МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика, искусственный интеллект и системы управления»
Кафедра «Системы обработки информации и управления»

ОТЧЕТ

Домашнее задание № 3
по дисциплине «Методы поддержки принятия решений»

Тема: «Методы поддержки принятия решений на основе методов вычислительного интеллекта»

ИСПОЛНИТЕЛЬ: Терентьев В.О.
ФИО
группа ИУ5-73Б _____
подпись
"__"_____2021 г.

ПРЕПОДАВАТЕЛЬ: Терехов В.И.
ФИО
_____
подпись
"__"_____2021 г.

Москва - 2021

1. Задание

1. Задание 1.
    a. Даны три временных ряда, определите какие из них связаны между собой.
    b. Решить задачу, как минимум двумя методами, сравнить методы между собой и обосновать их выбор для решения данной задачи.
2. Задание 2.
    a. Дано множество пар чисел. Разделить их на 2 класса примерно одинаковой мощности.
    b. Оценить качество разделения двумя методами.
    c. Обосновать выбор методов решения задачи.

2. Используемые технологии

Язык программирования Python, библиотека для машинного обучения TensorFlow и Jupyter Notebook.

3. Выполнение работы

3.1. Задание 1

Метод с использованием рекуррентной нейронной сети с простым полносвязным слоем, где выход возвращается на вход.

In [1]:
```
import tensorflow as tf
import pandas as pd
import os
```

Загрузка данных:

In [2]:
```
series1 = pd.read_csv("ds/time series 1.txt", header=None)
series2 = pd.read_csv("ds/time series 2.txt", header=None)
series3 = pd.read_csv("ds/time series 3.txt", header=None)
```

In [3]:
```
timesteps = series1.shape[0]
batch_size = 1
feature = series1.shape[1]

dataset1 = tf.keras.utils.timeseries_dataset_from_array(
    data=series1,
    targets=None,
    sequence_length=timesteps,
    batch_size=batch_size
)
dataset2 = tf.keras.utils.timeseries_dataset_from_array(
    data=series2,
    targets=None,
    sequence_length=timesteps,
    batch_size=batch_size
)
dataset3 = tf.keras.utils.timeseries_dataset_from_array(
    data=series3,
    targets=None,
    sequence_length=timesteps,
    batch_size=batch_size
)
```

```
2021-12-23 09:03:38.920326: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optim
ized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-criti
cal operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Построение модели для поиска зависимости между 1 и 2 временными рядами:

In [4]:
```
model12 = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(64, return_sequences=True),
    tf.keras.layers.Dense(1)
])
model12.compile(loss=tf.losses.MeanSquaredError(),
                optimizer=tf.keras.optimizers.SGD(),
                metrics=[tf.metrics.MeanAbsoluteError()])
```

In [5]:
```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='loss', mode='min', factor=0.1, min_lr=0.0000001, patien
early_stop = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3000, verbose=1, mode='min', restore_best_
#model_cp_cb12 = tf.keras.callbacks.ModelCheckpoint(filepath="./model12", monitor='loss', save_best_only=True, mo
```

Обучение модели:

In [6]:
```
if os.path.isfile("./model0_12.index"):
    print('\nLoaded!\n')
    model12.load_weights("./model0_12")
else:
    model12.fit(tf.data.Dataset.zip((dataset1, dataset2)), epochs=10000, callbacks=[reduce_lr, early_stop])
    model12.save_weights("./model0_12")
```

```
1/1 [==============================] - 0s 6ms/step - loss: 5.7868e-07 - mean_absolute_error: 5.3167e-04
Epoch 8420/10000
1/1 [==============================] - 0s 7ms/step - loss: 5.7875e-07 - mean_absolute_error: 5.3169e-04
Epoch 8421/10000
1/1 [==============================] - 0s 6ms/step - loss: 5.7870e-07 - mean_absolute_error: 5.3167e-04
Epoch 8422/10000
1/1 [==============================] - 0s 7ms/step - loss: 5.7864e-07 - mean_absolute_error: 5.3166e-04
Epoch 8423/10000
1/1 [==============================] - 0s 6ms/step - loss: 5.7862e-07 - mean_absolute_error: 5.3162e-04
Epoch 8424/10000
1/1 [==============================] - 0s 7ms/step - loss: 5.7865e-07 - mean_absolute_error: 5.3163e-04
Epoch 8425/10000
1/1 [==============================] - 0s 7ms/step - loss: 5.7859e-07 - mean_absolute_error: 5.3168e-04
Epoch 8426/10000
1/1 [==============================] - 0s 7ms/step - loss: 5.7865e-07 - mean_absolute_error: 5.3170e-04
Epoch 8427/10000
1/1 [==============================] - 0s 6ms/step - loss: 5.7866e-07 - mean_absolute_error: 5.3169e-04
Epoch 8428/10000
1/1 [==============================] - 0s 6ms/step - loss: 5.7870e-07 - mean_absolute_error: 5.3170e-04
Epoch 8429/10000
```

Построение модели для поиска зависимости между 1 и 3 временными рядами:

In [7]:
```
model13 = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(64, return_sequences=True),
    tf.keras.layers.Dense(1)
])
model13.compile(loss=tf.losses.MeanSquaredError(),
                optimizer=tf.keras.optimizers.SGD(),
                metrics=[tf.metrics.MeanAbsoluteError()])
```

Обучение модели:

In [8]:
```
if os.path.isfile("./model0_13.index"):
    print('\nLoaded!\n')
    model13.load_weights("./model0_13")
else:
    model13.fit(tf.data.Dataset.zip((dataset1, dataset3)), epochs=10000, callbacks=[reduce_lr, early_stop])
    model13.save_weights("./model0_13")
```

```
1/1 [==============================] - 0s 6ms/step - loss: 3.2410e-10 - mean_absolute_error: 1.0720e-05
Epoch 7351/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.2410e-10 - mean_absolute_error: 1.0720e-05
Epoch 7352/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.2410e-10 - mean_absolute_error: 1.0720e-05
Epoch 7353/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.2410e-10 - mean_absolute_error: 1.0720e-05
Epoch 7354/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.2410e-10 - mean_absolute_error: 1.0720e-05
Epoch 7355/10000
1/1 [==============================] - 0s 6ms/step - loss: 3.2410e-10 - mean_absolute_error: 1.0720e-05
Epoch 7356/10000
1/1 [==============================] - 0s 6ms/step - loss: 3.2410e-10 - mean_absolute_error: 1.0720e-05
Epoch 7357/10000
1/1 [==============================] - 0s 6ms/step - loss: 3.2410e-10 - mean_absolute_error: 1.0720e-05
Epoch 7358/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.2410e-10 - mean_absolute_error: 1.0720e-05
Epoch 7359/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.2410e-10 - mean_absolute_error: 1.0720e-05
Epoch 7360/10000
```

Построение модели для поиска зависимости между 2 и 3 временными рядами:

In [9]:
```
model23 = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(64, return_sequences=True),
    tf.keras.layers.Dense(1)
])
model23.compile(loss=tf.losses.MeanSquaredError(),
                optimizer=tf.keras.optimizers.SGD(),
                metrics=[tf.metrics.MeanAbsoluteError()])
```

Обучение модели:

In [10]:
```
if os.path.isfile("./model0_23.index"):
    print('\nLoaded!\n')
    model23.load_weights("./model0_23")
else:
    model23.fit(tf.data.Dataset.zip((dataset2, dataset3)), epochs=10000, callbacks=[reduce_lr, early_stop])
    model23.save_weights("./model0_23")
```

```
Epoch 6878/10000
1/1 [==============================] - 0s 6ms/step - loss: 3.5933e-11 - mean_absolute_error: 2.1149e-06
Epoch 6879/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.5933e-11 - mean_absolute_error: 2.1149e-06
Epoch 6880/10000
1/1 [==============================] - 0s 6ms/step - loss: 3.5933e-11 - mean_absolute_error: 2.1149e-06
Epoch 6881/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.5933e-11 - mean_absolute_error: 2.1149e-06
Epoch 6882/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.5933e-11 - mean_absolute_error: 2.1149e-06
Epoch 6883/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.5933e-11 - mean_absolute_error: 2.1149e-06
Epoch 6884/10000
1/1 [==============================] - 0s 6ms/step - loss: 3.5933e-11 - mean_absolute_error: 2.1149e-06
Epoch 6885/10000
1/1 [==============================] - 0s 6ms/step - loss: 3.5933e-11 - mean_absolute_error: 2.1149e-06
Epoch 6886/10000
1/1 [==============================] - 0s 6ms/step - loss: 3.5933e-11 - mean_absolute_error: 2.1149e-06
Epoch 6887/10000
1/1 [==============================] - 0s 7ms/step - loss: 3.5933e-11 - mean_absolute_error: 2.1149e-06
```

Анализ результатов:

In [11]:

```
print("Input (series1)\t\tmodel12\t\t\tseries2\t\t\tmodel13\t\t\tseries3")
mae12 = tf.metrics.MeanAbsoluteError()
mae13 = tf.metrics.MeanAbsoluteError()
for step in range(1, series1.shape[0]+1):
    mae12.reset_state()
    mae13.reset_state()
    test_dataset1 = tf.keras.utils.timeseries_dataset_from_array(
        data=series1,
        targets=None,
        sequence_length=step,
        batch_size=batch_size
    )
    test_dataset2 = tf.keras.utils.timeseries_dataset_from_array(
        data=series2,
        targets=None,
        sequence_length=step,
        batch_size=batch_size
    )
    test_dataset3 = tf.keras.utils.timeseries_dataset_from_array(
        data=series3,
        targets=None,
        sequence_length=step,
        batch_size=batch_size
    )
    test_dataset123 =  tf.data.Dataset.zip((test_dataset1, test_dataset2, test_dataset3))
    print('\n')

    for x in test_dataset123.take(1):
        y12 = model12(x[0]).numpy()[0]
        y13 = model13(x[0]).numpy()[0]
        mae12.update_state(x[1].numpy()[0], y12)
        mae13.update_state(x[2].numpy()[0], y13)
        for i in range(len(y12)):
            print('{:.15f}\t{:.15f}\t{:.15f}\t{:.15f}\t{:.15f}'.format(x[0].numpy()[0][i][0], y12[i][0], x[1].num
print('model12 MAE: ', mae12.result().numpy())
print('model13 MAE: ', mae13.result().numpy())
```

```
Input (series1)         model12                 series2                 model13                 series3


1.000000000000000       0.999490499496460       1.000000000000000       1.000102400779724       1.000000000000
000


1.000000000000000       0.999490499496460       1.000000000000000       1.000102519989014       1.000000000000
000
1.000000000000000       0.999860167503357       1.000000000000000       0.999979913234711       1.000000000000
000


1.000000000000000       0.999490499496460       1.000000000000000       1.000102519989014       1.000000000000
000
1.000000000000000       0.999860167503357       1.000000000000000       0.999979913234711       1.000000000000
000
-0.300000000000000      -1.599960088729858      -1.600000000000000      -0.999990224838257      -1.000000000000
0000
```

In [12]:
```
print("Input (series2)\t\tmodel23\t\t\tseries3")
mae = tf.metrics.MeanAbsoluteError()
for step in range(1, series1.shape[0]+1):
    mae.reset_state()
    test_dataset2 = tf.keras.utils.timeseries_dataset_from_array(
        data=series2,
        targets=None,
        sequence_length=step,
        batch_size=batch_size
    )
    test_dataset3 = tf.keras.utils.timeseries_dataset_from_array(
        data=series3,
        targets=None,
        sequence_length=step,
        batch_size=batch_size
    )
    test_dataset23 =  tf.data.Dataset.zip((test_dataset2, test_dataset3))
    print('\n')

    for x in test_dataset23.take(1):
        y = model23(x[0]).numpy()[0]
        mae.update_state(x[1].numpy()[0], y)
        for i in range(len(y)):
            print('{:.15f}\t{:.15f}\t{:.15f}'.format(x[0].numpy()[0][i][0], y[i][0], x[1].numpy()[0][i][0]))
print('model23 MAE: ', mae.result().numpy())
```

```
Input (series2)         model23                 series3


1.000000000000000       1.000042915344238       1.000000000000000


1.000000000000000       1.000042915344238       1.000000000000000
1.000000000000000       0.999984502792358       1.000000000000000


1.000000000000000       1.000042915344238       1.000000000000000
1.000000000000000       0.999984502792358       1.000000000000000
-1.600000000000000      -0.999991774559021      -1.000000000000000


1.000000000000000       1.000042915344238       1.000000000000000
1.000000000000000       0.999984502792358       1.000000000000000
-1.600000000000000      -0.999991774559021      -1.000000000000000
1.129999999999999       -1.200001716613770      -1.200000000000000
```

In [ ]:

Данная модель нейронной сети смогла найти зависимость между всеми временными рядами. Средняя абсолютная ошибка найденной зависимости между 1 и 2 рядами составила около $5{,}3 \cdot 10^{-4}$, между 1 и 3 рядами – около $1{,}1 \cdot 10^{-5}$, между 2 и 3 рядами – около $2{,}2 \cdot 10^{-6}$. Наглядно увидеть работоспособность данных сетей можно в анализе результатов, где на вход моделям с зависимостями между 1 и 2 временными рядами и между 1 и 3 рядами подаются значения первого временного ряда и выводится их вывод, а модели с зависимостью между 2 и 3 рядами подаются значения второго временного ряда.

Метод с использованием рекуррентной нейронной сети с LSTM слоем (длинной цепью элементов краткосрочной памяти).

In [1]:
```
import tensorflow as tf
import pandas as pd
import os
```

Загрузка данных:

In [2]:
```
series1 = pd.read_csv("ds/time series 1.txt", header=None)
series2 = pd.read_csv("ds/time series 2.txt", header=None)
series3 = pd.read_csv("ds/time series 3.txt", header=None)
```

In [3]:
```
timesteps = series1.shape[0]
batch_size = 1
feature = series1.shape[1]

dataset1 = tf.keras.utils.timeseries_dataset_from_array(
    data=series1,
    targets=None,
    sequence_length=timesteps,
    batch_size=batch_size
)
dataset2 = tf.keras.utils.timeseries_dataset_from_array(
    data=series2,
    targets=None,
    sequence_length=timesteps,
    batch_size=batch_size
)
dataset3 = tf.keras.utils.timeseries_dataset_from_array(
    data=series3,
    targets=None,
    sequence_length=timesteps,
    batch_size=batch_size
)
```

```
2021-12-23 09:46:44.945796: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized wit
h oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:
AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Построение модели для поиска зависимости между 1 и 2 временными рядами:

In [4]:
```
model12 = tf.keras.Sequential([
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.Dense(1),
])
model12.compile(loss=tf.losses.MeanSquaredError(),
                optimizer=tf.keras.optimizers.SGD(),
                metrics=[tf.metrics.MeanAbsoluteError()])
```

In [5]:
```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='loss', mode='min', factor=0.1, min_lr=0.0000001, patience=1000,
early_stop = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3000, verbose=1, mode='min', restore_best_weights=T
#model_cp_cb12 = tf.keras.callbacks.ModelCheckpoint(filepath="./model12", monitor='loss', save_best_only=True, mode='min')
```

Обучение модели:

In [6]:
```
if os.path.isfile("./model12.index"):
    print('\nLoaded!\n')
    model12.load_weights("./model12")
else:
    model12.fit(tf.data.Dataset.zip((dataset1, dataset2)), epochs=10000, callbacks=[reduce_lr, early_stop])
    model12.save_weights("./model12")
```

```
1/1 [==============================] - 0s 15ms/step - loss: 5.2757e-04 - mean_absolute_error: 0.0185
Epoch 8176/10000
1/1 [==============================] - 0s 16ms/step - loss: 5.2789e-04 - mean_absolute_error: 0.0190
Epoch 8177/10000
1/1 [==============================] - 0s 15ms/step - loss: 5.2703e-04 - mean_absolute_error: 0.0185
Epoch 8178/10000
1/1 [==============================] - 0s 13ms/step - loss: 5.2736e-04 - mean_absolute_error: 0.0189
Epoch 8179/10000
1/1 [==============================] - 0s 15ms/step - loss: 5.2649e-04 - mean_absolute_error: 0.0185
Epoch 8180/10000
1/1 [==============================] - 0s 15ms/step - loss: 5.2682e-04 - mean_absolute_error: 0.0189
Epoch 8181/10000
1/1 [==============================] - 0s 15ms/step - loss: 5.2596e-04 - mean_absolute_error: 0.0185
Epoch 8182/10000
1/1 [==============================] - 0s 15ms/step - loss: 5.2629e-04 - mean_absolute_error: 0.0189
Epoch 8183/10000
1/1 [==============================] - 0s 14ms/step - loss: 5.2544e-04 - mean_absolute_error: 0.0185
Epoch 8184/10000
1/1 [==============================] - 0s 13ms/step - loss: 5.2577e-04 - mean_absolute_error: 0.0189
```

Построение модели для поиска зависимости между 1 и 3 временными рядами:

In [7]:
```
model13 = tf.keras.Sequential([
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.Dense(1),
])
model13.compile(loss=tf.losses.MeanSquaredError(),
                optimizer=tf.keras.optimizers.SGD(),
                metrics=[tf.metrics.MeanAbsoluteError()])
```

Обучение модели:

In [8]:
```
if os.path.isfile("./model13.index"):
    print('\nLoaded!\n')
    model13.load_weights("./model13")
else:
    model13.fit(tf.data.Dataset.zip((dataset1, dataset3)), epochs=10000, callbacks=[reduce_lr, early_stop])
    model13.save_weights("./model13")
```

```
Epoch 8235/10000
1/1 [==============================] - 0s 15ms/step - loss: 0.0018 - mean_absolute_error: 0.0322
Epoch 8236/10000
1/1 [==============================] - 0s 15ms/step - loss: 0.0018 - mean_absolute_error: 0.0335
Epoch 8237/10000
1/1 [==============================] - 0s 14ms/step - loss: 0.0018 - mean_absolute_error: 0.0322
Epoch 8238/10000
1/1 [==============================] - 0s 16ms/step - loss: 0.0018 - mean_absolute_error: 0.0334
Epoch 8239/10000
1/1 [==============================] - 0s 15ms/step - loss: 0.0018 - mean_absolute_error: 0.0321
Epoch 8240/10000
1/1 [==============================] - 0s 14ms/step - loss: 0.0018 - mean_absolute_error: 0.0333
Epoch 8241/10000
1/1 [==============================] - 0s 14ms/step - loss: 0.0018 - mean_absolute_error: 0.0320
Epoch 8242/10000
1/1 [==============================] - 0s 16ms/step - loss: 0.0018 - mean_absolute_error: 0.0332
Epoch 8243/10000
1/1 [==============================] - 0s 14ms/step - loss: 0.0018 - mean_absolute_error: 0.0320
Epoch 8244/10000
1/1 [==============================] - 0s 14ms/step - loss: 0.0018 - mean_absolute_error: 0.0332
```

Построение модели для поиска зависимости между 2 и 3 временными рядами:

In [9]:
```
model23 = tf.keras.Sequential([
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.Dense(1),
])
model23.compile(loss=tf.losses.MeanSquaredError(),
                optimizer=tf.keras.optimizers.SGD(),
                metrics=[tf.metrics.MeanAbsoluteError()])
```

Обучение модели:

In [10]:
```
if os.path.isfile("./model23.index"):
    print('\nLoaded!\n')
    model23.load_weights("./model23")
else:
    model23.fit(tf.data.Dataset.zip((dataset2, dataset3)), epochs=10000, callbacks=[reduce_lr, early_stop])
    model23.save_weights("./model23")
```

```
1/1 [==============================] - 0s 14ms/step - loss: 2.8581e-05 - mean_absolute_error: 0.0027
```

Анализ результатов:

In [11]:
```
print("Input (series1)\t\tmodel12\t\t\tseries2\t\t\tmodel13\t\t\tseries3")
mae12 = tf.metrics.MeanAbsoluteError()
mae13 = tf.metrics.MeanAbsoluteError()
for step in range(1, series1.shape[0]+1):
    mae12.reset_state()
    mae13.reset_state()
    test_dataset1 = tf.keras.utils.timeseries_dataset_from_array(
        data=series1,
        targets=None,
        sequence_length=step,
        batch_size=batch_size
    )
    test_dataset2 = tf.keras.utils.timeseries_dataset_from_array(
        data=series2,
        targets=None,
        sequence_length=step,
        batch_size=batch_size
    )
    test_dataset3 = tf.keras.utils.timeseries_dataset_from_array(
        data=series3,
        targets=None,
        sequence_length=step,
        batch_size=batch_size
    )
    test_dataset123 =  tf.data.Dataset.zip((test_dataset1, test_dataset2, test_dataset3))
    print('\n')

    for x in test_dataset123.take(1):
        y12 = model12(x[0]).numpy()[0]
        y13 = model13(x[0]).numpy()[0]
        mae12.update_state(x[1].numpy()[0], y12)
        mae13.update_state(x[2].numpy()[0], y13)
        for i in range(len(y12)):
            print('{:.15f}\t{:.15f}\t{:.15f}\t{:.15f}\t{:.15f}'.format(x[0].numpy()[0][i][0], y12[i][0], x[1].numpy()[0][i
print('model12 MAE: ', mae12.result().numpy())
print('model13 MAE: ', mae13.result().numpy())
```

```
Input (series1)         model12                 series2                 model13                 series3


1.000000000000000       1.022193908691406       1.000000000000000       0.993919074535370       1.000000000000000


1.000000000000000       1.022194266319275       1.000000000000000       0.993918955326080       1.000000000000000
1.000000000000000       0.979835271835327       1.000000000000000       1.016656517982483       1.000000000000000


1.000000000000000       1.022194266319275       1.000000000000000       0.993918955326080       1.000000000000000
1.000000000000000       0.979835271835327       1.000000000000000       1.016656517982483       1.000000000000000
-0.300000000000000      -1.590558767318726      -1.600000000000000      -1.010732889175415      -1.000000000000000


1.000000000000000       1.022194266319275       1.000000000000000       0.993918955326080       1.000000000000000
1.000000000000000       0.979835271835327       1.000000000000000       1.016656517982483       1.000000000000000
-0.300000000000000      -1.590558767318726      -1.600000000000000      -1.010732889175415      -1.000000000000000
-1.470000000000000      1.119035005569458       1.129999999999999       -1.181395769119263      -1.200000000000000
```

In [12]:
```
print("Input (series2)\t\tmodel23\t\t\tseries3")
mae = tf.metrics.MeanAbsoluteError()
for step in range(1, series1.shape[0]+1):
    mae.reset_state()
    test_dataset2 = tf.keras.utils.timeseries_dataset_from_array(
        data=series2,
        targets=None,
        sequence_length=step,
        batch_size=batch_size
    )
    test_dataset3 = tf.keras.utils.timeseries_dataset_from_array(
        data=series3,
        targets=None,
        sequence_length=step,
        batch_size=batch_size
    )
    test_dataset23 =  tf.data.Dataset.zip((test_dataset2, test_dataset3))
    print('\n')

    for x in test_dataset23.take(1):
        y = model23(x[0]).numpy()[0]
        mae.update_state(x[1].numpy()[0], y)
        for i in range(len(y)):
            print('{:.15f}\t{:.15f}\t{:.15f}'.format(x[0].numpy()[0][i][0], y[i][0], x[1].numpy()[0][i][0]))
print('model23 MAE: ', mae.result().numpy())
```

```
Input (series2)         model23                 series3


1.000000000000000       0.999900460243225       1.000000000000000


1.000000000000000       0.999900400638580       1.000000000000000
1.000000000000000       1.001101732254028       1.000000000000000


1.000000000000000       0.999900400638580       1.000000000000000
1.000000000000000       1.001101732254028       1.000000000000000
-1.600000000000000      -1.002073526382446      -1.000000000000000


1.000000000000000       0.999900400638580       1.000000000000000
1.000000000000000       1.001101732254028       1.000000000000000
-1.600000000000000      -1.002073526382446      -1.000000000000000
1.129999999999999       -1.194580078125000      -1.200000000000000
```

In [ ]:

Данная модель нейронной сети также смогла найти зависимость между всеми временными рядами, однако значения средних абсолютных ошибок возросли, по сравнению с предыдущей моделью. Наглядно увидеть работоспособность этих моделей можно также в анализе результатов.

В качестве нейронных сетей были выбраны рекуррентные нейронные сети, потому что они способны выдавать выходные значения на основе предыдущих данных, а не только текущих, тем самым, находить сложные зависимости между временными рядами, основанные на предыдущих значениях.

3.2. Задание 2

В качестве способа кластеризации входных векторов использована нейронная сеть Кохонена. Реализовал модель и алгоритм обучения с нормировкой входных векторов и весов каждого нейрона.

In [1]:
```
import tensorflow as tf
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
```

In [2]:
```
input_data = tf.constant(np.loadtxt('Домашнее задание 3_вар2.txt'), dtype='float32')
input_data
```

Out[2]:
```
<tf.Tensor: shape=(1000, 2), dtype=float32, numpy=
array([[9.1636, 6.3339],
       [8.3872, 1.6484],
       [8.9726, 2.0461],
       ...,
       [8.2343, 1.5666],
       [8.1887, 4.5177],
       [8.5591, 5.1374]], dtype=float32)>
```

In [3]:
```
class KohonenNN(tf.Module):
    def __init__(self, num_outputs):
        super().__init__()
        self.num_outputs = num_outputs

    def build(self, input_shape):
        self.W = tf.Variable(self.norm(tf.constant(tf.random.uniform([self.num_outputs, int(input_shape[-1])]))), name="kernel")

    def __call__(self, inputs):
        y = []
        for X in self.norm(inputs):
            y.append(tf.math.argmax(tf.math.reduce_sum(tf.math.multiply(X, self.W), axis=1)).numpy())
        return tf.constant(y)

    def train(self, data_train, learning_rate=0.9, max_distance=2):
        data_train = self.norm(data_train)
        delt_max_distance = max_distance/(learning_rate/0.001)
        epch = 0
        sum_corr = 0
        while(True):
            for X in data_train:
                prev_sum_corr = sum_corr
                sum_corr = 0
                ind_min_dist = tf.math.argmax(tf.math.reduce_sum(tf.math.multiply(X, self.W), axis=1))
                for i in range(self.num_outputs):
                    if tf.math.sqrt(tf.math.reduce_sum(tf.math.pow(tf.math.subtract(self.W[ind_min_dist], self.W[i]), 2))) <= max_di
                        corr = tf.math.multiply(tf.math.subtract(X, self.W[i]), learning_rate)
                        sum_corr += tf.abs(tf.math.reduce_sum(corr))
                        new_w = tf.math.add(self.W[i], corr)
                        self.W[i].assign(tf.math.divide(new_w, tf.math.sqrt(tf.math.reduce_sum(tf.math.pow(new_w, 2)))))
                if (learning_rate-0.001) >= 0:
                    learning_rate -= 0.001
                if (max_distance-delt_max_distance) >= 0:
                    max_distance -= delt_max_distance
                if abs(sum_corr-prev_sum_corr) < 0.0000001:
                    print('Last Learning rate: {}\tLast Max distance: {}'.format(learning_rate, max_distance))
                    return
                print('Epoch: ', epch, '\tLoss: ', abs(sum_corr-prev_sum_corr).numpy())
                epch += 1

    def norm(self, inp):
        denoms = tf.math.sqrt(tf.math.reduce_sum(tf.math.pow(inp, 2), axis=1, keepdims=True))
        denoms = tf.concat([denoms, denoms], 1)
        return tf.math.divide(inp, denoms)
```

```
In [4]: knn = KohonenNN(2)
        knn.build(input_data.shape)
        y = knn(input_data)
        y
```

```
Out[4]: <tf.Tensor: shape=(1000,), dtype=int32, numpy=
        array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0])>
```

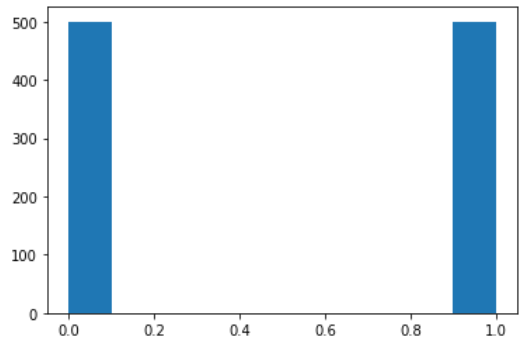
```
In [5]: knn.train(input_data)
```

```
Epoch:  1198    Loss:  8.652788e-05
Epoch:  1199    Loss:  4.6449204e-05
Epoch:  1200    Loss:  2.1882384e-05
Epoch:  1201    Loss:  1.7448634e-05
Epoch:  1202    Loss:  4.6739555e-05
Epoch:  1203    Loss:  5.2460335e-05
Epoch:  1204    Loss:  3.599732e-05
Epoch:  1205    Loss:  3.297715e-06
Epoch:  1206    Loss:  4.24817e-05
Epoch:  1207    Loss:  4.414816e-05
Epoch:  1208    Loss:  7.892227e-05
Epoch:  1209    Loss:  2.5114387e-05
Epoch:  1210    Loss:  4.0542065e-05
Epoch:  1211    Loss:  4.1628955e-05
Epoch:  1212    Loss:  0.000104433624
Epoch:  1213    Loss:  4.374142e-05
Epoch:  1214    Loss:  1.6099577e-05
Epoch:  1215    Loss:  2.465675e-05
Epoch:  1216    Loss:  4.3985536e-05
```

In [6]:
```
y = knn(input_data)
y
```

Out[6]:
```
<tf.Tensor: shape=(1000,), dtype=int32, numpy=
array([1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 1])>
```

In [7]:
```
n = plt.hist(y.numpy(), )
plt.show()
print('Мощность первого класса: ', int(n[0][0]), '\tМощность второго класса: ',  int(n[0][-1]))
```




```
Мощность первого класса:  500   Мощность второго класса:  500
```

In [8]:
```
# Средний коэффициент силуэта для евклидова расстояния
metrics.silhouette_score(input_data, y, metric='euclidean')
```

Out[8]:
```
0.7015748
```

In [9]:
```
# Средний коэффициент силуэта для косинусного сходства
metrics.silhouette_score(input_data, y, metric='cosine')
```

Out[9]:
```
0.9642009
```

In [10]:
```
# Индекс Дэвиса-Болдина
metrics.davies_bouldin_score(input_data, y)
```

Out[10]:
```
0.4400800708653689
```

In [ ]:
```

```

В результате кластеризации, данная сеть разделила входные векторы на два класса одинаковой мощности.

Для качества оценки разделения использованы метрики: средний коэффициент силуэта и индекс Дэвиса-Болдина. Для данных метрик не требуются истинные метки объектов, что как раз подходит данной задаче. Лучшее значение показал средний коэффициент силуэта для косинусного сходства (0,9642009), когда максимальное значение этой метрики, обозначающее максимальное качество разделения, равно 1,0.

4. Выводы

В результате выполнения домашнего задания были успешно найдены зависимости между всеми данными временными рядами с помощью двух моделей рекуррентных нейронных сетей, а также реализована сеть Кохонена, с помощью которой произведено разделение множества пар чисел на два класса.