Note

Click here to download the full example code

Neighbors Scalar Regression

Shows the usage of the nearest neighbors regressor with scalar response.

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

# sphinx_gallery_thumbnail_number = 3

import skfda
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from skfda.ml.regression import KNeighborsScalarRegressor
from skfda.misc.metrics import norm_lp
```

In this example, we are going to show the usage of the nearest neighbors regressors with scalar response. There is available a K-nn version, `KNeighborsScalarRegressor`, and other one based in the radius, `RadiusNeighborsScalarRegressor`.

Firstly we will fetch a dataset to show the basic usage.

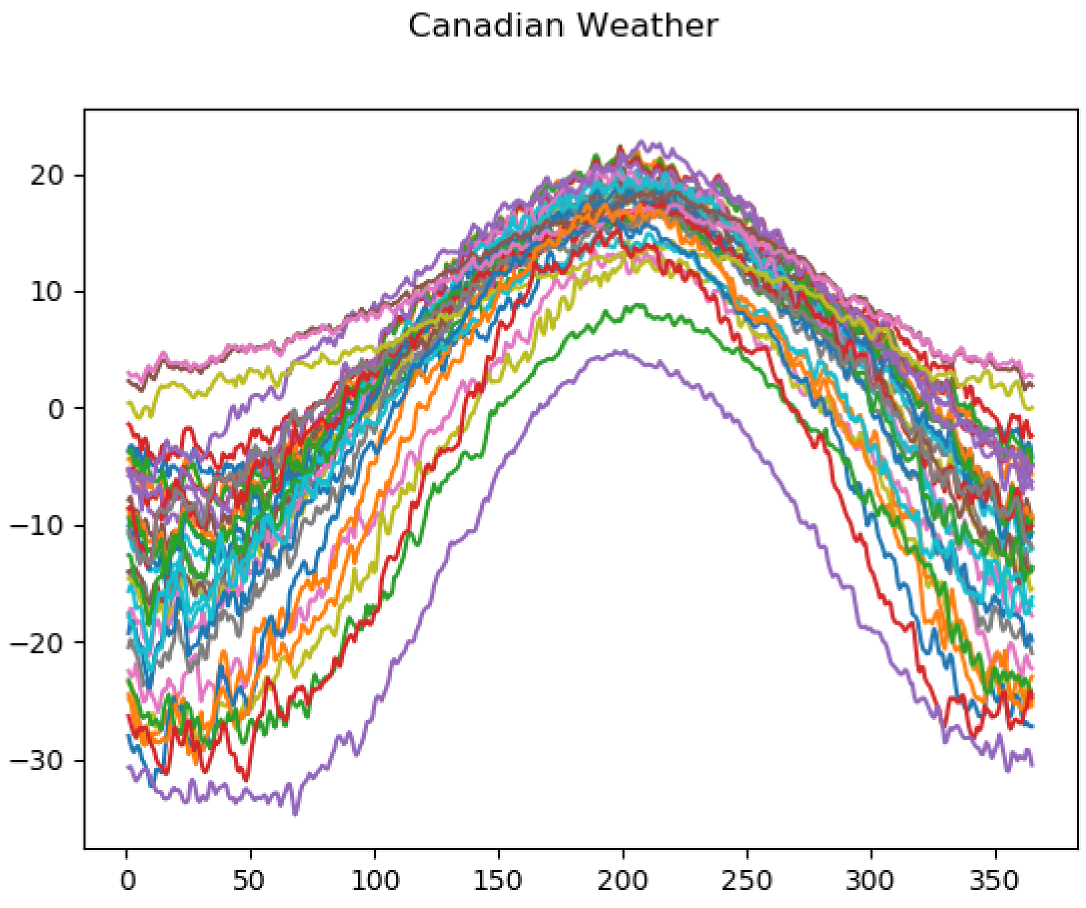
The caniadian weather dataset contains the daily temperature and precipitation at 35 different locations in Canada averaged over 1960 to 1994.

The following figure shows the different temperature curves.

```
data = skfda.datasets.fetch_weather()
fd = data['data']

# TODO: Change this after merge operations-with-images
fd.axes_labels = None
X = fd.copy(data_matrix=fd.data_matrix[..., 0])


X.plot()
```

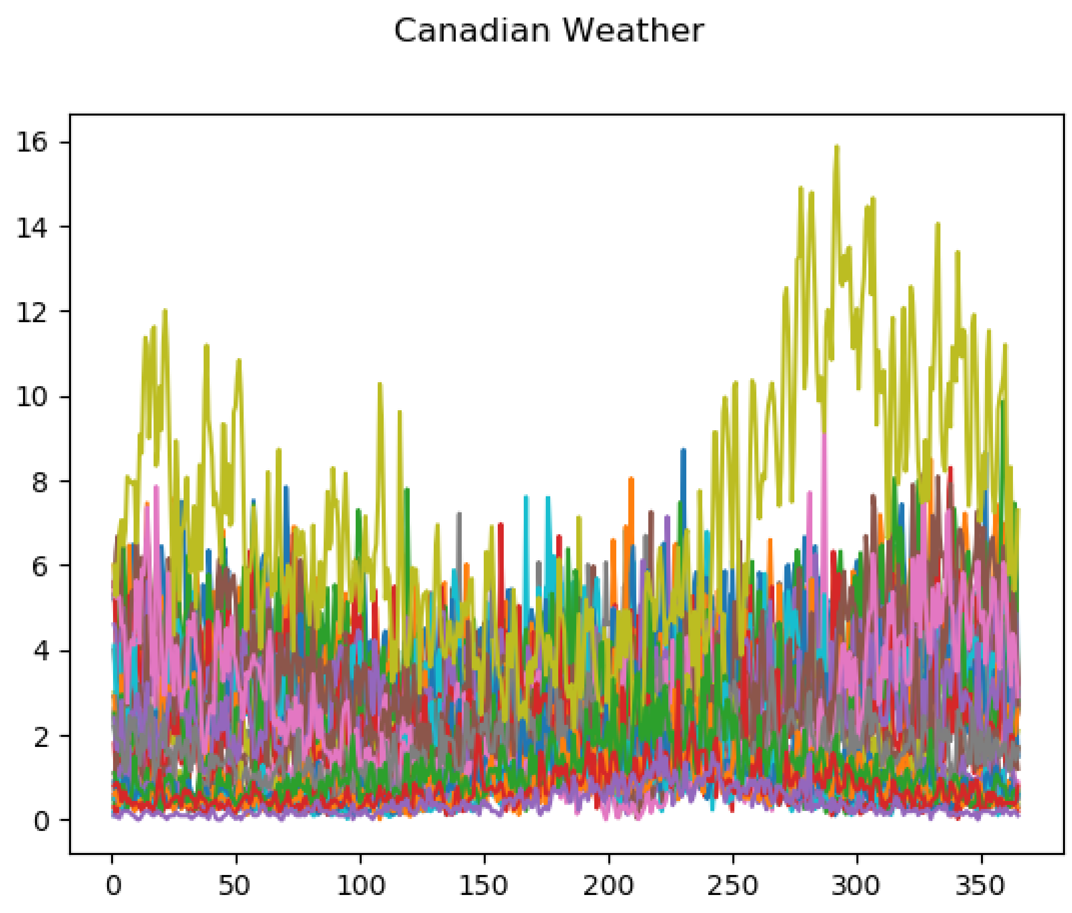


In this example we are not interested in the precipitation curves directly, as in the case with regression response, we will train a nearest neighbor regressor to predict a scalar magnitude.

In the next figure the precipitation curves are shown.

```
y_func = fd.copy(data_matrix=fd.data_matrix[..., 1])

plt.figure()
y_func.plot()
```



We will try to predict the total log precipitation, i.e, $logPrecTot_i = \log \int_0^{365} prec_i(t)dt$ using the temperature curves.

To obtain the precTot we will calculate the $\mathbb{L}^1$ norm of the precipitation curves.

```
prec = norm_lp(y_func, 1)
log_prec = np.log(prec)

print(log_prec)
```

Out:

```
[7.29351642 7.276003   7.28963327 7.13669533 7.09071552 7.02467869
 6.6863602  6.8024318  6.83507705 7.0947603  7.00802285 6.83904808
 6.82668983 6.67653755 6.86835588 6.55881227 6.22620723 6.10709712
 6.01737593 5.90518018 6.00355779 5.89770317 6.14175063 6.00938632
 5.60138054 7.0457185  6.73308438 6.40539373 7.85460108 5.58537424
 5.78751069 5.58086126 6.01713229 5.56119432 4.96097809]
```

As in the nearest neighbors classifier examples, we will split the dataset in two partitions, for training and test, using the sklearn function `sklearn.model_selection.train_test_split()`.

```
X_train, X_test, y_train, y_test = train_test_split(X, log_prec, random_state=7)
```

Firstly we will try make a prediction with the default values of the estimator, using 5 neighbors and the $\mathbb{L}^2$.

We can fit the `KNeighborsScalarRegressor` in the same way than the sklearn estimators. This estimator is an extension of the sklearn `sklearn.neighbors.KNeighborsRegressor`, but accepting a `FDataGrid` as input instead of an array with multivariate data.

```
knn = KNeighborsScalarRegressor(weights='distance')
knn.fit(X_train, y_train)
```

We can predict values for the test partition using `predict()`.

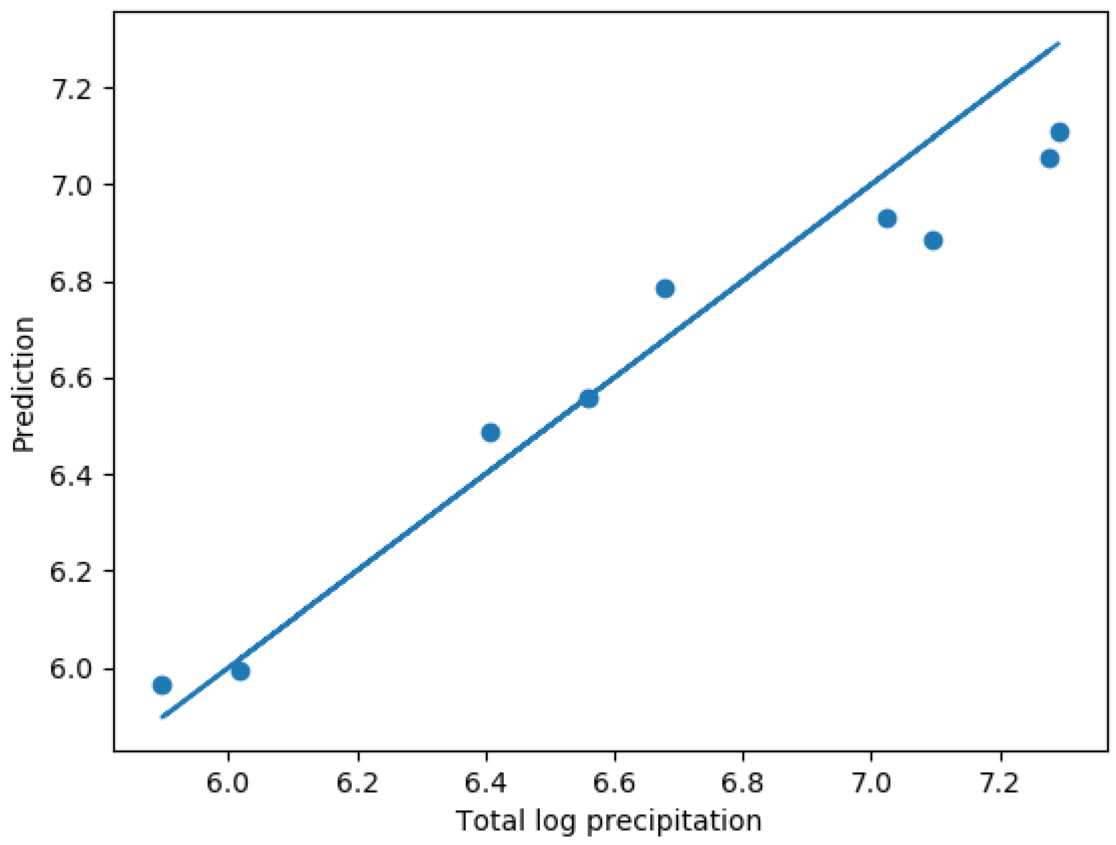
```
pred = knn.predict(X_test)
print(pred)
```

Out:

```
[7.10961556 5.99275367 7.05309816 6.88715827 6.78737106 5.96663073
 6.55900402 6.4855707  6.92975612]
```

The following figure compares the real precipitations with the predicted values.

```
plt.figure()
plt.scatter(y_test, pred)
plt.plot(y_test, y_test)
plt.xlabel("Total log precipitation")
plt.ylabel("Prediction")
```



We can quantify how much variability it is explained by the model with the coefficient of determination R^2 of the prediction, using `score()` for that.

The coefficient R^2 is defined as $(1 - u/v)$, where u is the residual sum of squares $\sum_i (y_i - y_{pred_i})^2$ and v is the total sum of squares $\sum_i (y_i - \bar{y})^2$.

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

Out:

```
0.9266325148983648
```

In this case, we obtain a really good aproximation with this naive approach, although, due to the small number of samples, the results will depend on how the partition was done. In the above case, the explained variation is inflated for this reason.

We will perform cross-validation to test more robustly our model.

As in the neighbors classifiers examples, we can use a sklearn metric to approximate the $\mathbb{L}^2$ metric between function, but with a much lower computational cost.

Also, we can make a grid search, using `sklearn.model_selection.GridSearchCV`, to determine the optimal number of neighbors and the best way to weight their votes.

```
param_grid = {'n_neighbors': np.arange(1, 12, 2),
              'weights': ['uniform', 'distance']}


knn = KNeighborsScalarRegressor(metric='euclidean', sklearn_metric=True)
gscv = GridSearchCV(knn, param_grid, cv=KFold(shuffle=True, random_state=0))
gscv.fit(X, log_prec)
```

We obtain that 7 is the optimal number of neighbors, and a lower value of the R^2 coefficient, but much closer to the real one.

```
print(gscv.best_params_)
print(gscv.best_score_)
```

Out:

```
{'n_neighbors': 7, 'weights': 'distance'}
0.5307446923249634
```

More detailed information about the canadian weather dataset can be obtained in the following references.

- Ramsay, James O., and Silverman, Bernard W. (2006). Functional Data Analysis, 2nd ed. , Springer, New York.
- Ramsay, James O., and Silverman, Bernard W. (2002). Applied Functional Data Analysis, Springer, New Yorkn’

```
plt.show()
```

Total running time of the script: (0 minutes 0.923 seconds)

Download Python source code: plot_neighbors_scalar_regression.py

Download Jupyter notebook: plot_neighbors_scalar_regression.ipynb