Анализ временных рядов с использованием глубокого обучения нейронных сетей

Использование методов глубокого обучения в анализе временных рядов. Исслед ование одномерной сверточной нейронной сети в задаче классификации времен ных рядов. Исследование одномерной сверточной нейронной сети в задаче регре ссии временных рядов.

Импорт библиотек и данных

```
!pip install -U sktime
```

```
from sklearn.preprocessing import StandardScaler

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import GlobalAvgPool1D
from keras.layers import Dropout
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.utils import to_categorical

from sktime.datasets import load_italy_power_demand
from sktime.utils.data_processing import from_nested_to_2d_array

import matplotlib.pyplot as plt
import numpy as np

import pandas as pd

%matplotlib inline
```

Здесь мы будем использовать набор данных о потреблении электроэнергии в Италии.

```
xdf, ydf = load_italy_power_demand( return_X_y=True)
print(xdf.shape,ydf.shape)
```

```
(1096, 1) (1096,)
```

Давайте преобразуем набор в традиционную форму.

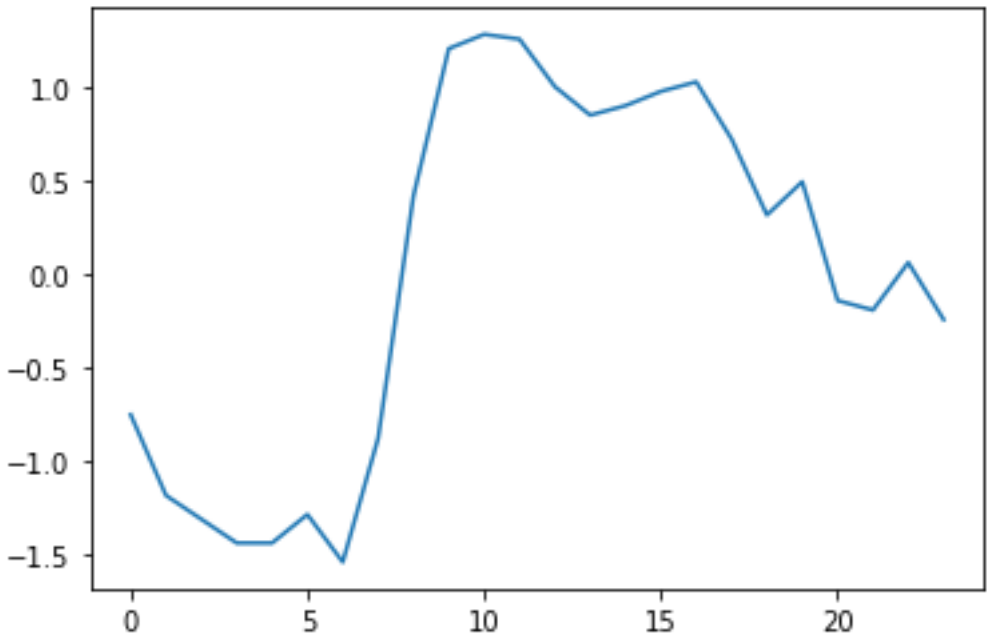
```
x = from_nested_to_2d_array(xdf)

x = x.values
print(x.shape)

y = ydf.values
print(y.shape)
```

```
(1096, 24)
(1096,)
```

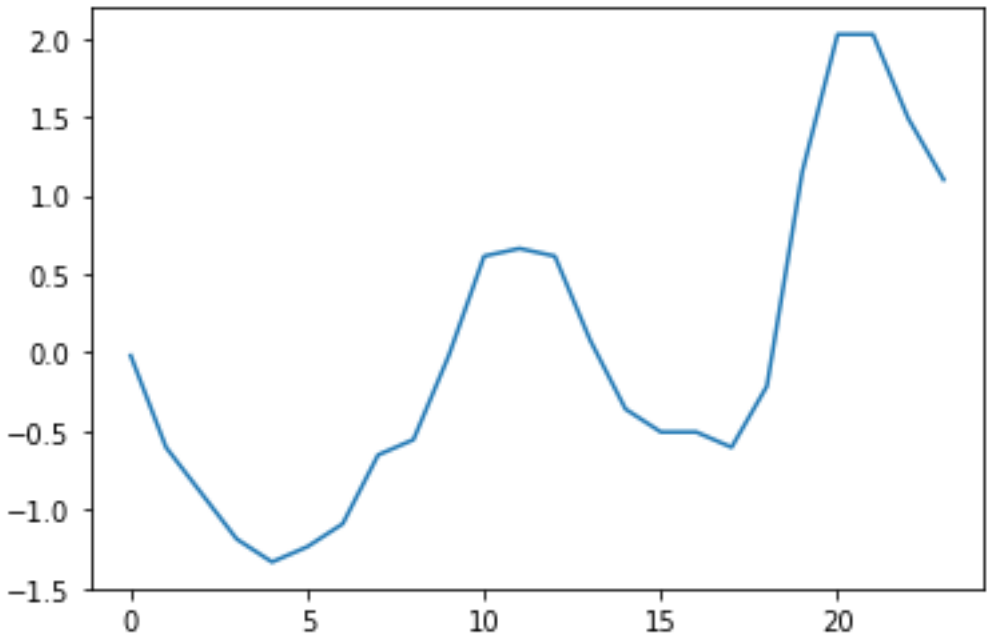
```
labels, count = np.unique(y, return_counts=True)
print('labels',lables, 'count', count)

y = [0 if yi=='2' else 1 for yi in y]

labels ['1' '2'] count [547 549]

y =np.array(y)
labels, count = np.unique(y, return_counts=True)
print('labels',labels, 'count', count)

labels [0 1] count [549 547]
```

И визуализируем его

```
instance = 9
plt.plot(x[instance,:])
print(y[instance]);
plt.show()
instance = 99
plt.plot(x[instance,:])
print(y[instance]);

0
```




```
1
```



Теперь мы можем сделать разделение на тренировочную и тестовую выборки и преобразовать наборы данных, в обычную форму для keras.

```
TEST_SIZE = int(x.shape[0]*0.5)

x_train,x_test = x[:TEST_SIZE,:,np.newaxis],x[-TEST_SIZE:,:,np.newaxis]
y_train,y_test = y[:TEST_SIZE],y[-TEST_SIZE:]

y_train.shape,y_test.shape,x_train.shape,x_test.shape
```

```
((548,), (548,), (548, 24, 1), (548, 24, 1))
```

Here we will build a base-line model for classification

```
epochs = 100
batch_size = 32


n_timesteps = x_train.shape[1]
n_features  = 1
n_outputs   = labels.shape[0]

# # scale data
# x_train, x_test = scale_data(x_train, x_test, param)

model = Sequential()
#24
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape
=(n_timesteps,n_features)))
#22x64
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
```

```
        #20x64*64
        model.add(Dropout(0.5))
        model.add(MaxPooling1D(pool_size=2))

        #10x64*64
        model.add(GlobalAvgPool1D())
        model.add(Dense(1, activation='sigmoid'))

        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accu
        racy'])

        model.summary()

        # # fit network
```

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_46 (Conv1D)           (None, 22, 64)            256
_________________________________________________________________
conv1d_47 (Conv1D)           (None, 20, 64)            12352
_________________________________________________________________
dropout_23 (Dropout)         (None, 20, 64)            0
_________________________________________________________________
max_pooling1d_23 (MaxPooling (None, 10, 64)            0
_________________________________________________________________
global_average_pooling1d_21  (None, 64)                0
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 65
=================================================================
Total params: 12,673
Trainable params: 12,673
Non-trainable params: 0
_________________________________________________________________
```

Теперь попытаемся провести обучение

```
        model.fit(x_train,
                  y_train,
                  epochs=epochs,
                  batch_size=batch_size,
                  verbose=1)
```

```
Epoch 1/100
18/18 [==============================] - 1s 3ms/step - loss: 0.6926 - accuracy:
0.4918
Epoch 2/100
18/18 [==============================] - 0s 3ms/step - loss: 0.6820 - accuracy:
0.5846
```

```
Epoch 3/100
18/18 [==============================] - 0s 3ms/step - loss: 0.6740 - accuracy:
0.6051
…

Epoch 31/100
18/18 [============================..] - ETA: 0s - loss: 0.4392 - accuracy: 0.87
```

Теперь попробуем оценить результаты

```
# evaluate model
_, accuracy = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
```

```
18/18 [==============================] - 0s 940us/step - loss: 0.1940 - accuracy
: 0.9416
```

Упражнение 1

1. Постарайтесь повысить точность модели CNN, используя свой опыт в архитектурах нейронных сетей.
2. Сравните полученные результаты CNN с полученными для классических методов классификации для того же набора данных (см. работу №6).
3. Выберите один из наборов данных для классификации, изученных в предыдущих работах, и попробуйте построить 1D-CNN классификатор для него.

Упражнение 2

1. Загрузите набор данных для VAR регрессии из работы №7 и попробуйте построить для него модель регрессии с использованием CNN (каждый столбец данных можно рассматривать как отдельный канал входных данных).