pca-22mcb1002

April 28, 2023

[1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot
from statistics import mean
from statistics import stdev
from sklearn.datasets import make_classification
```

[2]:
```
# Creating random dataset using make_classification
def random_dataset():
  X, y = make_classification(n_samples = 1000, n_features = 20,
                             n_informative = 15, n_redundant = 5, random_state␣
 ↪= 7)
  return X,y
```

[3]:
```
X,y = random_dataset()
print(X.shape, y.shape)
```

```
(1000, 20) (1000,)
```

[4]:
```
print(X, y)
```

```
[[ 0.2929949  -4.21223056 -1.288332   … -4.43170633 -2.82646737
   0.44916808]
 [-0.06839901  5.51884147 11.2389773  … -3.08994781  1.19029898
   1.62025622]
 [ 0.73161622 -0.68468633 -0.98174194 …  5.65429655 -0.64659866
  -3.15652999]
 …
 [ 0.81230832  0.29333773  3.55727154 …  7.52278375 -4.50067701
  -1.92525878]
 [ 2.62760166 -1.9607565  -7.1050466  …  0.02433393 -0.77573778
   4.04660465]
 [-0.97292653  0.76166769  3.98307684 …  0.85864477  2.406057
   2.33338943]] [1 1 1 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1 0 0 1 0 0 1 1 0 1 0 1 1 1 0
1 0 0 1 0
 1 0 1 0 1 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 1 1 0 0
 0 1 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 1 1 0 1 1 0 0 1 1 0 1 0
 0 0 1 1 1 1 0 0 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 1 0 1 1 0 1 0
```

```
1 1 1 0 1 1 0 1 1 0 1 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1
0 0 1 1 0 1 1 1 0 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0 0
1 1 1 0 0 0 1 1 0 0 1 0 1 1 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 1 1 1 0 0 0 0 1
0 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0 0 1
0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 1 0 0
1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 1 1 1 0 1 1 0 1 1 0 1 0 1 0 1 1 1 0 0 0 0 1
0 1 1 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 1 0 0 1 1 0
0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0
1 0 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 0
0 0 0 0 0 1 0 1 1 1 1 0 1 1 1 0 0 0 1 0 0 1 1 0 1 1 1 0 0 1 1 0 1 0 0 0 0
0 1 1 1 0 1 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 0
1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0
0 1 0 1 0 0 1 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 1
0 1 1 1 1 1 1 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 0 1 1
0 0 1 0 0 1 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 1 1 0 1 1 1 0 1 1 1 1 0 1
1 0 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0 1 1 0 1 1 1 0 1 0 1 0 1 0 1
0 0 0 1 1 0 1 1 0 0 0 0 1 1 1 0 1 1 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0
0 1 1 0 0 1 1 0 0 1 0 1 1 0 0 0 1 0 1 0 1 1 1 1 0 0 1 1 1 1 1 0 0 0 1 0 0
0 0 0 1 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1 0 1 0 0 1 1 1 0 1 0 0 0 0 0 1 1 1
1 0 1 1 1 0 1 0 1 1 0 1 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 1 1 1 1 0
0 0 0 0 1 1 0 1 0 0 1 1 1 0 1 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1
0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0 0
1]
```

[5]:
```
# Defining pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

[6]:
```
# Defining steps as a list of tuples
steps = [('PCA:', PCA(n_components = 10)), ('m', LogisticRegression())]
models  = Pipeline(steps = steps)
```

[7]:
```
# Evaluation of models
cv = RepeatedStratifiedKFold(n_splits = 10, n_repeats = 3, random_state = 1)
n_scores = cross_val_score(models, X, y, cv = cv, n_jobs = -1, scoring =␣
 ↪'accuracy')
```

[8]:
```
print('Accuracy: %.3f (%.3f)'%(mean(n_scores), stdev(n_scores)))
```

```
Accuracy: 0.816 (0.035)
```

[9]:
```
def create_models():
  models = dict()
  for i in range(1,21):
    steps = [('PCA', PCA(n_components = 10)), ('m', LogisticRegression())]
    models = Pipeline(steps = steps)
  return models
```

[20]:
```
def evaluation_model(models, X, y):
  cv = RepeatedStratifiedKFold(n_splits = 10, n_repeats = 3, random_state = 1)
  scores = cross_val_score(models, X, y, cv = cv, n_jobs = -1, scoring =␣
 ↪'accuracy')
  return scores
```

[11]:
```
models = create_models()
```

[12]:
```
sampledict = {'one':1, 'two':2, 'three':3}
```

[13]:
```
sampledict
```

[13]:
```
{'one': 1, 'two': 2, 'three': 3}
```

[14]:
```
list(sampledict.keys())
```

[14]:
```
['one', 'two', 'three']
```

[15]:
```
list(sampledict.values())
```

[15]:
```
[1, 2, 3]
```

[16]:
```
list(sampledict.items())
```

[16]:
```
[('one', 1), ('two', 2), ('three', 3)]
```

[17]:
```
for k,v in list(sampledict.items()):
  print(k,v)
```

```
one 1
two 2
three 3
```

[24]:
```
models = create_models()
models
```

[24]:
```
Pipeline(steps=[('PCA', PCA(n_components=10)), ('m', LogisticRegression())])
```

[34]:
```
results, names = list(), list()
for name, model in models.items():
```

```
    scores = evaluation_model(model, X, y)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), stdev(scores)))
```

```
---------------------------------------------------------------------------
Empty                                     Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in␣
 ↪dispatch_one_batch(self, iterator)
    861             try:
--> 862                 tasks = self._ready_batches.get(block=False)
    863             except queue.Empty:

/usr/lib/python3.10/queue.py in get(self, block, timeout)
    167                 if not self._qsize():
--> 168                     raise Empty
    169             elif timeout is None:

Empty:

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-34-9f0949cf24b7> in <cell line: 2>()
      1 results, names = list(), list()
      2 for name, model in models.items():
----> 3     scores = evaluation_model(model, X, y)
      4     results.append(scores)
      5     names.append(name)

<ipython-input-20-6d5d203a9548> in evaluation_model(models, X, y)
      1 def evaluation_model(models, X, y):
      2   cv = RepeatedStratifiedKFold(n_splits = 10, n_repeats = 3,␣
 ↪random_state = 1)
----> 3   scores = cross_val_score(models, X, y, cv = cv, n_jobs = -1, scoring␣
 ↪'accuracy')
      4   return scores

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py␣
 ↪in cross_val_score(estimator, X, y, groups, scoring, cv, n_jobs, verbose,␣
 ↪fit_params, pre_dispatch, error_score)
    513     scorer = check_scoring(estimator, scoring=scoring)
    514
--> 515     cv_results = cross_validate(
    516         estimator=estimator,
    517         X=X,
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py␣
 ↪in cross_validate(estimator, X, y, groups, scoring, cv, n_jobs, verbose,␣
 ↪fit_params, pre_dispatch, return_train_score, return_estimator, error_score)
    264     # independent, and that it is pickle-able.
    265     parallel = Parallel(n_jobs=n_jobs, verbose=verbose,␣
 ↪pre_dispatch=pre_dispatch)
--> 266     results = parallel(
    267         delayed(_fit_and_score)(
    268             clone(estimator),

/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py in␣
 ↪__call__(self, iterable)
     61             for delayed_func, args, kwargs in iterable
     62         )
---> 63         return super().__call__(iterable_with_config)
     64
     65

/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in __call__(self,␣
 ↪iterable)
   1083             # remaining jobs.
   1084             self._iterating = False
-> 1085             if self.dispatch_one_batch(iterator):
   1086                 self._iterating = self._original_iterator is not None
   1087

/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in␣
 ↪dispatch_one_batch(self, iterator)
    871                 big_batch_size = batch_size * n_jobs
    872
--> 873                 islice = list(itertools.islice(iterator, big_batch_size ))
    874                 if len(islice) == 0:
    875                     return False

/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py in <genexpr>(
 ↪0)
     57         # pre_dispatch and n_jobs.
     58         config = get_config()
---> 59         iterable_with_config = (
     60             (_with_config(delayed_func, config), args, kwargs)
     61             for delayed_func, args, kwargs in iterable

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py␣
 ↪in <genexpr>(.0)
    264     # independent, and that it is pickle-able.
    265     parallel = Parallel(n_jobs=n_jobs, verbose=verbose,␣
 ↪pre_dispatch=pre_dispatch)
```

```
--> 266     results = parallel(
    267         delayed(_fit_and_score)(
    268             clone(estimator),

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py in␣
 ↪split(self, X, y, groups)
   1474         for idx in range(n_repeats):
   1475             cv = self.cv(random_state=rng, shuffle=True, **self.cvargs)
-> 1476             for train_index, test_index in cv.split(X, y, groups):
   1477                 yield train_index, test_index
   1478

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py in␣
 ↪split(self, X, y, groups)
    350             )
    351
--> 352         for train, test in super().split(X, y, groups):
    353             yield train, test
    354

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py in␣
 ↪split(self, X, y, groups)
     83         X, y, groups = indexable(X, y, groups)
     84         indices = np.arange(_num_samples(X))
---> 85         for test_index in self._iter_test_masks(X, y, groups):
     86             train_index = indices[np.logical_not(test_index)]
     87             test_index = indices[test_index]

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py in␣
 ↪_iter_test_masks(self, X, y, groups)
    731
    732     def _iter_test_masks(self, X, y=None, groups=None):
--> 733         test_folds = self._make_test_folds(X, y)
    734         for i in range(self.n_splits):
    735             yield test_folds == i

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py in␣
 ↪_make_test_folds(self, X, y)
    674         allowed_target_types = ("binary", "multiclass")
    675         if type_of_target_y not in allowed_target_types:
--> 676             raise ValueError(
    677                 "Supported target types are: {}. Got {!r} instead.".
 ↪format(
    678                     allowed_target_types, type_of_target_y
```

```
ValueError: Supported target types are: ('binary', 'multiclass'). Got␣
 ↪'continuous' instead.
```

[ ]:
```
pyplot.boxplot(results, labels = names, showmeans = True)
pyplot.xticks(rotation = 45)
pyplot.show()
```

[ ]:
```
#Making predictions
#define steps with 15 PC
steps=[('pca',PCA(n_components=15)),('m',LogisticRegression())]
models=Pipeline(steps=steps)
```

[ ]:
```
models.fit(X,y)
Pipeline(memory=None, steps=[('pca', PCA(copy=True, iterated_power='auto',␣
 ↪n_components=15,
                                          random_state=None, svd_solver='auto',␣
 ↪tol=0.0, whiten=False)),
                             ('m', LogisticRegression(C=1.0, class_weight=None,␣
 ↪dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None,␣
 ↪max_iter=100,
                                                      multi_class='auto',␣
 ↪n_jobs=None, penalty='l2', random_state=None, solver='lbfgs', tol=0.0001,␣
 ↪verbose=0,warm_start=False))], verbose=False)
row = [[0.2929949, -4.21223056, -1.288332, -2.17849815, -0.64527665, 2.58097719,
0.28422388, -7.1827928, -1.91211104, 2.73729512, 0.81395695, 3.96973717, -2.
 ↪66939799,
3.34692332, 4.19791821, 0.99990998, -0.30201875, -4.43170633, -2.82646737, 0.
 ↪44916808]]
yhat = models.predict(row)
print('Predicted Class: %d' % yhat[0])
```

[26]:
```
from sklearn.datasets import load_diabetes
```

[27]:
```
diabetes = load_diabetes()
df = pd.DataFrame(data=diabetes.data,columns=diabetes.feature_names)
df.head(6)
```

[27]:

	age	sex	bmi	bp	s1	s2	s3	\
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	
5	-0.092695	-0.044642	-0.040696	-0.019442	-0.068991	-0.079288	0.041277	

	s4	s5	s6

```
0 -0.002592  0.019907 -0.017646
1 -0.039493 -0.068332 -0.092204
2 -0.002592  0.002861 -0.025930
3  0.034309  0.022688 -0.009362
4 -0.002592 -0.031988 -0.046641
5 -0.076395 -0.041176 -0.096346
```

[28]:
```
df.columns
```

[28]:
```
Index(['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6'],
dtype='object')
```

[29]:
```
dataset=diabetes.data
X=dataset[:,:-1]
y=dataset[:,-1]
X.shape,y.shape
```

[29]:
```
((442, 9), (442,))
```

[30]:
```
scaler = StandardScaler()
scaler.fit(df)
Diabetes_scaled = scaler.transform(df)
steps=[('norm',StandardScaler()),('pca',PCA(n_components=10)),('m',LogisticRegression())]
model=Pipeline(steps=steps)
```

[31]:
```
def create_models():
  models=dict()
  for i in range(1,11):
   ␣
 ↪steps=[('norm',StandardScaler()),('pca',PCA(n_components=10)),('m',LogisticRegression())]
    models[str(i)]=Pipeline(steps=steps)
  return models
```

[32]:
```
models=create_models()
models
```

[32]:
```
{'1': Pipeline(steps=[('norm', StandardScaler()), ('pca', PCA(n_components=10)),
                      ('m', LogisticRegression())]),
 '2': Pipeline(steps=[('norm', StandardScaler()), ('pca', PCA(n_components=10)),
                      ('m', LogisticRegression())]),
 '3': Pipeline(steps=[('norm', StandardScaler()), ('pca', PCA(n_components=10)),
                      ('m', LogisticRegression())]),
 '4': Pipeline(steps=[('norm', StandardScaler()), ('pca', PCA(n_components=10)),
                      ('m', LogisticRegression())]),
 '5': Pipeline(steps=[('norm', StandardScaler()), ('pca', PCA(n_components=10)),
                      ('m', LogisticRegression())]),
 '6': Pipeline(steps=[('norm', StandardScaler()), ('pca', PCA(n_components=10)),
```

```
                ('m', LogisticRegression())]),
 '7': Pipeline(steps=[('norm', StandardScaler()), ('pca', PCA(n_components=10)),
                ('m', LogisticRegression())]),
 '8': Pipeline(steps=[('norm', StandardScaler()), ('pca', PCA(n_components=10)),
                ('m', LogisticRegression())]),
 '9': Pipeline(steps=[('norm', StandardScaler()), ('pca', PCA(n_components=10)),
                ('m', LogisticRegression())]),
 '10': Pipeline(steps=[('norm', StandardScaler()), ('pca',
PCA(n_components=10)),
                ('m', LogisticRegression())])}
```

[35]:
```
results,names=list(),list()
for name,model in models.items():
  scores=evaluation_model(model,X,y)
  results.append(scores)
  names.append(name)
  print('#%s %.3f (%.3f)' %(name,mean(scores),stdev(scores)))
```

```
---------------------------------------------------------------------------
Empty                                     Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in␣
 ↪dispatch_one_batch(self, iterator)
    861             try:
--> 862                 tasks = self._ready_batches.get(block=False)
    863             except queue.Empty:

/usr/lib/python3.10/queue.py in get(self, block, timeout)
    167                 if not self._qsize():
--> 168                     raise Empty
    169             elif timeout is None:

Empty:

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-35-8762d805c76d> in <cell line: 2>()
      1 results,names=list(),list()
      2 for name,model in models.items():
----> 3   scores=evaluation_model(model,X,y)
      4   results.append(scores)
      5   names.append(name)

<ipython-input-20-6d5d203a9548> in evaluation_model(models, X, y)
      1 def evaluation_model(models, X, y):
      2   cv = RepeatedStratifiedKFold(n_splits = 10, n_repeats = 3,␣
 ↪random_state = 1)
```

```
----> 3   scores = cross_val_score(models, X, y, cv = cv, n_jobs = -1, scoring =␣
 ↪'accuracy')
      4   return scores

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py␣
 ↪in cross_val_score(estimator, X, y, groups, scoring, cv, n_jobs, verbose,␣
 ↪fit_params, pre_dispatch, error_score)
    513     scorer = check_scoring(estimator, scoring=scoring)
    514
--> 515     cv_results = cross_validate(
    516         estimator=estimator,
    517         X=X,

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py␣
 ↪in cross_validate(estimator, X, y, groups, scoring, cv, n_jobs, verbose,␣
 ↪fit_params, pre_dispatch, return_train_score, return_estimator, error_score)
    264     # independent, and that it is pickle-able.
    265     parallel = Parallel(n_jobs=n_jobs, verbose=verbose,␣
 ↪pre_dispatch=pre_dispatch)
--> 266     results = parallel(
    267         delayed(_fit_and_score)(
    268             clone(estimator),

/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py in␣
 ↪__call__(self, iterable)
     61             for delayed_func, args, kwargs in iterable
     62         )
---> 63         return super().__call__(iterable_with_config)
     64
     65

/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in __call__(self,␣
 ↪iterable)
   1083             # remaining jobs.
   1084             self._iterating = False
-> 1085             if self.dispatch_one_batch(iterator):
   1086                 self._iterating = self._original_iterator is not None
   1087

/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in␣
 ↪dispatch_one_batch(self, iterator)
    871                 big_batch_size = batch_size * n_jobs
    872
--> 873                 islice = list(itertools.islice(iterator, big_batch_size))
    874                 if len(islice) == 0:
    875                     return False
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py in <genexpr>(
 ↪0)
     57         # pre_dispatch and n_jobs.
     58         config = get_config()
---> 59         iterable_with_config = (

     60             (_with_config(delayed_func, config), args, kwargs)
     61             for delayed_func, args, kwargs in iterable

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py␣
 ↪in <genexpr>(.0)
    264     # independent, and that it is pickle-able.
    265     parallel = Parallel(n_jobs=n_jobs, verbose=verbose,␣
 ↪pre_dispatch=pre_dispatch)
--> 266     results = parallel(

    267         delayed(_fit_and_score)(
    268             clone(estimator),

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py in␣
 ↪split(self, X, y, groups)
   1474         for idx in range(n_repeats):
   1475             cv = self.cv(random_state=rng, shuffle=True, **self.cvargs)
-> 1476             for train_index, test_index in cv.split(X, y, groups):
   1477                 yield train_index, test_index
   1478

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py in␣
 ↪split(self, X, y, groups)
    350             )
    351
--> 352         for train, test in super().split(X, y, groups):
    353             yield train, test
    354

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py in␣
 ↪split(self, X, y, groups)
     83         X, y, groups = indexable(X, y, groups)
     84         indices = np.arange(_num_samples(X))
---> 85         for test_index in self._iter_test_masks(X, y, groups):
     86             train_index = indices[np.logical_not(test_index)]
     87             test_index = indices[test_index]

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py in␣
 ↪_iter_test_masks(self, X, y, groups)
    731
    732     def _iter_test_masks(self, X, y=None, groups=None):
--> 733         test_folds = self._make_test_folds(X, y)
    734         for i in range(self.n_splits):
```

```
    735             yield test_folds == i

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py in␣
 ↪_make_test_folds(self, X, y)
    674         allowed_target_types = ("binary", "multiclass")
    675         if type_of_target_y not in allowed_target_types:
--> 676             raise ValueError(
    677                 "Supported target types are: {}. Got {!r} instead.".
 ↪format(
    678                     allowed_target_types, type_of_target_y

ValueError: Supported target types are: ('binary', 'multiclass'). Got␣
 ↪'continuous' instead.
```