Crafting a minimal reproducer for scikit-learn

Whether submitting a bug report, designing a suite of tests, or simply posting a question in the discussions, being able to craft minimal, reproducible examples (or minimal, workable examples) is the key to communicating effectively and efficiently with the community.

There are very good guidelines on the internet such as this StackOverflow document or this blogpost by Matthew Rocklin on crafting Minimal Complete Verifiable Examples (referred below as MCVE). Our goal is not to be repetitive with those references but rather to provide a step-by-step guide on how to narrow down a bug until you have reached the shortest possible code to reproduce it.

The first step before submitting a bug report to scikit-learn is to read the Issue template. It is already quite informative about the information you will be asked to provide.

Good practices

In this section we will focus on the **Steps/Code to Reproduce** section of the Issue template. We will start with a snippet of code that already provides a failing example but that has room for readability improvement. We then craft a MCVE from it.

Example

```
# I am currently working in a ML project and when I tried to fit a
# GradientBoostingRegressor instance to my_data.csv I get a UserWarning:
# "X has feature names, but DecisionTreeRegressor was fitted without
# feature names". You can get a copy of my dataset from
# https://example.com/my_data.csv and verify my features do have
# names. The problem seems to arise during fit when I pass an integer
# to the n_iter_no_change parameter.

df = pd.read_csv('my_data.csv')
X = df[["feature_name"]] # my features do have names
y = df["target"]

# We set random_state=42 for the train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42
)

scaler = StandardScaler(with_mean=False)
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# An instance with default n_iter_no_change raises no error nor warnings
gbdt = GradientBoostingRegressor(random_state=0)
gbdt.fit(X_train, y_train)
default_score = gbdt.score(X_test, y_test)

# the bug appears when I change the value for n_iter_no_change
gbdt = GradientBoostingRegressor(random_state=0, n_iter_no_change=5)
gbdt.fit(X_train, y_train)
other_score = gbdt.score(X_test, y_test)

other_score = gbdt.score(X_test, y_test)
```

Provide a failing code example with minimal comments

Writing instructions to reproduce the problem in English is often ambiguous. Better make sure that all the necessary details to reproduce the problem are illustrated in the Python code snippet to avoid any ambiguity. Besides, by this point you already provided a concise description in the **Describe the bug** section of the Issue template.

The following code, while **still not minimal**, is already **much better** because it can be copy-pasted in a Python terminal to reproduce the problem in one step. In particular:

- it contains **all necessary imports statements**;
- it can fetch the public dataset without having to manually download a file and put it in the expected location on the disk.

Improved example

```
import pandas as pd

df = pd.read_csv("https://example.com/my_data.csv")
X = df[["feature_name"]]
y = df["target"]

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42
)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler(with_mean=False)
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.ensemble import GradientBoostingRegressor

gbdt = GradientBoostingRegressor(random_state=0)
gbdt.fit(X_train, y_train)  # no warning
default_score = gbdt.score(X_test, y_test)

gbdt = GradientBoostingRegressor(random_state=0, n_iter_no_change=5)
gbdt.fit(X_train, y_train)  # raises warning
other_score = gbdt.score(X_test, y_test)
other_score = gbdt.score(X_test, y_test)
```

Boil down your script to something as small as possible

You have to ask yourself which lines of code are relevant and which are not for reproducing the bug. Deleting unnecessary lines of code or simplifying the function calls by omitting unrelated non-default options will help you and other contributors narrow down the cause of the bug.

In particular, for this specific example:

- the warning has nothing to do with the *train_test_split* since it already appears in the training step, before we use the test set.
- similarly, the lines that compute the scores on the test set are not necessary;
- the bug can be reproduced for any value of *random_state* so leave it to its default;
- the bug can be reproduced without preprocessing the data with the *StandardScaler*.

Improved example

```
import pandas as pd
df = pd.read_csv("https://example.com/my_data.csv")
X = df[["feature_name"]]
y = df["target"]

from sklearn.ensemble import GradientBoostingRegressor

gbdt = GradientBoostingRegressor()
gbdt.fit(X, y)  # no warning

gbdt = GradientBoostingRegressor(n_iter_no_change=5)
gbdt.fit(X, y)  # raises warning
```

DO NOT report your data unless it is extremely necessary

The idea is to make the code as self-contained as possible. For doing so, you can use a :ref:`synth_data`. It can be generated using numpy, pandas or the :mod:`sklearn.datasets` module. Most of the times the bug is not related to a particular structure of your data. Even if it is, try to find an available dataset that has similar characteristics to yours and that reproduces the problem. In this particular case, we are interested in data that has labeled feature names.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]minimal_reproducer.rst, line 169); *backlink***
>
> Unknown interpreted text role "ref".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]minimal_reproducer.rst, line 169); *backlink***
>
> Unknown interpreted text role "mod".

Improved example

```
import pandas as pd
from sklearn.ensemble import GradientBoostingRegressor

df = pd.DataFrame(
    {
        "feature_name": [-12.32, 1.43, 30.01, 22.17],
```

```
        "target": [72, 55, 32, 43],
    }
)
X = df[["feature_name"]]
y = df["target"]

gbdt = GradientBoostingRegressor()
gbdt.fit(X, y) # no warning
gbdt = GradientBoostingRegressor(n_iter_no_change=5)
gbdt.fit(X, y) # raises warning
```

As already mentioned, the key to communication is the readability of the code and good formatting can really be a plus. Notice that in the previous snippet we:

- try to limit all lines to a maximum of 79 characters to avoid horizontal scrollbars in the code snippets blocks rendered on the GitHub issue;
- use blank lines to separate groups of related functions;
- place all the imports in their own group at the beginning.

The simplification steps presented in this guide can be implemented in a different order than the progression we have shown here. The important points are:

- a minimal reproducer should be runnable by a simple copy-and-paste in a python terminal;
- it should be simplified as much as possible by removing any code steps that are not strictly needed to reproducing the original problem;
- it should ideally only rely on a minimal dataset generated on-the-fly by running the code instead of relying on external data, if possible.

Use markdown formatting

To format code or text into its own distinct block, use triple backticks. Markdown supports an optional language identifier to enable syntax highlighting in your fenced code block. For example:

````
```python
from sklearn.datasets import make_blobs

n_samples = 100
n_components = 3
X, y = make_blobs(n_samples=n_samples, centers=n_components)
```
````

will render a python formatted snippet as follows

```python
from sklearn.datasets import make_blobs

n_samples = 100
n_components = 3
X, y = make_blobs(n_samples=n_samples, centers=n_components)
```

It is not necessary to create several blocks of code when submitting a bug report. Remember other reviewers are going to copy-paste your code and having a single cell will make their task easier.

In the section named **Actual results** of the Issue template you are asked to provide the error message including the full traceback of the exception. In this case, use the *python-traceback* qualifier. For example:

````
```python-traceback
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-a674e682c281> in <module>
    4 vectorizer = CountVectorizer(input=docs, analyzer='word')
    5 lda_features = vectorizer.fit_transform(docs)
----> 6 lda_model = LatentDirichletAllocation(
    7     n_topics=10,
    8     learning_method='online',

TypeError: __init__() got an unexpected keyword argument 'n_topics'
```
````

yields the following when rendered:

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-a674e682c281> in <module>
    4 vectorizer = CountVectorizer(input=docs, analyzer='word')
    5 lda_features = vectorizer.fit_transform(docs)
----> 6 lda_model = LatentDirichletAllocation(
    7     n_topics=10,
```

```
    8     learning_method='online',

TypeError: __init__() got an unexpected keyword argument 'n_topics'
```

Synthetic dataset

Before choosing a particular synthetic dataset, first you have to identify the type of problem you are solving: Is it a classification, a regression, a clustering, etc?

Once that you narrowed down the type of problem, you need to provide a synthetic dataset accordingly. Most of the times you only need a minimalistic dataset. Here is a non-exhaustive list of tools that may help you.

NumPy

NumPy tools such as numpy.random.randn and numpy.random.randint can be used to create dummy numeric data.

- regression

  Regressions take continuous numeric data as features and target.

  ```
  import numpy as np

  rng = np.random.RandomState(0)
  n_samples, n_features = 5, 5
  X = rng.randn(n_samples, n_features)
  y = rng.randn(n_samples)
  ```

A similar snippet can be used as synthetic data when testing scaling tools such as :class:`sklearn.preprocessing.StandardScaler`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]minimal_reproducer.rst, line 319); *backlink***
>
> Unknown interpreted text role "class".

- classification

  If the bug is not raised during when encoding a categorical variable, you can feed numeric data to a classifier. Just remember to ensure that the target is indeed an integer.

  ```
  import numpy as np

  rng = np.random.RandomState(0)
  n_samples, n_features = 5, 5
  X = rng.randn(n_samples, n_features)
  y = rng.randint(0, 2, n_samples)  # binary target with values in {0, 1}
  ```

  If the bug only happens with non-numeric class labels, you might want to generate a random target with numpy.random.choice.

  ```
  import numpy as np

  rng = np.random.RandomState(0)
  n_samples, n_features = 50, 5
  X = rng.randn(n_samples, n_features)
  y = np.random.choice(
      ["male", "female", "other"], size=n_samples, p=[0.49, 0.49, 0.02]
  )
  ```

Pandas

Some scikit-learn objects expect pandas dataframes as input. In this case you can transform numpy arrays into pandas objects using pandas.DataFrame, or pandas.Series.

```
import numpy as np
import pandas as pd

rng = np.random.RandomState(0)
n_samples, n_features = 5, 5
X = pd.DataFrame(
    {
        "continuous_feature": rng.randn(n_samples),
        "positive_feature": rng.uniform(low=0.0, high=100.0, size=n_samples),
        "categorical_feature": rng.choice(["a", "b", "c"], size=n_samples),
    }
)
```

```
y = pd.Series(rng.randn(n_samples))
```

In addition, scikit-learn includes various :ref:`sample_generators` that can be used to build artificial datasets of controlled size and complexity.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]minimal_reproducer.rst, line 378); *backlink***
>
> Unknown interpreted text role "ref".

make_regression

As hinted by the name, :class:`sklearn.datasets.make_regression` produces regression targets with noise as an optionally-sparse random linear combination of random features.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]minimal_reproducer.rst, line 384); *backlink***
>
> Unknown interpreted text role "class".

```
from sklearn.datasets import make_regression

X, y = make_regression(n_samples=1000, n_features=20)
```

make_classification

:class:`sklearn.datasets.make_classification` creates multiclass datasets with multiple Gaussian clusters per class. Noise can be introduced by means of correlated, redundant or uninformative features.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]minimal_reproducer.rst, line 397); *backlink***
>
> Unknown interpreted text role "class".

```
from sklearn.datasets import make_classification

X, y = make_classification(
    n_features=2, n_redundant=0, n_informative=2, n_clusters_per_class=1
)
```

make_blobs

Similarly to *make_classification*, :class:`sklearn.datasets.make_blobs` creates multiclass datasets using normally-distributed clusters of points. It provides greater control regarding the centers and standard deviations of each cluster, and therefore it is useful to demonstrate clustering.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]minimal_reproducer.rst, line 412); *backlink***
>
> Unknown interpreted text role "class".

```
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=10, centers=3, n_features=2)
```

Dataset loading utilities

You can use the :ref:`datasets` to load and fetch several popular reference datasets. This option is useful when the bug relates to the particular structure of the data, e.g. dealing with missing values or image recognition.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]minimal_reproducer.rst, line 426); *backlink***
>
> Unknown interpreted text role "ref".

```
from sklearn.datasets import load_breast_cancer

X, y = load_breast_cancer(return_X_y=True)
```