Note

Click here to download the full example code

K-nearest neighbors classification

Shows the usage of the k-nearest neighbors classifier.

```
# Author: Pablo Marcos Manchón
# License: MIT

import skfda
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from skfda.ml.classification import KNeighborsClassifier
```

In this example we are going to show the usage of the K-nearest neighbors classifier in their functional version, which is a extension of the multivariate one, but using functional metrics between the observations.

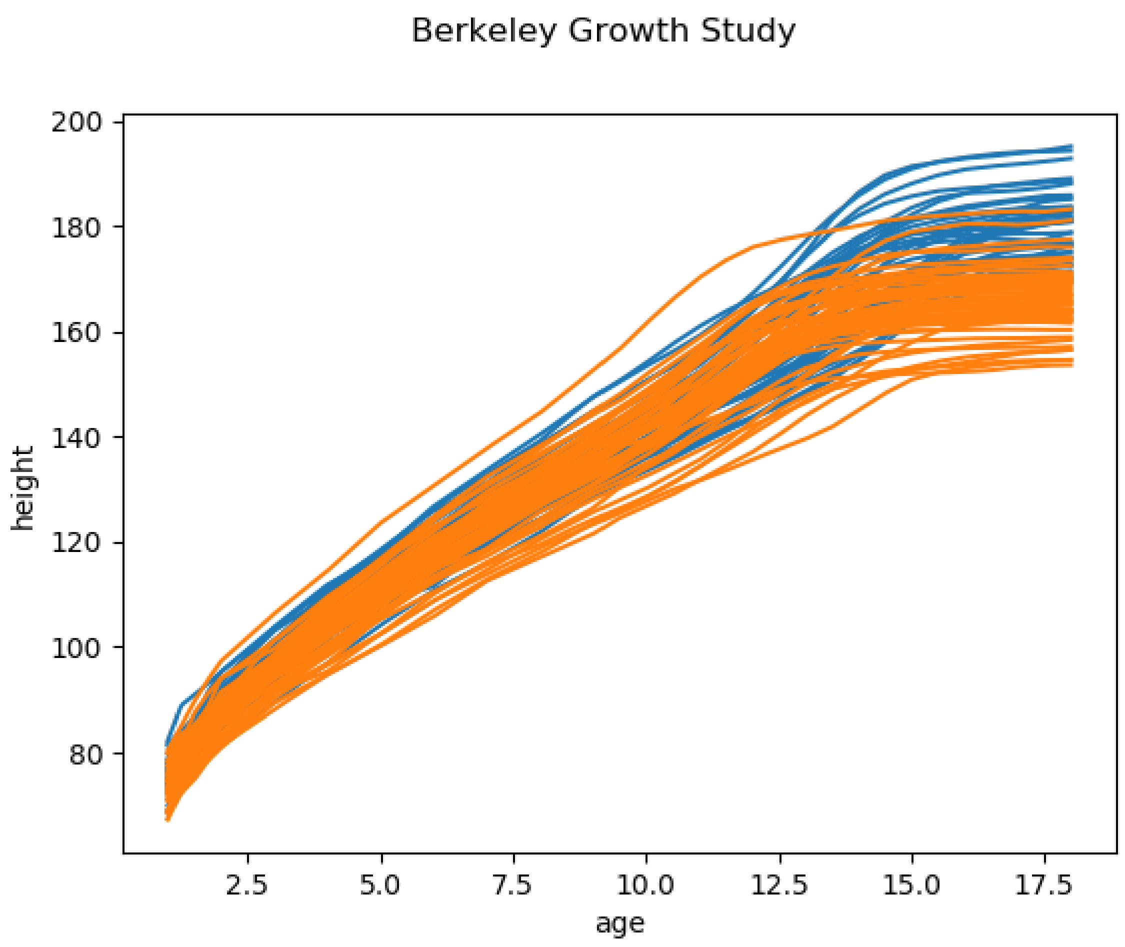
Firstly, we are going to fetch a functional data dataset, such as the Berkeley Growth Study. This dataset correspond to the height of several boys and girls measured until the 18 years of age.

We will try to predict the sex by using its growth curves.

The following figure shows the growth curves grouped by sex.

```
data = skfda.datasets.fetch_growth()
X = data['data']
y = data['target']

X[y==0].plot(color='C0')
X[y==1].plot(color='C1')
```



In this case, the class labels are stored in an array with 0's in the male samples and 1's in the positions with female ones.

```
print(y)
```

Out:

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

We can split the dataset using the sklearn function `train_test_split`.

We will use two thirds of the dataset for the training partition and the remaining samples for testing.

The function will return two `FDataGrid`'s, `X_train` and `X_test` with the corresponding partitions, and arrays with their class labels.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=0)
```

We will fit the classifier `KNeighborsClassifier` with the training partition. This classifier works exactly like the sklearn multivariate classifier `KNeighborsClassifier` , but will accept as input a `FDataGrid` with functional observations instead of an array with multivariate data.

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

Once it is fitted, we can predict labels for the test samples.

To predict the label of a test sample, the classifier will calculate the k-nearest neighbors and will asign the majority class. By default, it is used the $\mathbb{L}^2$ distance between functions, to determine the neighbourhood of a sample, with 5 neighbors.

Can be used any of the functional metrics of the module `skfda.misc.metrics`.

```
pred = knn.predict(X_test)
print(pred)
```

Out:

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

The `score()` method allows us to calculate the mean accuracy for the test data. In this case we obtained around 96% of accuracy.

```
score = knn.score(X_test, y_test)
print(score)
```

Out:

```
0.967741935483871
```

We can also estimate the probability of membership to the predicted class using `predict_proba()`, which will return an array with the probabilities of the classes, in lexicographic order, for each test sample.

```
probs = knn.predict_proba(X_test[:5])
print(probs)
```

Out:

```
[[1.  0. ]
 [0.6 0.4]
 [0.  1. ]
 [0.6 0.4]
 [0.8 0.2]]
```

We can use the sklearn `GridSearchCV` to perform a grid search to select the best hyperparams, using cross-validation.

In this case, we will vary the number of neighbors between 1 and 11.

```
# only odd numbers
param_grid = {'n_neighbors': np.arange(1, 12, 2)}


knn = KNeighborsClassifier()
gscv = GridSearchCV(knn, param_grid, cv=KFold(shuffle=True, random_state=0))
gscv.fit(X, y)


print("Best params:", gscv.best_params_)
print("Best score:", gscv.best_score_)
```

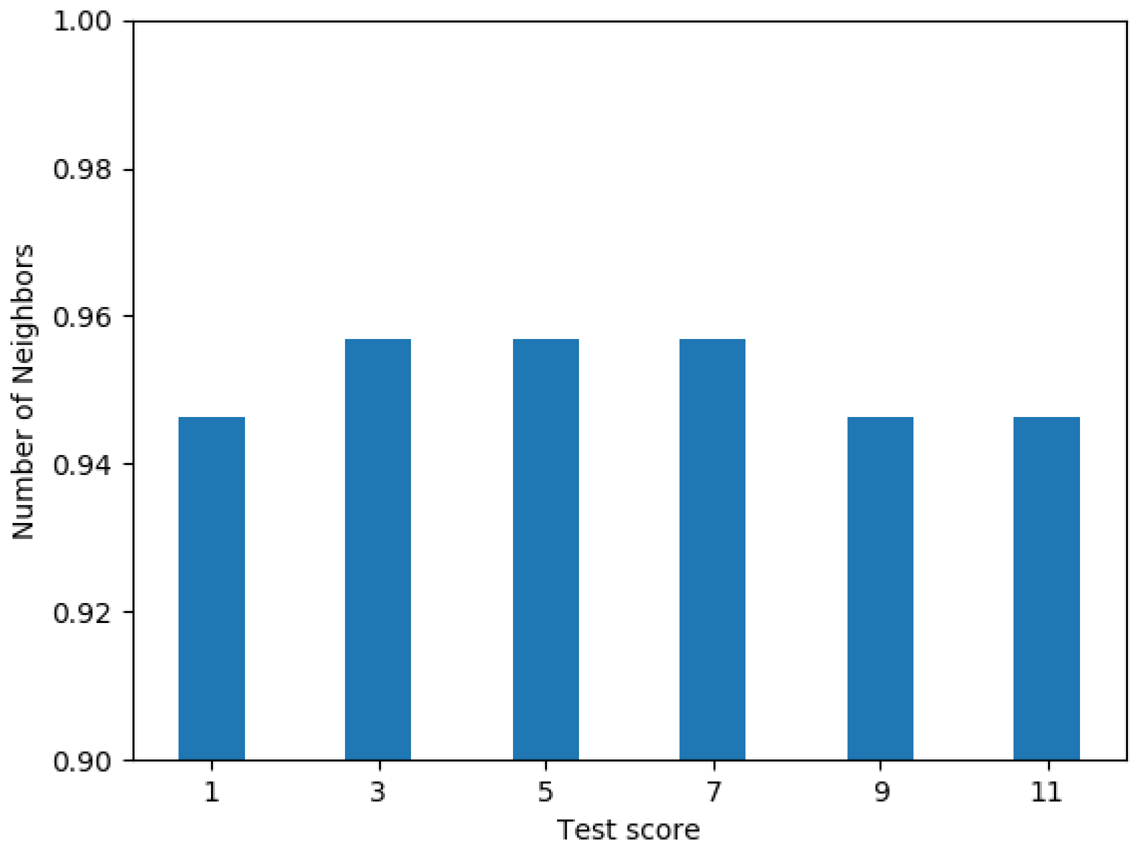
Out:

```
Best params: {'n_neighbors': 3}
Best score: 0.956989247311828
```

We have obtained the greatest mean accuracy using 3 neighbors. The following figure shows the score depending on the number of neighbors.

```
plt.figure()
plt.bar(param_grid['n_neighbors'], gscv.cv_results_['mean_test_score'])

plt.xticks(param_grid['n_neighbors'])
plt.ylabel("Number of Neighbors")
plt.xlabel("Test score")
plt.ylim((0.9, 1))
```



In this dataset, the functional observations have been sampled equiespaciated. If we approximate the integral of the $\mathbb{L}^2$ distance as a Riemann sum (actually the Simpson's rule it is used), we obtain that it is approximately equivalent to the euclidean distance between vectors.

$$\|f - g\|_{\mathbb{L}^2} = \left(\int_a^b |f(x) - g(x)|^2 dx \right)^{\frac{1}{2}} \approx \left(\sum_{n=0}^{N} \triangle\, h\, |f(x_n) - g(x_n)|^2 \right)^{\frac{1}{2}}$$
$$= \sqrt{\triangle\, h}\, d_{euclidean}(\vec{f}, \vec{g})$$

So, in this case, it is roughly equivalent to use this metric instead of the functional one, due to the constant multiplication do no affect the order of the neighbors.

Setting the parameter `sklearn_metric` of the classifier to True, a vectorial metric of sklearn can be passed. In `sklearn.neighbors.DistanceMetric` there are listed all the metrics supported.

We will fit the model with the sklearn distance and search for the best parameter. The results can vary sightly, due to the approximation during the integration, but the result should be similar.

```
knn = KNeighborsClassifier(metric='euclidean', sklearn_metric=True)
gscv2 = GridSearchCV(knn, param_grid, cv=KFold(shuffle=True, random_state=0))
gscv2.fit(X, y)

print("Best params:", gscv2.best_params_)
print("Best score:", gscv2.best_score_)
```

Out:

```
Best params: {'n_neighbors': 7}
Best score: 0.967741935483871
```

The advantage of use the sklearn metrics is the computational speed, three orders of magnitude faster. But it is not always possible to resample samples equiespaced nor do all functional metrics have a vector equivalent in this way.

The mean score time depending on the metric is shown below.

```
print("Mean score time (seconds)")
print("L2 distance:", np.mean(gscv.cv_results_['mean_score_time']), "(s)")
print("Sklearn distance:", np.mean(gscv2.cv_results_['mean_score_time']), "(s)")
```

Out:

```
Mean score time (seconds)
L2 distance: 0.609623114267985 (s)
Sklearn distance: 0.0009377532535129123 (s)
```

This classifier can be used with multivariate funcional data, as surfaces or curves in $\mathbb{R}^N$, if the metric support it too.

```
plt.show()
```

Total running time of the script: (0 minutes 34.861 seconds)

Download Python source code: plot_k_neighbors_classification.py

Download Jupyter notebook: plot_k_neighbors_classification.ipynb