Note

Click here to download the full example code

Radius neighbors classification

Shows the usage of the radius nearest neighbors classifier.

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

# sphinx_gallery_thumbnail_number = 2


import skfda
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from skfda.ml.classification import RadiusNeighborsClassifier
from skfda.misc.metrics import pairwise_distance, lp_distance
```

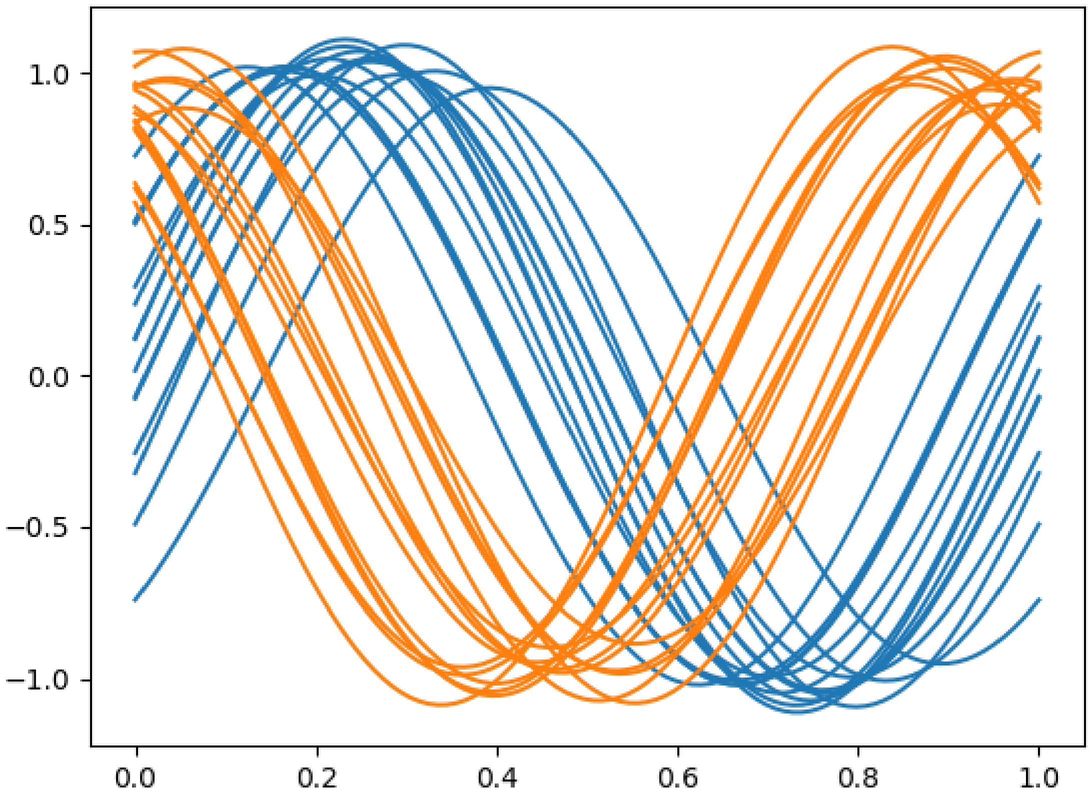
In this example, we are going to show the usage of the radius nearest neighbors classifier in their functional version, a variation of the K-nearest neighbors classifier, where it is used a vote among neighbors within a given radius, instead of use the k nearest neighbors.

Firstly, we will construct a toy dataset to show the basic usage of the API.

We will create two classes of sinusoidal samples, with different locations of their phase.

```
fd1 = skfda.datasets.make_sinusoidal_process(error_std=.0, phase_std=.35,
                                             random_state=0)
fd2 = skfda.datasets.make_sinusoidal_process(phase_mean=1.9, error_std=.0,
                                             random_state=1)

fd1.plot(color='C0')
fd2.plot(color='C1')
```



As in the K-nearest neighbor example, we will split the dataset in two partitions, for training and test, using the sklearn function `sklearn.model_selection.train_test_split()`.

```
# Concatenate the two classes in the same FDataGrid
X = fd1.concatenate(fd2)
y = np.array(15*[0] + 15*[1])

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

As in the multivariate data, the label assigned to a test sample will be the majority class of its neighbors, in this case all the samples in the ball center in the sample.

If we use the $\mathbb{L}^\infty$ metric, we can visualize a ball as a bandwidth with a fixed radius around a function.

The following figure shows the ball centered in the first sample of the test partition.

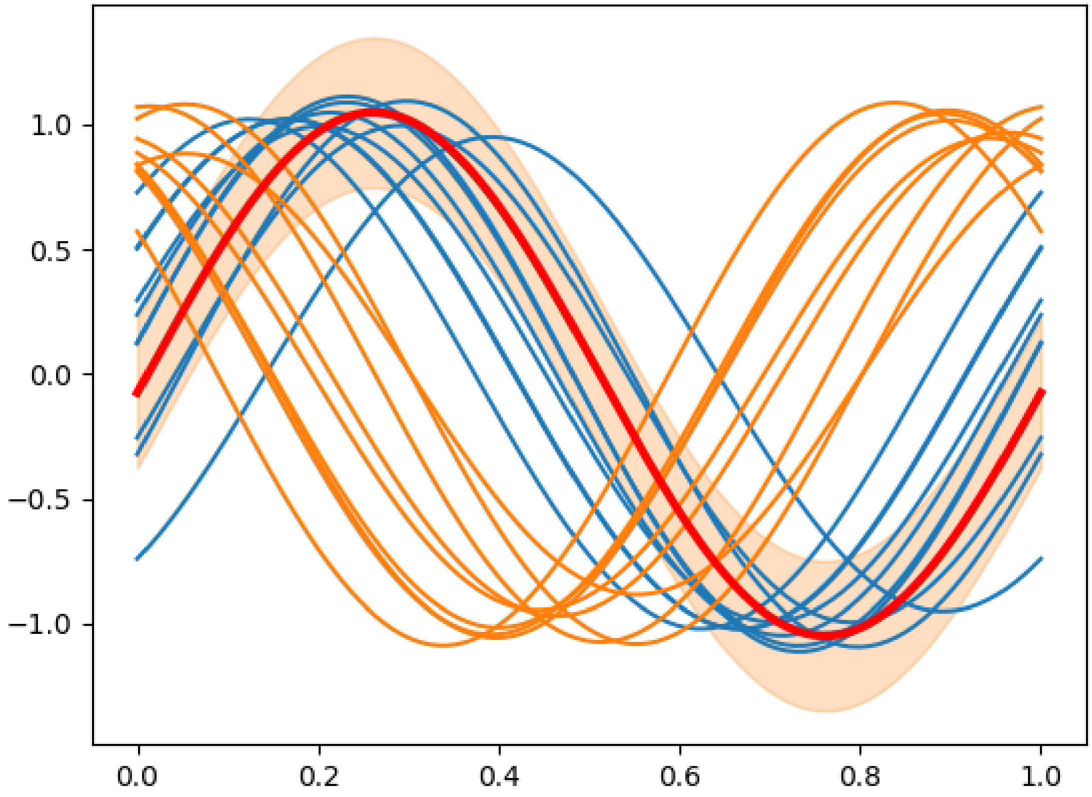
```
plt.figure()

sample = X_test[0]

X_train[y_train == 0].plot(color='C0')
X_train[y_train == 1].plot(color='C1')
sample.plot(color='red', linewidth=3)

lower = sample - 0.3
upper = sample + 0.3

plt.fill_between(sample.sample_points[0], lower.data_matrix.flatten(),
                 upper.data_matrix[0].flatten(),  alpha=.25, color='C1')
```



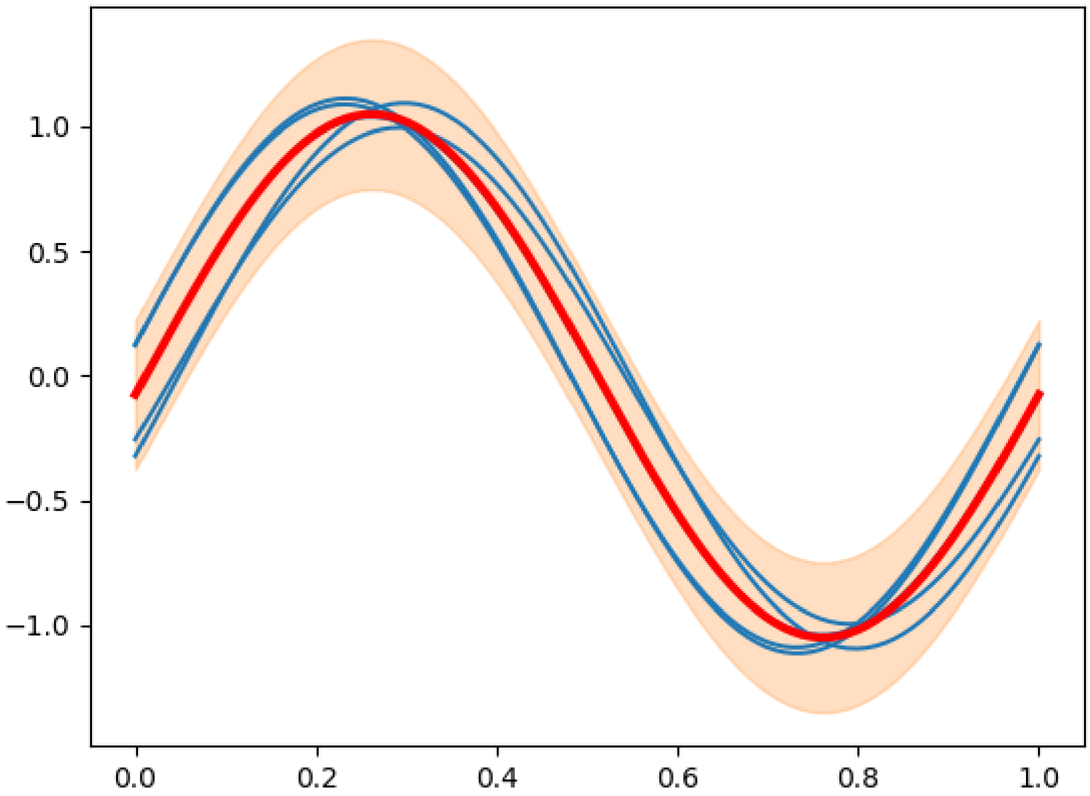
In this case, all the neighbors in the ball belong to the first class, so this will be the class predicted.

```
# Creation of pairwise distance
l_inf = pairwise_distance(lp_distance, p=np.inf)
distances = l_inf(sample, X_train)[0] # L_inf distances to 'sample'

plt.figure()

X_train[distances <= .3].plot(color='C0')
sample.plot(color='red', linewidth=3)

plt.fill_between(sample.sample_points[0], lower.data_matrix.flatten(),
                 upper.data_matrix[0].flatten(),  alpha=.25, color='C1')
```



We will fit the classifier `RadiusNeighborsClassifier`, which has a similar API than the sklearn estimator `sklearn.neighbors.RadiusNeighborsClassifier` but accepting `FDataGrid` instead of arrays with multivariate data.

The vote of the neighbors can be weighted using the paramenter `weights`. In this case we will weight the vote inversely proportional to the distance.

```
radius_nn = RadiusNeighborsClassifier(radius=.3,  weights='distance')
radius_nn.fit(X_train, y_train)
```

We can predict labels for the test partition with `predict()`.

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

Out:

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

In this case, we get 100% accuracy, althouth, it is a toy dataset and it does not have much merit.

```
test_score = radius_nn.score(X_test, y_test)
print(test_score)
```

Out:

```
1.0
```

As in the K-nearest neighbor example, we can use a sklearn metric approximately equivalent to the functional $\mathbb{L}^2$ one, but computationally faster.

We saw that $||f - g||_{\mathbb{L}^2} \approx \sqrt{\triangle\ h}\ d_{euclidean}(\vec{f}, \vec{g})$ if the samples are equiespaced (or almost).

In the KNN case, the constant $\sqrt{\triangle\ h}$ does not matter, but in this case will affect the value of the radius, dividing by $\sqrt{\triangle\ h}$.

In this dataset $\triangle\ h = 0.001$, so, we have to multiply the radius by 10 to achieve the same result.

The computation using this metric it is 1000 times faster. See the K-neighbors classifier example and the API documentation to get detailled information.

We obtain 100% accuracy with this metric too.

```
radius_nn = RadiusNeighborsClassifier(radius=3, metric='euclidean',
                                      weights='distance', sklearn_metric=True)


radius_nn.fit(X_train, y_train)

test_score = radius_nn.score(X_test, y_test)
print(test_score)
```

Out:

```
1.0
```

If the radius is too small, it is possible to get samples with no neighbors. The classifier will raise and exception in this case.

```
radius_nn.set_params(radius=.5) # Radius 0.05 in the L2 distance
radius_nn.fit(X_train, y_train)

try:
    radius_nn.predict(X_test)
except ValueError as e:
    print(e)
```

Out:

```
No neighbors found for test samples [3, 4, 5, 6, 7, 8, 9], you can try using larger
radius, give a label for outliers, or consider removing them from your dataset.
```

A label to these oulier samples can be provided to avoid this problem.

```
radius_nn.set_params(outlier_label=2)
radius_nn.fit(X_train, y_train)
pred = radius_nn.predict(X_test)

print(pred)
```

Out:

```
[0 1 0 2 2 2 2 2 2 2]
```

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 0.617 seconds)

Download Python source code: plot_radius_neighbors_classification.py

Download Jupyter notebook: plot_radius_neighbors_classification.ipynb

Gallery generated by Sphinx-Gallery