ± 0.002

Clustering done in 0.20 ± 0.05 s

Homogeneity: 0.334 ± 0.072

Completeness: 0.338 ± 0.073

V-measure: 0.336 ± 0.073

Adjusted Rand-Index: 0.302 ± 0.090

Silhouette coefficient: 0.025 ± 0.004

```

3. Test case 3: plot\_hashing\_vs\_dict\_vectorizer

Run the use case:

```bash

python plot\_hashing\_vs\_dict\_vectorizer.py

```

Operation results:

```

Loading 20 newsgroups training data

3803 documents - 6.245MB

done in 6.216 s at 1.0 MB/s

Found 47928 unique terms

done in 3.101 s at 2.0 MB/s

Found 43873 unique tokens

done in 2.784 s at 2.2 MB/s

Found 47668 unique tokens

done in 2.766 s at 2.3 MB/s

Found 43873 unique tokens

done in 3.821 s at 1.6 MB/s

Found 47885 unique terms

done in 2.323 s at 2.7 MB/s

done in 2.911 s at 2.1 MB/s

Found 47885 unique terms

```

### tree

\*\*Enter directory:\*\*

```

cd scikit-learn/examples/tree

```

1. Test case 1: plot\_cost\_complexity\_pruning

Run the use case:

```bash

python plot\_cost\_complexity\_pruning.py

```

Operation results:

```

Number of nodes in the last tree is: 1 with ccp\_alpha: 0.3272984419327777

```

2. Test case 2: plot\_iris\_dtc

Run the use case:

```bash

python plot\_iris\_dtc.py

```

Operation results:

```

NULL

```

3. Test case 3: plot\_tree\_regression\_multioutput

Run the use case:

```bash

python plot\_tree\_regression\_multioutput.py

```

Operation results:

```

NULL

```

4. Test case 4: plot\_tree\_regression

Run the use case:

```bash

python plot\_tree\_regression.py

```

Operation results:

```

NULL

```

5. Test case 5: plot\_unveil\_tree\_structure

Run the use case:

```bash

python plot\_unveil\_tree\_structure.py

```

Operation results:

```

The binary tree structure has 5 nodes and has the following tree structure:

node=0 is a split node with value=[[0.33035714 0.30357143 0.36607143]]: go to node 1 if X[:, 3] <= 0.800000011920929 else to node 2.

node=1 is a leaf node with value=[[1. 0. 0.]].

node=2 is a split node with value=[[0. 0.45333333 0.54666667]]: go to node 3 if X[:, 2] <= 4.950000047683716 else to node 4.

node=3 is a leaf node with value=[[0. 0.91666667 0.08333333]].

node=4 is a leaf node with value=[[0. 0.02564103 0.97435897]].

Rules used to predict sample 0:

decision node 0: (X\_test[0, 3] = 2.4) > 0.800000011920929)

decision node 2: (X\_test[0, 2] = 5.1) > 4.950000047683716)

The following samples [0, 1] share the node(s) [0 2] in the tree.

This is 40.0% of all nodes.

```

# sklearn/tests

> Test cases in the `scikit-learn/sklearn/tests` directory

The following are special use cases:

-test\_docstrings.py

This test case is used to check whether the script document string conforms to the regulations. To run it, you need to provide an import\_path, which is the import path of the Python module to be tested. This path will point to the Python module containing the document string.

## Test items

\*\*Enter this directory:\*\*

```

cd scikit-learn/sklearn/tests

```

1. Test case 1: metadata\_routing\_common

Run the use case:

```bash

python metadata\_routing\_common.py

```

Operation results:

```

NULL

```

2. Test case 2: random\_seed

Run the use case:

```bash

python random\_seed.py

```

Operation results:

```

NULL

```

3. Test case three: test\_base

Run the use case:

```bash

python test\_base.py

```

Operation results:

```

NULL

```

4. Test case 4: test\_build

Run the use case:

```bash

python test\_build.py

```

Operation results:

```

NULL

```

5. Test case 5: test\_calibration

Run the use case:

```bash

python test\_calibration.py

```

Operation results:

```

NULL

```

6. Test case six: test\_check\_build

Run the use case:

```bash

python test\_check\_build.py

```

Operation results:

```

NULL

```

7. Test case seven: test\_common

Run the use case:

```bash

python test\_common.py

```

Operation results:

```

NULL

```

8. Test case eight: test\_config

Run the use case:

```bash

python test\_config.py

```

Operation results:

```

NULL

```

9. Test case nine: test\_discriminant\_analysis

Run the use case:

```bash

python test\_discriminant\_analysis.py

```

Operation results:

```

NULL

```

10. Test case 10: test\_docstring\_parameters

Run the use case:

```bash

python test\_docstring\_parameters.py

```

Operation results:

```

NULL

```

11. Test case 11: test\_docstrings

Run the use case:

```bash

python test\_docstrings.py

```

Operation results:

```

test\_docstrings.py: error: the following arguments are required: import\_path

```

12. Test case 12: test\_dummy

Run the use case:

```bash

python test\_dummy.py

```

Operation results:

```

NULL

```

13. Test case 13: test\_init

Run the use case:

```bash

python test\_init.py

```

Operation results:

```

NULL

```

14. Test case 14: test\_isotonic

Run the use case:

```bash

python test\_isotonic.py

```

Operation results:

```

NULL

```

15. Test case 15: test\_kernel\_approximation

Run the use case:

```bash

python test\_kernel\_approximation.py

```

Operation results:

```

NULL

```

16. Test case 16: test\_kernel\_ridge

Run the use case:

```bash

python test\_kernel\_ridge.py

```

Operation results:

```

NULL

```

17. Test case 17: test\_metadata\_routing

Run the use case:

```bash

python test\_metadata\_routing.py

```

Operation results:

```

NULL

```

18. Test case 18: test\_metaestimators\_metadata\_routing

Run the use case:

```bash

python test\_metaestimators\_metadata\_routing.py

```

Operation results:

```

NULL

```

19. Test case 19: test\_metaestimators

Run the use case:

```bash

python test\_metaestimators.py

```

Operation results:

```

NULL

```

20. Test case 20: test\_min\_dependencies\_readme

Run the use case:

```bash

python test\_min\_dependencies\_readme.py

```

Operation results:

```

NULL

```

21. Test case 21: test\_multiclass

Run the use case:

```bash

python test\_multiclass.py

```

Operation results:

```

NULL

```

22. Test case 22: test\_multioutput

Run the use case:

```bash

python test\_multioutput.py

```

Operation results:

```

NULL

```

23. Test case 23: test\_naive\_bayes

Run the use case:

```bash

python test\_naive\_bayes.py

```

Operation results:

```

NULL

```

24. Test case 24: test\_pipeline

Run the use case:

```bash

python test\_pipeline.py

```

Operation results:

```

NULL

```

25. Test case 25: test\_public\_functions

Run the use case:

```bash

python test\_public\_functions.py

```

Operation results:

```

NULL

```

26. Test case 26: test\_random\_projection

Run the use case:

```bash

python test\_random\_projection.py

```

Operation results:

```

NULL

```

#Perf performance analysis

##test.sh

\*\*<u>This script is used to test the performance data of a single instruction and print the perf data to the screen</u>\*\*

Since only one test item is tested, the data will not be saved.

\*\*This script has one parameter:\*\*

- The parameter is the test item, for example, execute the test case "python plot\_cyclical\_feature\_engineering.py"

\*\*Use Case:\*\*

Run the `plot\_cyclical\_feature\_engineering` test

```

sudo bash path/test.sh "python plot\_cyclical\_feature\_engineering.py"

```

\*\*Operation results:\*\*

```

Categorical features: ['season', 'holiday', 'workingday', 'weather']

Mean model.n\_iter\_ = 100.0

Mean Absolute Error: 0.044 +/- 0.003

Root Mean Squared Error: 0.068 +/- 0.005

Mean model.alpha\_ = 2.7298221281347037

Mean Absolute Error: 0.142 +/- 0.014

Root Mean Squared Error: 0.184 +/- 0.020

Mean Absolute Error: 0.099 +/- 0.011

Root Mean Squared Error: 0.131 +/- 0.011

Mean Absolute Error: 0.125 +/- 0.014

Root Mean Squared Error: 0.166 +/- 0.020

Mean Absolute Error: 0.097 +/- 0.011

Root Mean Squared Error: 0.132 +/- 0.013

Mean Absolute Error: 0.078 +/- 0.009

Root Mean Squared Error: 0.104 +/- 0.009

Mean Absolute Error: 0.053 +/- 0.002

Root Mean Squared Error: 0.076 +/- 0.004

Mean Absolute Error: 0.082 +/- 0.006

Root Mean Squared Error: 0.111 +/- 0.011

147764000000

142377.34

284735000000

265530000000

13609628611

13609628778

22978223447

358011045

86958085900

861606093

2351702003

13609629358

0.932

```

## asv\_test.sh

\*\*<u>This script automatically executes a series of asv commands, covering benchmark tests of multiple modules such as cluster, decomposition, ensemble, etc.</u>\*\*

The perf data will be organized and saved in a text file `cumulative\_performance.txt`

\*\*This script has no parameters\*\*

\*\*Use Case:\*\*

1. Enter the `asv\_benchmarks` directory. After installing asv, you can use the provided patch file `asv\_conf.patch` to patch `asv.conf.json`:

```

cd scikit-learn/asv\_benchmarks

patch asv.conf.json < asv\_conf.patch

```

2. After patching `asv.conf.json`, you can use the script to test the modules in the `asv\_benchmark/benchmarks` directory:

```

sudo bash asv\_test.sh

```

The perf data results will be saved to `cumulative\_performance.txt` in this directory

\*\*cumulative\_performance.txt results:\*\*

The following is part of the file

```

Running:SKLBENCH\_NJOBS=[1,4] asv run -b cluster HEAD^!

3483403584858

6851315.47

13697974103063

6724843200182

197576000000

197576000000

1069770000000

39764937636

1746769114976

52947846658

65379573864

197576000000

0.49

---------------------------------------------------------

Running:SKLBENCH\_NJOBS=[1,4] asv run -b decomposition HEAD^!

1587890000000

1599161.55

3196840000000

3743740000000

113310000000

113310000000

250372000000

8607500070

1424860000000

9015552321

16751070658

113310000000

1.171

--------------------------------------------------------

```

## perf\_test.sh

\*\*<u>This script executes all .py files in the current directory and executes the perf command at the same time to obtain performance data and print it to the screen</u>\*\*

The number of successful, failed, and skipped file executions will also be counted.

\*\*This script can have parameters:\*\*

- When there is no parameter, execute all py examples

- With parameters, you can specify the test cases to be skipped. For example, you can use the following command:

```

// Skip a.py and b.py

bash path/perf\_test.sh a.py b.py

```

\*\*Use Case:\*\*

1. Go to the test case folder, for example, the folder `scikit-learn/benchmarks`

```

cd scikit-learn/benchmarks

```

2. Execute the script

```

sudo bash perf\_test.sh

```

\*\*Operation results:\*\*

The following is part of the output

```

-----------------------------------------------------

Executing ./bench\_mnist.py...

-----------------------------------------------------

------------------perf information--------------------

610970000000

610658.58

1221290000000

986363000000

51694665510

51694665704

86349703924

1024595643

340912000000

6387316807

17721794938

51694666102

0.807

Finished executing ./bench\_mnist.py.

All Python scripts have been executed.

Total:

Success:

Failure:

```

## special\_test.sh

This script is based on the previously written test.sh script and mainly tests the use cases in the `scikit-learn/benchmarks` directory that require additional parameters and outputs these data to the screen.

\*\*This script has one fixed parameter:\*\*

- The parameter is the path to the script `test.sh`

\*\*Use Case:\*\*

Enter the scikit-learn/benchmarks directory

1. Install test dependencies:

```

cd scikit-learn/benchmarks

pip install pytest matplotlib pandas memory\_profiler pooch seaborn plotly jinja2 pip install scikit-image

```

2. Use the `perf\_test.sh` script to test:

```

cd benchmarks

// Skip bench\_20newsgroups.py, bench\_covertype.py, and bench\_isotonic.py test cases

sudo bash path/perf\_test.sh bench\_20newsgroups.py bench\_covertype.py bench\_isotonic.py

```

Test results and perf data will be output to the screen

3. The special cases are all from the directory scikit-learn/benchmarks, and then the perf\_test.sh script is used to test the directory:

```

cd scikit-learn/benchmarks

sudo bash path/special\_test.sh path/test.sh

```