# Классификация в SKTime

## Импорт библиотек

Начнем с импорта нужных библиотек

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt

try:  
 import sktime  
except:  
 !pip install sktime --user  
 !pip install tsfresh   
 !pip install numba  
 !pip install llvmlite  
import sktime

from sktime.transformations.panel.rocket import MiniRocket

from sktime.datasets import load\_from\_tsfile  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.model\_selection import train\_test\_split

## Импорт данных

Теперь выберем набор данных. Для нашего урока мы воспользуемся уже имеющимся набором данных [открытые данные энергетических систем](https://open-power-system-data.org/).

url = 'https://data.open-power-system-data.org/time\_series/2020-10-06/'  
datafile = url + 'time\_series\_60min\_singleindex.csv'  
df\_all = pd.read\_csv(datafile, index\_col='utc\_timestamp', parse\_dates=True, low\_memory=False)

Также как и ранее выберем нужную страну, для примеров это была Германия.

def extract\_country(df\_all, country\_code, year\_min=None, year\_max=None):  
 """Extract data for a single country"""  
   
 # List of columns to extract  
 columns = [col for col in df\_all.columns if col.startswith(country\_code)]  
   
 # Extract columns and remove country codes from column labels  
 columns\_map = {col : col[3:] for col in columns}  
 df\_out = df\_all[columns].rename(columns=columns\_map)  
   
 # Exclude years outside of specified range, if any  
 if year\_min is not None:  
 df\_out = df\_out[df\_out.index.year >= year\_min]  
 if year\_max is not None:  
 df\_out = df\_out[df\_out.index.year <= year\_max]  
   
 return df\_out  
  
def transform\_dataframe(df, cols\_map):  
 # Rename columns for convenience  
 df = df[list(cols\_map.keys())].rename(columns=cols\_map)  
 # Convert from MW to GW  
 df = df / 1000  
# df = df.groupby(df.index.hour).mean()  
 df = df.rename\_axis('Date')  
# df.index = df.index.strftime('%Y-%m-%d-%h')  
 return df

df\_hrly = extract\_country(df\_all, country\_code='DE', year\_min=2015, year\_max=2019)  
df\_hrly.sample(3)  
  
cols\_map = {'load\_actual\_entsoe\_transparency' : 'Consumption',  
 'wind\_generation\_actual' : 'Wind',  
 'solar\_generation\_actual' : 'Solar'}  
df = transform\_dataframe(df\_hrly, cols\_map).dropna()  
  
# df wind + solar generation  
df['Wind+Solar'] = df[['Wind', 'Solar']].sum(axis=1, skipna=False)  
df.to\_csv('de\_clf\_data.csv')  
df.head()

Consumption Wind Solar Wind+Solar  
Date   
2015-01-01 07:00:00+00:00 41.133 10.208 0.071 10.279  
2015-01-01 08:00:00+00:00 42.963 10.029 0.773 10.802  
2015-01-01 09:00:00+00:00 45.088 10.550 2.117 12.667  
2015-01-01 10:00:00+00:00 47.013 11.390 3.364 14.754  
2015-01-01 11:00:00+00:00 48.159 12.103 4.198 16.301

Однако для текущей цели - изучить классификацию временных рядов давайте сделаем набор данных с классами. Для этого предлагается поставить задачу определение по дневному потреблению сезона (зима, весна, осень, лето):

* Разделить все данные на 4 сезона (зима, весна, осень, лето)
* взять данные с шагом 1 час.
* создать сегменты размером 24 часа (1 день).

Данная задача полностью искусственная, однако в целом соответствует некоторым из задачи стандартных тестов классификации временных рядов, например смотрите [тут](http://www.timeseriesclassification.com/description.php?Dataset=PowerCons).

Напомним, что для проведения классификации, как правило, данные следует разделять на сегменты. Как правило, эти сегменты не должны пересекаться. В нашем случае длина сегмента будет 24 точки.

Для цели создания набора данных давайте создадим 2 функции, первая split\_by\_month будет разделять наш набор данных на месяца. Вторая функция to\_segments будет создавать сегменты по 24 часа в сутках.

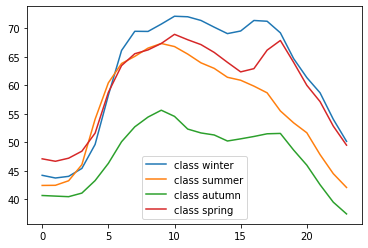
def split\_by\_month(df, months):  
 df\_ = pd.DataFrame()  
 for month in np.atleast\_1d(months):  
 df\_ = df\_.append(df.loc[df.index.month == month])  
 return df\_  
  
def to\_segments(df, column, size = 24):   
 df.index.hour[0]  
 start\_idx = 24-df.index.hour[0]  
 df = df.iloc[start\_idx:]  
 val = df[[column]].values  
 return val[:size\*(val.size//size)].reshape(-1,size)

winter = to\_segments(split\_by\_month(df, [1,2, 12]), 'Consumption', size = 24)  
summer = to\_segments(split\_by\_month(df, [6, 7, 8]), 'Consumption', size = 24)  
autumn = to\_segments(split\_by\_month(df, [9,10,11]), 'Consumption', size = 24)  
spring = to\_segments(split\_by\_month(df, [3, 4, 5]), 'Consumption', size = 24)  
print(winter.shape, summer.shape, autumn.shape, spring.shape)

(448, 24) (458, 24) (452, 24) (458, 24)

Теперь мы создали 4 класса. Давайте посмотрим как выглядят их примеры.

plt.figure()  
day = 10  
for i,(c,d) in enumerate(zip([winter,summer,autumn,spring],['winter','summer','autumn','spring'])):  
 plt.plot(c[day], label="class " + str(d))  
plt.legend(loc="best")  
plt.show()  
plt.close()



Хотя примеры показывают высокую степень сходства мы все же попробуем их классифицировать. Для этого создадим массив тренировочных и тестовых данных.

X = np.concatenate((  
 winter,   
 summer,   
 autumn,   
 spring))  
y = np.concatenate((  
 0\*np.ones(winter.shape[0]),  
 1\*np.ones(summer.shape[0]),  
 2\*np.ones(autumn.shape[0]),  
 3\*np.ones(spring.shape[0])  
 ))  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
print(X\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape)

(1271, 24) (1271,) (545, 24) (545,)

Современная версия SKTime допускает формат массивов в виде 2d ndarray (numpy) для однопеременных временных рядов. Для многопеременных временных рядов массив может быть задан в виде 3d ndarray. В общем случае однопеменный ряд также может быть задан как многопеременный с размерностью массива Также в пакете SKTime предусмотрены и другие форматы. В том числе специализированные. Для преобразования между форматами используются convert и convert\_to из модуля sktime.datatypes. Например, для преобразования двухмерного массива во встроенный однопеременных формат можно воспользоваться следующим скриптом

df = convert\_to(np.atleast\_3d(X).transpose(0,2,1), to\_type="nested\_univ")

Однако не все функции SKTime поддерживают 2d ndarray поэтому давайте создадим nested\_univ копии данных.

from sktime.datatypes import convert\_to  
df\_train = convert\_to(np.atleast\_3d(X\_train).transpose(0,2,1), to\_type="nested\_univ")   
df\_test = convert\_to(np.atleast\_3d(X\_test ).transpose(0,2,1), to\_type="nested\_univ")

## Классификация sklearn

Перед тестированием специализированных методов из пакета SKTime давайте посмотрим на результаты стандартных подходов из sklearn. В анализе временных рядов среди таких стандартных подходов наиболее распространен случайный лес. Также давайте попробуем алгоритм k-соседей с евклидовым расстоянием как некоторую предварительную базовую оценку.

from sklearn.ensemble import RandomForestClassifier  
from sklearn.neighbors import KNeighborsClassifier  
  
clf = RandomForestClassifier(n\_estimators=100)  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.881

clf = KNeighborsClassifier(n\_neighbors=1)  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.879

## Классификаторы на основе данных

В теоретической части мы уже обсуждали типы алгоритмов классификации временных рядов. Теперь давайте попробуем их на практике. Ранее мы обсуждали что в качестве некоторой базовой оценки можно использовать 1-NN DTW классификатор. Напомним, что этот подход сравнивает временные ряды "в целом". Расстояние DTW относится к эластичным мерам, стабильным к искажениям различных сегментов ряда.

В пакете SKTime данный тип алгоритмов находится в модуле distance\_based.

from sktime.classification.distance\_based import KNeighborsTimeSeriesClassifier  
clf = KNeighborsTimeSeriesClassifier(n\_neighbors=1, distance="dtw")  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.839

Полученный результат не превзошёл результат с евклидовым расстоянием. Вероятно, это связано с близостью форм сегментов. Однако 1-NN DTW может быть реализован и в более продвинутых формах. Одной из таких форм является расстояние разностей ddtw.

from sktime.classification.distance\_based import KNeighborsTimeSeriesClassifier  
  
clf = KNeighborsTimeSeriesClassifier(n\_neighbors=1, distance="ddtw")  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.917

Другим вариантом классификатора на основе данных являются шейплет классификатора. Напомним, что шейплет представляет собой участок временного ряда, обладающий наибольшей дискриминационной способностью для задачи. Другими словами, для участков шейплетов различия между классами наиболее существенны.

В ракете SKTime шейплеты могут быть найдены в модуле sktime.classification.shapelet\_based.

from sktime.classification.shapelet\_based import ShapeletTransformClassifier  
  
clf = ShapeletTransformClassifier(estimator=RandomForestClassifier(n\_estimators=100),  
 n\_shapelet\_samples=100,  
 max\_shapelets=100,  
 batch\_size=20)  
  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.916

Классификаторы на основе шейплетов позволяют найти наиболее различающиеся паттерны в данных. Как обсуждалось в теоретической части в ряде случаев более полезным оказываться повторяемость такие паттернов. В таких и подобных задачах для описания временных рядов (для их представления) могут быть использованы подходы на основе т.н. словаря (Dictionary-based).

В пакете SKTime для работы со словарями используется модуль sktime.classification.dictionary\_based.

from sktime.classification.dictionary\_based import (ContractableBOSS,   
 IndividualBOSS,   
 IndividualTDE,   
 MUSE, WEASEL)

clf = IndividualBOSS(random\_state=47)  
clf.fit(X\_train, y\_train)  
clf.score(X\_test, y\_test)

0.8110091743119267

clf = IndividualTDE(random\_state=47)  
clf.fit(X\_train, y\_train)  
clf.score(X\_test, y\_test)

0.8403669724770643

clf = MUSE(random\_state=47)  
clf.fit(X\_train, y\_train)  
clf.score(X\_test, y\_test)

0.9504587155963303

## Классификаторы на основе признаков

Классификаторы, рассмотрены выше можно было отнести к т.н. классификаторам на основе данных. Однако как уже обсуждалось в теоретической части, часто следует отдавать предпочтения классификаторам на основе признаков. Одними из наиболее популярных подходов среди таких классификаторов являются интервальные подходы. Суть таких подходов сводится к

* поиску или случайному выбору интервалов внутри временных рядов;
* извлечению некоторого набора признаков для такого интервала;
* построению некоторого классификатора, например, дерева для каждого интервала;
* объединение результатов для всех интервалов, например, при помощи леса деревьев.

В пакете SKTime такие методы могут быть найдены в модуле sktime.classification.interval\_based.

from sktime.classification.interval\_based import (TimeSeriesForestClassifier,   
 RandomIntervalSpectralEnsemble,   
 SupervisedTimeSeriesForest)

clf = TimeSeriesForestClassifier(n\_estimators=100,random\_state=47)  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.947

clf = RandomIntervalSpectralEnsemble(n\_estimators=100, random\_state=47)  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.934

clf = SupervisedTimeSeriesForest(n\_estimators=100, random\_state=47)  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.947

Помимо алгоритмов на основе деревьев, популярным на сегодня является методы на основе ядерных преобразований. Одним из наиболее успешных алгоритмов в этом направлении является алгоритм Rocket. Суть данного алгоритма сводится к трансформации данных при помощи набора сверточных ядре со случайно выбранными параметрами. Затем авторы рекомендуются использовать RidgeClassifierCV, который предоставляет собой Логистическую регрессию с L2 регуляризацией, параметр регуляризации выбирается при помощи кросс-валидации.

Мы рассмотрим ускоренную версию преобразования MiniRocket.

from sklearn.linear\_model import RidgeClassifierCV  
  
transform = MiniRocket(num\_kernels=10000, n\_jobs = -1)  
Xt\_train = transform.fit\_transform(X\_train)  
Xt\_test = transform.transform(X\_test)  
  
clf = RidgeClassifierCV()  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.912

Другим важным подходом на основе признаков является подход выбора признакового пространства. Таким пространством может быть лишь использование небольшого числа суммарных статистик (среднее, медиана, дисперсия и т.д.), которые должны как-то описать временной ряд.

В некоторых случаях суммарных статистик может быть недостаточно. В общем случае помимо суммарных статистик могут быть и Временные или частотные (псевдочастотные) признаки. Давайте перечислим примеры таких признаков в виде списка:

* суммарные статистики: среднее, медиана, дисперсия, энтропия и т.д.;
* временные признаки: наклон, характер тренда, точки пересечения некоторых пороговых значений, значения пиков и их позиции, автокорреляционные коэффициенты и т.д.
* частотные признаки: значения пиков спектра, коэффициенты авторегрессии, коэффициенты, например, вейвлет разложений и т.д.
* другие признаки. В общем случае признаковые пространства могут быть достаточно объемными. Однако, как правило, не все признаки важны, поэтому для таких пространств следует проводить отбор признаков. Существуют и готовые решения, где из объемных пространств уже выделены некоторые наиболее значимые подпространства. Среди таких подходов популярным является Catch22. В данном подходе среди порядка 4500 признаков выделено 22, имеющих большую значимость для стандартных тестов.

В пакете SKTime описанные подходы можно найти в модуле sktime.classification.feature\_based. Давайте протестируем 2 классификатора из этого пакета SummaryClassifier, представляющий описание временных рядов суммарными статистиками и Catch22Classifier, представляющий описание временного ряда указанным выше признаковым пространством.

from sktime.classification.feature\_based import SummaryClassifier, Catch22Classifier  
from sklearn.ensemble import RandomForestClassifier

clf = SummaryClassifier(estimator=RandomForestClassifier(n\_estimators=100))  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(df\_test, y\_test):.3f}')

test score 0.631

clf = Catch22Classifier(estimator=RandomForestClassifier(n\_estimators=100))  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(df\_test, y\_test):.3f}')

test score 0.839

Дополнительно отметим, что пакет SKTime также позволяет создавать пайплайны выделения признаков и классификации в ручном режиме. Например, так, как это показано ниже. Показанный пример повторяет алгоритм TimeSeriesForestClassifier, однако, не полностью.

from sktime.transformations.panel.summarize import RandomIntervalFeatureExtractor  
from sktime.utils.slope\_and\_trend import \_slope  
from sklearn.pipeline import Pipeline  
  
steps = [(  
 "extract", RandomIntervalFeatureExtractor(  
 n\_intervals="sqrt", features=[np.mean, np.std, \_slope]  
 )),  
 ("clf", RandomForestClassifier()),]  
  
clf = Pipeline(steps)  
clf.fit(df\_train, y\_train)  
print(f' test score {clf.score(df\_test, y\_test):.3f}')

test score 0.910

Также важно отметить возможность работы с некоторыми другими библиотеками работы с признаками. Так, одной из наиболее популярных библиотек является [tsfresh](https://tsfresh.readthedocs.io/en/latest/text/introduction.html). Данная библиотека является достаточно популярной, хотя и не самой современной. На данный момент ведется интеграция функционала tsfresh в SKTime. По этому мы покажем возможность работы с tsfresh как с отдельным инструментом.

В самом простом случае tsfresh требует формат данных в виде DataFrame, в котором все сегменты данных были бы представлены в одной колонке (для однопеременных данных). Для преобразования форматов воспользуемся встроенной функцией sktime from\_nested\_to\_long.

Для выделения признаков tsfresh используется метод extract\_features.

from tsfresh import extract\_features  
from sklearn.preprocessing import MinMaxScaler  
from sktime.datatypes.\_panel.\_convert import from\_nested\_to\_long

Xt\_train = extract\_features(  
 from\_nested\_to\_long(df\_train),  
 column\_id="index",  
 column\_value="value",  
 column\_kind="column",  
 column\_sort="time\_index", )  
Xt\_train.head(2)

Feature Extraction: 100%|██████████████████████████████████████████████████████████████| 10/10 [00:42<00:00, 4.24s/it]

[2 rows x 789 columns]

Xt\_test = extract\_features(  
 from\_nested\_to\_long(df\_test),  
 column\_id="index",  
 column\_value="value",  
 column\_kind="column",  
 column\_sort="time\_index", )  
Xt\_test.head(2)

Feature Extraction: 100%|██████████████████████████████████████████████████████████████| 10/10 [00:23<00:00, 2.38s/it]

[2 rows x 789 columns]

И так мы выделили 789 признаков. Среди признаков есть значения достаточно разных порядков, а также отсутствующие значения. Проведем стандартизацию полученных признаков.

Xt\_train = (Xt\_train - Xt\_train.min())/(Xt\_train.max() - Xt\_train.min()).astype(np.float32)  
Xt\_test = (Xt\_test - Xt\_train.min())/(Xt\_train.max() - Xt\_train.min()).astype(np.float32)

Xt\_train = Xt\_train.dropna(axis=1)  
Xt\_test = Xt\_test[Xt\_train.columns]

clf = RandomForestClassifier(n\_estimators=100)  
clf.fit(Xt\_train, y\_train)  
clf.score(Xt\_test, y\_test)

0.5798165137614679

Отметим, что при необходимости в пакете SKTime можно найти и ряд не рассмотренных тут преобразований, техник выделения признаков и техник классификации временных рядов.

Вопросы:

1 Какой шаг временного ряда мы взяли для задачи классификации:

Ответ: 1 час или 1h

2 Какой формат данных ожидается в SKtime для данных в виде ndarray

* samples×variable×segment
* variable×samples×segment
* segment×samples×variable

ответ: samples×variable×segment

3 Какой из классификаторов основан на случайных интервалах:

* IndividualBOSS
* TimeSeriesForestClassifier
* KNeighborsTimeSeriesClassifier

Ответ: TimeSeriesForestClassifier

4 Какой из классификаторов основан на словарях:

* ShapeletTransformClassifier
* MUSE
* KNeighborsTimeSeriesClassifier

Ответ: MUSE

5 Какой подход в наших примерах выделял наибольшее пространство признаков:

* tsfresh
* SummaryClassifier,
* Catch22Classifier
* MiniRocket

Ответ: MiniRocket

6 Какой подход основан на выделении признаков для случайных интервалов временного ряда:

* TimeSeriesForestClassifier
* MiniRocket
* ShapeletTransformClassifier

Ответ: TimeSeriesForestClassifier

7 Вопрос на сопоставление. Выбирите методы для каждого класса

* Интервальные методы: RandomIntervalSpectralEnsemble
* Методы на основе словарей: IndividualBOSS
* Методы на основе DTW: KNeighborsTimeSeriesClassifier
* Метод на основе больших признаковых пространств: Catch22Classifier