Классификация временных рядов

Классификация одномерных временных рядов с использование методов машинного обучения библиотек sklearn и sktime. Представление временных рядов для задач классификации. Использование традиционных методов машинного обучения библиотеки sklearn для классификации временных рядов. Использование специальных методов sktime: временное дерево и временной лес, расстояние DTW и метод dtw-knn, классификаторы на основе словарей. Классификатор rocket.

Импорт библиотек и данных

```
!pip install -U pyts
!pip install -U tslearn
!pip install -U sktime

import matplotlib.pyplot as plt
import numpy as np

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline

import pandas as pd
```

Для тестирования классификаторов временных рядов мы будем использовать небольшой набор данных GunPoint из библиотеки pyts. Набор данных включает в себя одну женщину-актера и одного актера-мужчину, которые делают движение рукой как будто достают пистолет. В первом классе они вынимают копию пистолета из кобуры, закрепленной на бедре, наводят его на цель примерно на одну секунду, затем возвращают пистолет в кобуру и складывают руки по бокам. Во втором классе актеры держат пистолеты по бокам. Они указывают указательными пальцами на цель примерно на одну секунду, а затем разводят руками по бокам. Для обоих классов отслеживается центроиды правой руки актера по осям X и Y, которые, по-видимому, сильно коррелированы.

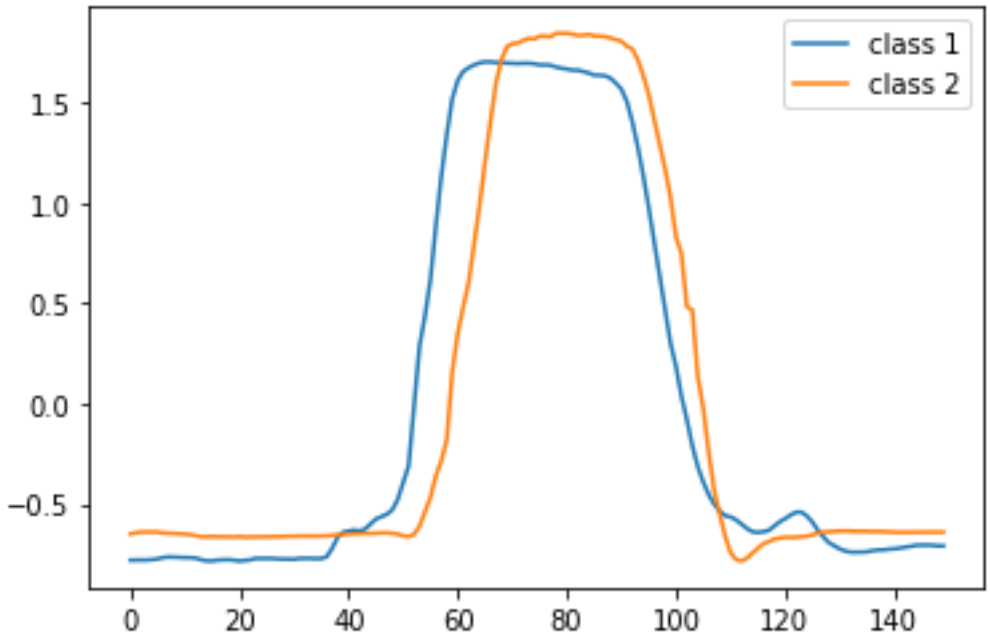
```
from pyts.datasets import load_gunpoint

x_train,x_test, y_train, y_test  = load_gunpoint(return_X_y=True)
```

Давайте посмотрим на классы.

```
classes = np.unique(np.concatenate((y_train, y_test), axis=0))
print('calss lables',classes)
print('train x',x_train.shape,'train lables', y_train.shape,'test
x',x_test.shape,'test lables', y_test.shape)

plt.figure()
for c in classes:
    c_x_train = x_train[y_train == c]
    plt.plot(c_x_train[0], label="class " + str(c))
```

```
plt.legend(loc="best")
plt.show()
plt.close()
```

```
calss lables [1 2]
train x (50, 150) train lables (50,) test x (150, 150) test lables (150,)
```



Библиотека SKTime использует различные форматы данных в задачах классификации. Вы можете прочитать об этом в документации. Для преобразования данных из обычных данных в набор для SKTime мы будем использовать функцию from_2d_array_to_nested из sktime.utils.data_processing.

Примечание. Многие хранилища временных рядов используют собственные форматы данных. Для преобразования формата данных вы можете использовать некоторые утилиты, из tslearn.utils.

```
from tslearn.utils import from_pyts_dataset, to_sktime_dataset
X_train = to_sktime_dataset(from_pyts_dataset(x_train))
X_test  = to_sktime_dataset(from_pyts_dataset(x_test))
X_train.head()
```

```
from sktime.utils.data_processing import from_2d_array_to_nested
X_train = from_2d_array_to_nested(x_train)
X_test  = from_2d_array_to_nested(x_test)
X_train.head()
```

```
                                                    0
0  0     -0.647885
   1     -0.641992
   2     -0.63818...
1  0     -0.644427
   1     -0.645401
   2     -0.64705...
```

```
2  0     -0.778353
1     -0.778279
2     -0.77715...
3  0     -0.750060
1     -0.748103
2     -0.74616...
4  0     -0.599539
1     -0.597422
2     -0.59926...
```

Исследование классификаторов

Методы пакета Sklearn

Для начала попробуем использовать классификатор SKlearn в качестве основы. Здесь мы не будем использовать измененный формат данных. Если вы собираетесь использовать вложенные данные SKTime в SKLearn, используйте from_nested_to_2d_arrayfrom sktime.utils.data_processing или Tabularizer в конвейерах sklearn..

```
# let's get a baseline for comparison
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=100)
classifier.fit(x_train, y_train)
classifier.score(x_test, y_test)
```

```
0.9266666666666666
```

Упражнение 1

1. Попробуйте еще несколько классификаторов из `sklearn`. Например, SVM, KNN или еще какие-то на ваш выбор.

Методы пакета Sktime

Дерево временных рядов.

Дерево временных рядов представляет собой модификацию алгоритма дерева классификации для временных рядов. В этом методе предлагается разбить выборку на случайные интервалы, извлечь три признака (среднее, стандартное отклонение и наклон) из каждого интервала, обучить дерево решений по извлеченным признакам.

```
from sktime.transformations.panel.summarize import
RandomIntervalFeatureExtractor
from sklearn.tree import DecisionTreeClassifier
from sktime.utils.slope_and_trend import _slope

steps = [
    ("extract",
        RandomIntervalFeatureExtractor(
            n_intervals="sqrt",
            features=[np.mean, np.std, _slope])),
```

```
    ("clf", DecisionTreeClassifier()),
]
time_series_tree = Pipeline(steps)
time_series_tree.fit(X_train, y_train)
time_series_tree.score(X_test, y_test)
```

```
0.8533333333333334
```

Лес временных рядов

Фактически, мы можем объединить несколько деревьев для повышения точности и надежности.

```
tsf = ComposableTimeSeriesForestClassifier(
    estimator=time_series_tree,
    n_estimators=100)
tsf.fit(X_train, y_train)
tsf.score(X_test, y_test)
```

```
0.9466666666666667
```

На самом деле мы можем использовать встроенный классификатор лес временных рядов (TSF). TSF здесь представляет собой ансамбль древовидных классификаторов, построенных на полученных статистиках по случайно выбранным интервалам. Для каждого дерева интервалы выбираются случайным образом. Полное количество интервалов $\sqrt{series\_length}$. Из каждого из этих интервалов извлекаются среднее значение, стандартное отклонение и наклон и объединяются в вектор признаков. Эти новые функции затем используются для построения дерева, которое добавляется к ансамблю.

```
from sktime.classification.interval_based import
TimeSeriesForestClassifier
tsf = TimeSeriesForestClassifier(n_estimators=100, random_state=47)
tsf.fit(X_train, y_train)
tsf.score(X_test, y_test)
```

```
0.9666666666666667
```

Спектральный ансамбль со случайными интервалами (RISE)

Еще одним популярным вариантом леса временных рядов является так называемый спектральный ансамбль со случайными интервалами (RISE), который использует несколько преобразователей выделения признаков от ряда к ряду, в том числе: коэффициенты автокорреляции и коэффициенты спектральной мощности. Подобно Лесу временных рядов, здесь извлечение всех признаков производится на случайных интервалах временных рядов для нескольких деревьев.

```
from sktime.classification.interval_based import
RandomIntervalSpectralForest

rise = RandomIntervalSpectralForest(n_estimators=10, random_state=47)
rise.fit(X_train, y_train)
rise.score(X_test, y_test)
```

```
0.94
```

Классификатор на основе расстояния с динамическим искажением времени (DTW)

Для временных рядов наиболее популярный алгоритм k-ближайших соседей основан на измерении расстояния с динамическим искажением времени (DTW). Алгоритм DTW состоит из следующих шагов:

- Вычислить расстояние между первой точкой в сегменте первом сегменте ряда и каждой точкой во втором сегменте.
- Выберите минимум вычисленных значений и сохраните его (это этап «деформации времени»).
- Перейдите ко второй точке и повторите этап 1.
- Двигайтесь шаг за шагом по точкам и повторяйте этап 1, пока не будут исчерпаны все точки.
- Вычислите и выберите минимальное расстояние между первой точкой во втором сегменте серии и каждой точкой в первой серии.
- Двигайтесь шаг за шагом по точкам во втором сегменте и повторяйте этап 3, пока не будут исчерпаны все точки.
- Просуммируйте все сохраненные минимальные расстояния.

Алгоритм DTW формально можно описать как

$$D(i,j) = 0$$

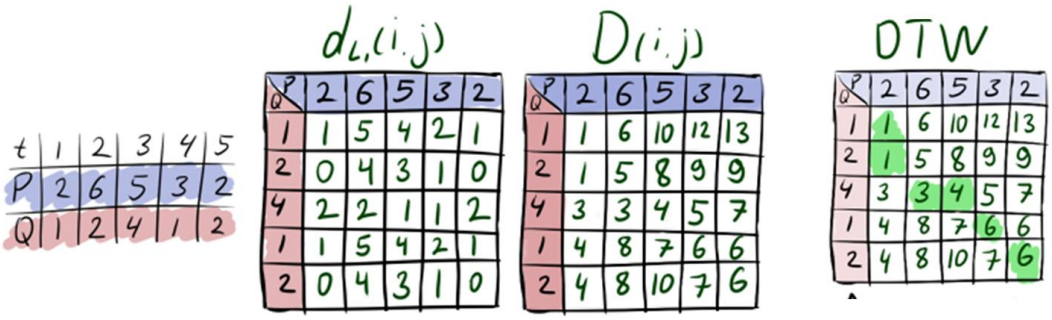
$$D(i,j) = dist(x_i, y_j) + min\{D(i-1,j), D(i,j-1), D(i-1,j-1)\}$$

$$d_{DWT} = min_i \frac{\sum_{j=0}^{K} D\,(i,j)}{K}$$

где

- D это элемент виртуальной матрицы $D(i,j)$ с размером $M_x \times M_y$;
- $dist$ функция расстояния, например эвклидово;
- x и y сегменты ряда, с размерами M_x и M_y соответственно; данные размеры могут быть не равными;
- K - расстояние по виртуальной траектории от нижнего правого угла D ($D(M_x, M_y)$) к верхнему левому ($D(0,0)$), по такой траектории, чтобы сумма расстояния была минимальной.

Пример расчета DTW



Dynamic Time Wrapping (DTW) нелинейный алгоритм, основанный на поиске максимального сходства между точками независимо от позиции индекса. Фактически этот алгоритм в сочетании с K ближайших соседей можно рассматривать как основу для классификации всех временных рядов.

Основным преимуществом расстояния DTW является независимость (инвариантность) от сдвигов или других незначительных изменений в сегментах (например, его сжатие и растяжение). Другими словами, метод DTW пытается построить матрицу отображения (или преобразования) одного сегмента в другой и найти лучшее (расстояние между ними с минимальной стоимостью). Главный недостаток DTW - высокая сложность и неявность поиска подобия. Таким образом, в некоторых случаях DTW может показывать сходство там, где его быть не должно.

Примечание. Здесь мы будем использовать так называемую разностную dtw distance - расстояние первых произовдных временных рядов.

```
from sktime.classification.distance_based import
KNeighborsTimeSeriesClassifier

knn = KNeighborsTimeSeriesClassifier(n_neighbors=1, distance="ddtw")
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

```
0.9866666666666667
```

Dictionary based Classifier

Подходы классификации временных рядов на основе словаря адаптируют модель набора слов, обычно используемую в обработке сигналов, компьютерном зрении и обработке звука, для классификации временных рядов.

- Скользящее окно длины w пробегается по серии.
- Для каждого окна действительно-значный ряд длины w преобразуется посредством процессов аппроксимации и дискретизации в символьную строку длины l, состоящую из α возможных букв.
- Подсчитывается появление в сегменте ряда каждого «слова» из словаря, определенного l и α,

- После завершения прохождения скользящего окна серия преобразуется в гистограмму.
- Классификация производится основана на гистограммах слов, извлеченных из ряда.

Среди всех классификаторов на основе словаря мы будем рассматривать Мешок символов SFA (BOSS). Мешок символов SFA (BOSS) BOSS - это совокупность отдельных классификаторов BOSS, использующих преобразование SFA (символьное приближение Фурье). Классификатор выполняет поиск по сетке однотипных классификаторов для разных значений параметров l, α, w и p (нормализует каждое окно). Из искомых классификаторов сохраняются только те, точность которых не менее 92% от лучшего классификатора. Отдельные классификаторы BOSS используют несимметричную функцию расстояния, расстояние BOSS, в сочетании с классификатором ближайшего соседа.

Contractable BOSS (cBOSS)

cBOSS значительно ускоряет BOSS без существенной разницы в точности за счет улучшения способа формирования ансамбля. cBOSS использует отфильтрованный выбор параметров для поиска членов своего ансамбля. Каждый член ансамбля построен на подвыборке размером 70% от выборки данных с использованием метода случайной выборки без замены. Также в данном методе введена экспоненциальная схема взвешивания прогнозов базовых классификаторов.

```
from sktime.classification.dictionary_based import ContractableBOSS,
TemporalDictionaryEnsemble
boss = ContractableBOSS(random_state=47,max_ensemble_size=100)
boss.fit(X_train, y_train)
boss.score(X_test, y_test)
```

```
0.9933333333333333
```

Классификация на основе Шейплетов (Shapelets)

Шейплеты определяются как подвыборки временного ряда (или сегмента временного ряда), которые в некотором смысле являются максимально репрезентативными для своего класса. Если предполагается задача бинарной классификации, то шейплет - это часть серии (интервал, диапазон), которая по некоторой мере представлена в большинстве сегментов одного класса и отсутствует в сериях другого класса.

ROCKET Classifer - это тип классификаторов на основе Shapelets, основанный на так называемых ROCKET преобразованиях. **Преобразования ROCKET** - это преобразования временных рядов с использованием случайных сверточных ядер (случайная длина, веса, смещение, расширение и заполнение). ROCKET Classifer вычисляет две характеристики из результирующих карт признаков (полученных после преобразований): максимальное значение и соотношение положительных значений ко всем (ppv). Преобразованные объекты используются для обучения линейного классификатора.

```
from sktime.classification.shapelet_based import ROCKETClassifier
shapelet = ROCKETClassifier(random_state=47)
```

```
shapelet.fit(X_train, y_train)
shapelet.score(X_test, y_test)

0.9933333333333333
```

Упражнение 2

1. Сравните результаты работы классификаторов из SKTime и sklearn для исследуемого выше набора данных.

Упражнение 3

1. Работа со встроенными данными.
    a. Загрузите набор данных *load_italy_power_demand* из *sktime.datasets*.
    b. Используйте split = "train", return_X_y = True для тренировочных данных и split = "test", return_X_y = True для тестовых данных.
    c. Выберите тестовую часть с размером до 50 экземпляров.
    d. Попробуйте визуализировать тестовые и тренировочные данные.
    e. Выберите 4 классификатора SKlearn и 4 классификатора SKTIme для этого набора данных.

Упражнение 4

1. Подберите произвольный набор данных для набора данных из интернета
2. Загрузите набор данных по следующей ссылке: https://raw.githubusercontent.com/jbrownlee/Datasets/master/IndoorMovement.zip
3. Набор данных включает сигналы (временные ряды) от людей, перемещающихся через 2 комнаты. Сигналы принимаются с 4 датчиков. Сигналы разделены на 3 пути (группы) того, как движутся люди, и два класса (цели): "+1" - когда человек проходит между двумя комнатами "-1" - когда человек не проходит между двумя комнатами.
4. Задача - по этим датчикам классифицировать переходы между комнатами.
5. Для начала загрузим данные.

The task is inspired by this tutorial: https://machinelearningmastery.com/indoor-movement-time-series-classification-with-machine-learning-algorithms/

```
import os

def load_dataset(prefix='Indoor/'):
    '''
    load_dataset of Indoor Movements.

    Parameters
    -----------
    prefix: str
        path to the directory with the dataset

    Returns
    -----------
    sequences: list,
        data of 4-th sensors for each instance (human).
    targets: 1d ndarray,
```

```
            targets for each instance +1 or -1.
        groups: 1d ndarray,
            group id for each instance 1,2 or 3.

        '''
        grps_dir, data_dir = prefix+'groups/', prefix+'dataset/'

        # load mapping files
        targets = pd.read_csv(data_dir + 'MovementAAL_target.csv', header=0)
        groups  = pd.read_csv(grps_dir + 'MovementAAL_DatasetGroup.csv',
header=0)
        paths   = pd.read_csv(grps_dir + 'MovementAAL_Paths.csv', header=0)

        # load traces
        sequences = list()

        for name in os.listdir(data_dir):

            filename = os.path.join(data_dir,name)

            if filename.endswith('_target.csv'): continue

            df = pd.read_csv(filename, header=0)

            values = df.values.reshape(-1,1)

            sequences.append(values.tolist())

        return sequences, targets.values[:,1], groups.values[:,1]


    sequences, targets, groups =load_dataset(prefix='Indoor/')

    labels, counts = np.unique(targets, return_counts=True)
    print('targets',labels, counts)
    labels, counts = np.unique(groups, return_counts=True)
    print('targets',labels, counts)

    targets [-1  1] [156 158]
    targets [1 2 3] [104 106 104]
```

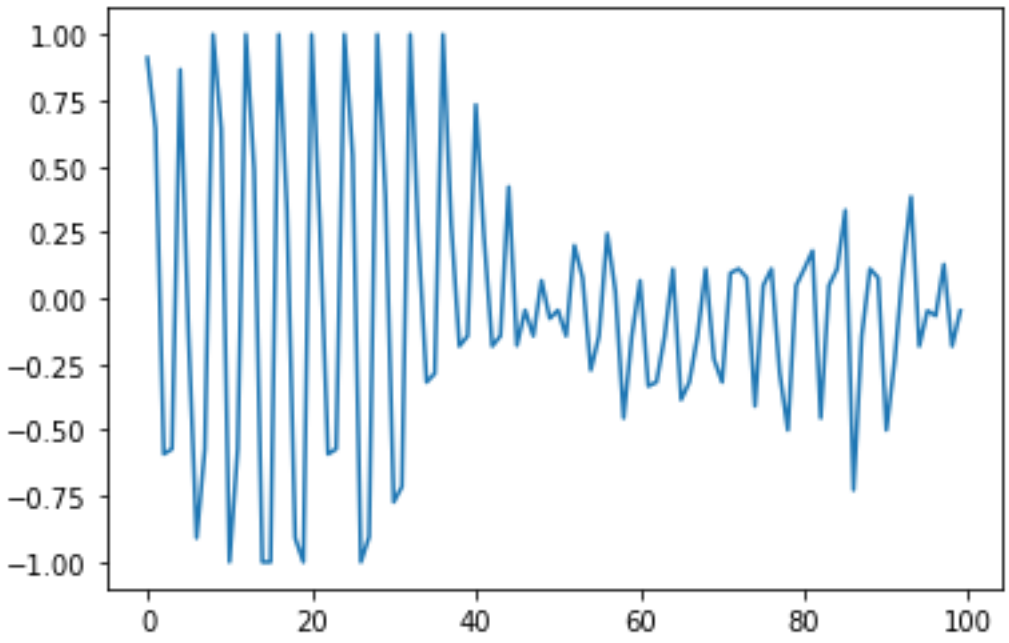
6. Теперь создаем наборы данных. Мы будем использовать 1 и 2 группы в качестве данных для обучения и 3 группы в качестве тестовых данных. Обратите внимание, что все данные имеют разную длину, и мы дополняем все данные до максимальной длины.

```
  def create_dataset(sequences, targets, groups, max_len = None):
      '''
      Create train and test datasets from raw data.
          The data set is created with padding all instance
          to the max_len variable.
```

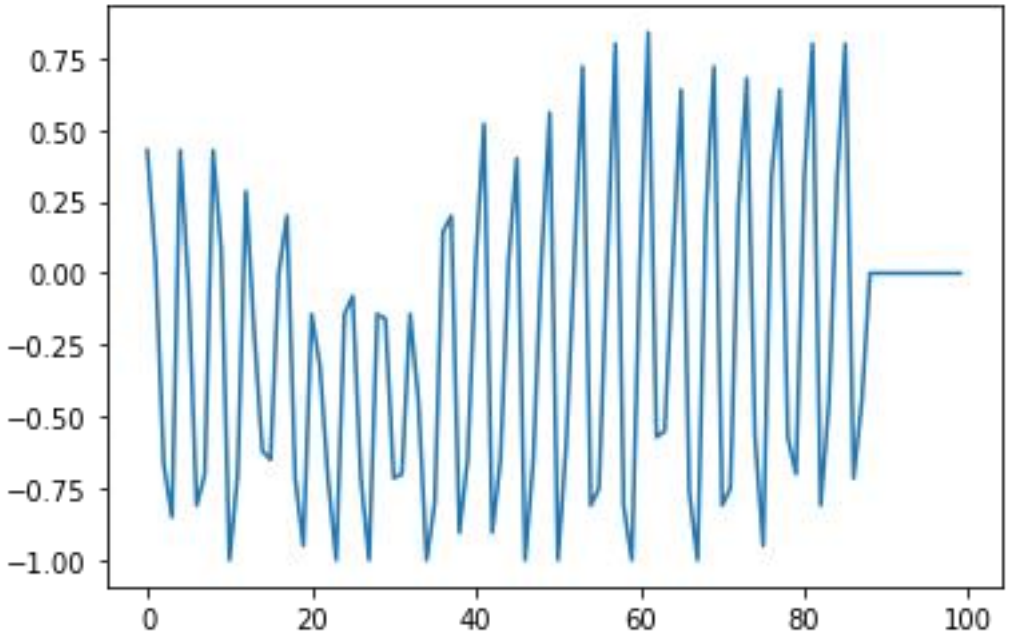
```
  Here group 1 and 2 are appended to the train data,
    and froup 3 for test data.

Parameters
----------
sequences: list,
    data of 4-th sensors for each instance (human).
targets: 1d ndarray,
    targets for each instance +1 or -1.
groups: 1d ndarray,
    group id for each instance 1,2 or 3.
max_len: int or None,
    if None: max_len  is the maximum length of instance.
    if int: all instance with length higher will be cut,
        all instance with length smaller
        will be padded at the end.

Returns
---------
x_train,x_test: 2d ndarrays,
    train and test data.
y_train,y_test: 1d ndarrays,
    train and test targets (labels).

'''
if max_len is None:
    max_len =0
    for i in range(len(sequences)):
        if len(sequences[i])>max_len:
            max_len = len(sequences[i])
else:
    max_len = int(max_len)

labels, counts = np.unique(groups, return_counts=True)

test_size  = counts[2]
train_size = counts[0]+counts[1]

x_train = np.zeros((train_size,max_len))
y_train = np.zeros(train_size)

x_test = np.zeros((test_size,max_len))
y_test = np.zeros(test_size)

cnt_test  = 0
cnt_train = 0

for i in range(len(sequences)):

    signal = np.squeeze(np.asarray(sequences[i]))
```

```
        if (max_len-len(signal) >0):
            signal = np.pad(signal,(0,max_len-len(signal)))
        elif (max_len-len(signal) <0):
            signal = signal[:max_len]


        if groups[i]==3:
            x_test[cnt_test] = signal
            y_test[cnt_test] = targets[i]
            cnt_test +=1
        else:
            x_train[cnt_train] = signal
            y_train[cnt_train] = targets[i]
            cnt_train +=1

    return x_train,x_test,y_train,y_test

x_train,x_test,y_train,y_test = create_dataset(sequences, targets, groups,
max_len = 100)

print('train :',x_train.shape,y_train.shape,'test :',
x_test.shape,y_test.shape)

plt.plot(x_train[90])
print(y_train[90])
plt.show()

plt.plot(x_test[100])
print(y_test[100])
plt.show()
```

```
train : (210, 100) (210,) test : (104, 100) (104,)
1.0
```




```
-1.0
```



7. Задача этого упражнения - поиск лучшего классификатора для описываемой проблемы.