РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра информационных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № **2**

Дисциплина: Методы машинного обучения

Студент: Шалыгин Георгий

Группа: НФИ-02

Москва **2023**

Вариант № **18**

1. Загрузите заданный в индивидуальном задании набор данных из **Tensorflow Datasets,** включая указанные в задании независимый признак и зависимый признак **(**отклик**).**

   Набор данных**: forest_fires.**

   Независимая переменная**: features/FFMC.**

   Зависимая переменная**: features/temp**

In [ ]:

```
import tensorflow_datasets as tfds
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [8]:

```
ds = tfds.load("forest_fires", split='train')
df = tfds.as_dataframe(ds)
df.info()
```

```
<class 'tensorflow_datasets.core.as_dataframe.StyledDataFrame'>
RangeIndex: 517 entries, 0 to 516
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   area            517 non-null    float32
 1   features/DC     517 non-null    float32
 2   features/DMC    517 non-null    float32
 3   features/FFMC   517 non-null    float32
 4   features/ISI    517 non-null    float32
 5   features/RH     517 non-null    float32
 6   features/X      517 non-null    uint8
 7   features/Y      517 non-null    uint8
 8   features/day    517 non-null    int64
 9   features/month  517 non-null    int64
 10  features/rain   517 non-null    float32
```

```
 11  features/temp   517 non-null    float32
 12  features/wind   517 non-null    float32
dtypes: float32(9), int64(2), uint8(2)
memory usage: 27.4 KB
```

In [9]:

```
#уберем выбросы
df = df[df['features/FFMC'] > 80]
```

1. Решите задачу полиномиальной регрессии для степени полинома, указанной в индивидуальном задании, при помощи нейронной сети с одним нейроном и оцените качество полученной модели по показателю, указанному в индивидуальном задании.

   Степень полинома: **3.**

   Показатель качества регрессии – **MSE (mean squared error)**

In [10]:

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
import tensorflow as tf
```

In [11]:

```
x = df['features/FFMC']
y = df['features/temp']
```

In [12]:

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=3)
X = poly.fit_transform(pd.DataFrame(x))
X
```

Out[12]:

```
array([[1.0000000e+00, 9.6099998e+01, 9.2352100e+03, 8.8750369e+05],
       [1.0000000e+00, 9.0500000e+01, 8.1902500e+03, 7.4121762e+05],
       [1.0000000e+00, 9.4300003e+01, 8.8924902e+03, 8.3856188e+05],
       ...,
       [1.0000000e+00, 9.0000000e+01, 8.1000000e+03, 7.2900000e+05],
       [1.0000000e+00, 8.9400002e+01, 7.9923604e+03, 7.1451700e+05],
       [1.0000000e+00, 9.1599998e+01, 8.3905596e+03, 7.6857525e+05]],
      dtype=float32)
```

In [13]:

```
model = Sequential(Dense(1, input_shape=(4,)))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 1)                 5

=================================================================
Total params: 5
Trainable params: 5
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```
model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(learning_rate=0.01))
history0 = model.fit(X, y, epochs=100, verbose=1, validation_split = 0.2)
```

```
Epoch 1/100
13/13 [==============================] - 1s 15ms/step - loss: 119379216.0000 - val_loss:
2973795.2500
Epoch 2/100
13/13 [==============================] - 0s 5ms/step - loss: 20347816.0000 - val_loss: 17
433640.0000
Epoch 3/100
13/13 [==============================] - 0s 4ms/step - loss: 7491957.0000 - val_loss: 248
7498.7500
Epoch 4/100
13/13 [==============================] - 0s 4ms/step - loss: 1307519.6250 - val_loss: 829
402.1875
Epoch 5/100
13/13 [==============================] - 0s 6ms/step - loss: 565715.9375 - val_loss: 1094
66.6328
Epoch 6/100
13/13 [==============================] - 0s 4ms/step - loss: 113819.5781 - val_loss: 1269
90.0703
Epoch 7/100
13/13 [==============================] - 0s 5ms/step - loss: 42185.8945 - val_loss: 19586
.4219
Epoch 8/100
13/13 [==============================] - 0s 3ms/step - loss: 16434.4395 - val_loss: 2659.
8562
Epoch 9/100
13/13 [==============================] - 0s 5ms/step - loss: 7096.1226 - val_loss: 2907.7
590
Epoch 10/100
13/13 [==============================] - 0s 4ms/step - loss: 4493.2363 - val_loss: 3138.7
163
Epoch 11/100
13/13 [==============================] - 0s 3ms/step - loss: 3580.7092 - val_loss: 2813.1
382
Epoch 12/100
13/13 [==============================] - 0s 5ms/step - loss: 3385.5947 - val_loss: 2627.5
432
Epoch 13/100
13/13 [==============================] - 0s 4ms/step - loss: 3330.6953 - val_loss: 2619.7
920
Epoch 14/100
13/13 [==============================] - 0s 5ms/step - loss: 3303.0657 - val_loss: 2646.3
250
Epoch 15/100
13/13 [==============================] - 0s 5ms/step - loss: 3344.6057 - val_loss: 2623.4
355
Epoch 16/100
13/13 [==============================] - 0s 5ms/step - loss: 3403.4895 - val_loss: 2596.2
371
Epoch 17/100
13/13 [==============================] - 0s 4ms/step - loss: 3379.2166 - val_loss: 3040.6
875
Epoch 18/100
13/13 [==============================] - 0s 5ms/step - loss: 3661.9424 - val_loss: 3115.8
992
Epoch 19/100
13/13 [==============================] - 0s 4ms/step - loss: 3327.3320 - val_loss: 2915.6
975
Epoch 20/100
13/13 [==============================] - 0s 3ms/step - loss: 3444.9070 - val_loss: 2710.4
817
Epoch 21/100
13/13 [==============================] - 0s 5ms/step - loss: 3355.6929 - val_loss: 2634.5
259
Epoch 22/100
13/13 [==============================] - 0s 4ms/step - loss: 3463.7859 - val_loss: 2841.4
585
Epoch 23/100
13/13 [==============================] - 0s 5ms/step - loss: 3453.2803 - val_loss: 2695.4
561
Epoch 24/100
13/13 [==============================] - 0s 4ms/step - loss: 3468.9619 - val_loss: 2580.7
942
```

```
Epoch 25/100
13/13 [==============================] - 0s 5ms/step - loss: 3412.7820 - val_loss: 2597.7
952
Epoch 26/100
13/13 [==============================] - 0s 3ms/step - loss: 3327.9944 - val_loss: 2691.6
843
Epoch 27/100
13/13 [==============================] - 0s 6ms/step - loss: 3381.5222 - val_loss: 2633.7
358
Epoch 28/100
13/13 [==============================] - 0s 4ms/step - loss: 3607.4414 - val_loss: 3080.8
391
Epoch 29/100
13/13 [==============================] - 0s 5ms/step - loss: 3697.8325 - val_loss: 2591.1
282
Epoch 30/100
13/13 [==============================] - 0s 5ms/step - loss: 3351.3110 - val_loss: 2777.6
680
Epoch 31/100
13/13 [==============================] - 0s 5ms/step - loss: 3442.8828 - val_loss: 2709.7
698
Epoch 32/100
13/13 [==============================] - 0s 5ms/step - loss: 3406.9583 - val_loss: 2573.5
745
Epoch 33/100
13/13 [==============================] - 0s 5ms/step - loss: 3470.1436 - val_loss: 2758.8
147
Epoch 34/100
13/13 [==============================] - 0s 5ms/step - loss: 3524.8660 - val_loss: 3360.3
376
Epoch 35/100
13/13 [==============================] - 0s 5ms/step - loss: 3530.3147 - val_loss: 2630.4
187
Epoch 36/100
13/13 [==============================] - 0s 6ms/step - loss: 3275.4651 - val_loss: 2735.5
903
Epoch 37/100
13/13 [==============================] - 0s 6ms/step - loss: 3475.3149 - val_loss: 2569.3
027
Epoch 38/100
13/13 [==============================] - 0s 6ms/step - loss: 3261.4829 - val_loss: 2547.0
696
Epoch 39/100
13/13 [==============================] - 0s 6ms/step - loss: 3302.9302 - val_loss: 2615.3
823
Epoch 40/100
13/13 [==============================] - 0s 5ms/step - loss: 3396.3230 - val_loss: 3336.4
824
Epoch 41/100
13/13 [==============================] - 0s 6ms/step - loss: 3548.2236 - val_loss: 2537.3
044
Epoch 42/100
13/13 [==============================] - 0s 6ms/step - loss: 3257.0020 - val_loss: 2945.9
976
Epoch 43/100
13/13 [==============================] - 0s 5ms/step - loss: 3751.4900 - val_loss: 3291.1
282
Epoch 44/100
13/13 [==============================] - 0s 6ms/step - loss: 3508.1987 - val_loss: 3148.3
435
Epoch 45/100
13/13 [==============================] - 0s 7ms/step - loss: 3406.5493 - val_loss: 2558.8
176
Epoch 46/100
13/13 [==============================] - 0s 4ms/step - loss: 3259.0432 - val_loss: 2564.1
226
Epoch 47/100
13/13 [==============================] - 0s 5ms/step - loss: 3217.2959 - val_loss: 2633.2
188
Epoch 48/100
13/13 [==============================] - 0s 6ms/step - loss: 3306.0186 - val_loss: 2650.2
722
```

```
Epoch 49/100
13/13 [==============================] - 0s 5ms/step - loss: 3264.5273 - val_loss: 2616.5
535
Epoch 50/100
13/13 [==============================] - 0s 5ms/step - loss: 3389.5044 - val_loss: 3160.6
099
Epoch 51/100
13/13 [==============================] - 0s 5ms/step - loss: 3743.8655 - val_loss: 2877.5
671
Epoch 52/100
13/13 [==============================] - 0s 6ms/step - loss: 3279.3394 - val_loss: 2863.2
312
Epoch 53/100
13/13 [==============================] - 0s 5ms/step - loss: 3541.9338 - val_loss: 2801.7
937
Epoch 54/100
13/13 [==============================] - 0s 5ms/step - loss: 3583.7485 - val_loss: 2865.9
849
Epoch 55/100
13/13 [==============================] - 0s 4ms/step - loss: 3441.3171 - val_loss: 2644.4
744
Epoch 56/100
13/13 [==============================] - 0s 6ms/step - loss: 3250.7349 - val_loss: 2695.7
710
Epoch 57/100
13/13 [==============================] - 0s 6ms/step - loss: 3269.8787 - val_loss: 2810.7
300
Epoch 58/100
13/13 [==============================] - 0s 6ms/step - loss: 3337.4016 - val_loss: 2607.9
722
Epoch 59/100
13/13 [==============================] - 0s 6ms/step - loss: 3245.2605 - val_loss: 2522.1
880
Epoch 60/100
13/13 [==============================] - 0s 5ms/step - loss: 3372.3621 - val_loss: 2832.9
563
Epoch 61/100
13/13 [==============================] - 0s 5ms/step - loss: 3448.5593 - val_loss: 2490.8
892
Epoch 62/100
13/13 [==============================] - 0s 5ms/step - loss: 3176.4585 - val_loss: 2468.7
019
Epoch 63/100
13/13 [==============================] - 0s 4ms/step - loss: 3307.0757 - val_loss: 2480.3
381
Epoch 64/100
13/13 [==============================] - 0s 5ms/step - loss: 3229.1582 - val_loss: 2695.0
244
Epoch 65/100
13/13 [==============================] - 0s 4ms/step - loss: 3566.0994 - val_loss: 2865.9
805
Epoch 66/100
13/13 [==============================] - 0s 6ms/step - loss: 3532.8364 - val_loss: 2751.1
201
Epoch 67/100
13/13 [==============================] - 0s 3ms/step - loss: 3524.8904 - val_loss: 2578.6
426
Epoch 68/100
13/13 [==============================] - 0s 4ms/step - loss: 3849.0762 - val_loss: 2489.5
798
Epoch 69/100
13/13 [==============================] - 0s 4ms/step - loss: 3431.2085 - val_loss: 2662.2
837
Epoch 70/100
13/13 [==============================] - 0s 5ms/step - loss: 3883.3059 - val_loss: 2463.5
774
Epoch 71/100
13/13 [==============================] - 0s 3ms/step - loss: 3604.3242 - val_loss: 2979.6
169
Epoch 72/100
13/13 [==============================] - 0s 3ms/step - loss: 3741.5940 - val_loss: 2482.8
828
```

```
Epoch 73/100
13/13 [==============================] - 0s 4ms/step - loss: 3633.8206 - val_loss: 2519.5
913
Epoch 74/100
13/13 [==============================] - 0s 5ms/step - loss: 3487.7961 - val_loss: 2877.7
751
Epoch 75/100
13/13 [==============================] - 0s 5ms/step - loss: 3827.7971 - val_loss: 4162.6
948
Epoch 76/100
13/13 [==============================] - 0s 5ms/step - loss: 4002.3401 - val_loss: 2448.3
921
Epoch 77/100
13/13 [==============================] - 0s 4ms/step - loss: 3310.6270 - val_loss: 2405.1
548
Epoch 78/100
13/13 [==============================] - 0s 4ms/step - loss: 3102.6477 - val_loss: 2651.1
086
Epoch 79/100
13/13 [==============================] - 0s 4ms/step - loss: 3465.4536 - val_loss: 2720.2
617
Epoch 80/100
13/13 [==============================] - 0s 5ms/step - loss: 3174.8218 - val_loss: 2488.4
612
Epoch 81/100
13/13 [==============================] - 0s 4ms/step - loss: 3261.4280 - val_loss: 3013.2
861
Epoch 82/100
13/13 [==============================] - 0s 3ms/step - loss: 3325.5125 - val_loss: 2466.0
176
Epoch 83/100
13/13 [==============================] - 0s 4ms/step - loss: 3051.4414 - val_loss: 2393.5
234
Epoch 84/100
13/13 [==============================] - 0s 4ms/step - loss: 3051.5706 - val_loss: 2537.3
347
Epoch 85/100
13/13 [==============================] - 0s 5ms/step - loss: 3343.4368 - val_loss: 2408.0
598
Epoch 86/100
13/13 [==============================] - 0s 3ms/step - loss: 3120.0254 - val_loss: 2486.7
454
Epoch 87/100
13/13 [==============================] - 0s 5ms/step - loss: 3373.7480 - val_loss: 2360.3
062
Epoch 88/100
13/13 [==============================] - 0s 4ms/step - loss: 3258.4258 - val_loss: 2355.4
253
Epoch 89/100
13/13 [==============================] - 0s 4ms/step - loss: 3246.7439 - val_loss: 2361.2
554
Epoch 90/100
13/13 [==============================] - 0s 5ms/step - loss: 3412.7014 - val_loss: 2560.8
977
Epoch 91/100
13/13 [==============================] - 0s 3ms/step - loss: 3194.9314 - val_loss: 3208.7
832
Epoch 92/100
13/13 [==============================] - 0s 5ms/step - loss: 3666.6191 - val_loss: 3495.2
871
Epoch 93/100
13/13 [==============================] - 0s 5ms/step - loss: 3523.7188 - val_loss: 2504.3
440
Epoch 94/100
13/13 [==============================] - 0s 3ms/step - loss: 3415.9880 - val_loss: 2487.1
514
Epoch 95/100
13/13 [==============================] - 0s 5ms/step - loss: 3008.7063 - val_loss: 2584.5
432
Epoch 96/100
13/13 [==============================] - 0s 3ms/step - loss: 3266.6077 - val_loss: 2322.6
196
```

```
Epoch 97/100
13/13 [==============================] - 0s 4ms/step - loss: 3298.0808 - val_loss: 3409.8
081
Epoch 98/100
13/13 [==============================] - 0s 4ms/step - loss: 3779.3186 - val_loss: 2492.8
074
Epoch 99/100
13/13 [==============================] - 0s 4ms/step - loss: 2976.4343 - val_loss: 2308.1
936
Epoch 100/100
13/13 [==============================] - 0s 5ms/step - loss: 3078.4272 - val_loss: 2451.1
138
```

In [15]:

```
y_pred = model.predict(X)
y_pred = [x[0] for x in y_pred]
print(f'MSE = {np.sum((y-y_pred)**2) / len(y_pred)}')
```

```
16/16 [==============================] - 0s 1ms/step
MSE = 2947.633663366337
```

Как видно, модель обучилась до какого-то предела (локального минимума?).

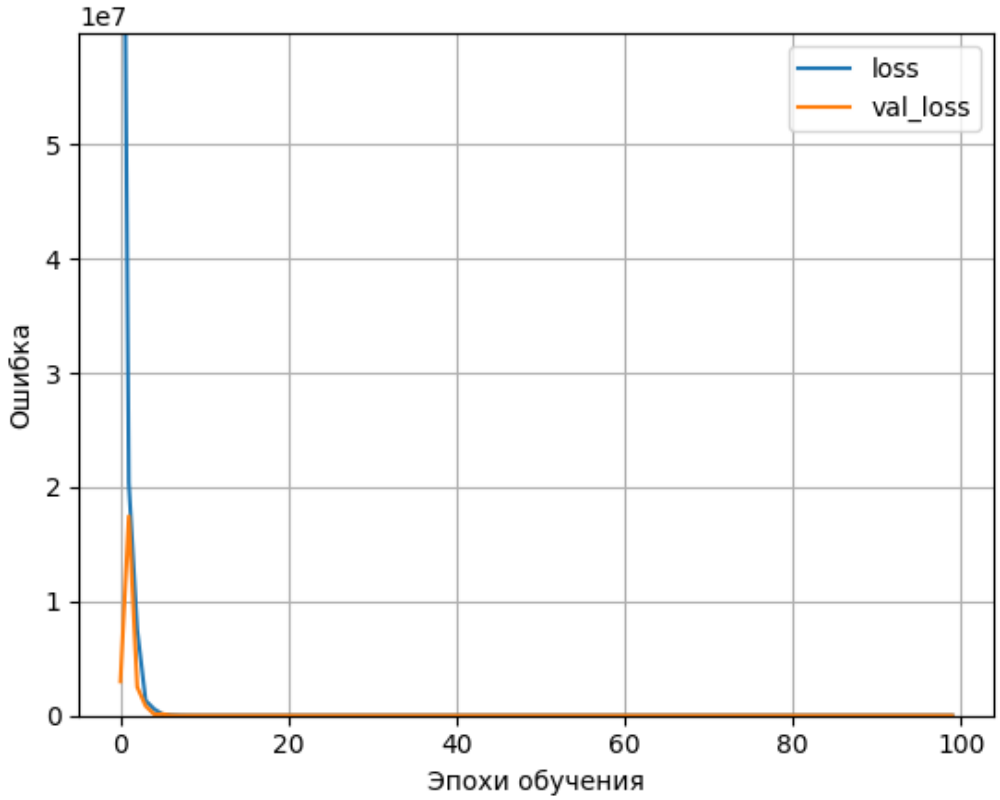
1. Постройте кривые обучения с зависимостью от количества эпох.

In [20]:

```
def plot_loss(history):
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.ylim([0, max(history.history['loss'])*0.5])
  plt.xlabel('Эпохи обучения')
  plt.ylabel('Ошибка')
  plt.legend()
  plt.grid(True)
```

In [21]:

```
plot_loss(history0)
```



1. Визуализируйте точки набора данных на плоскости в виде диаграммы рассеяния (ось **X** – независимый

признак, ось **Y –** зависимый признак**),** а также линию регрессии **(**другим цветом**),** подписывая оси и рисунок**.**

In [23]:

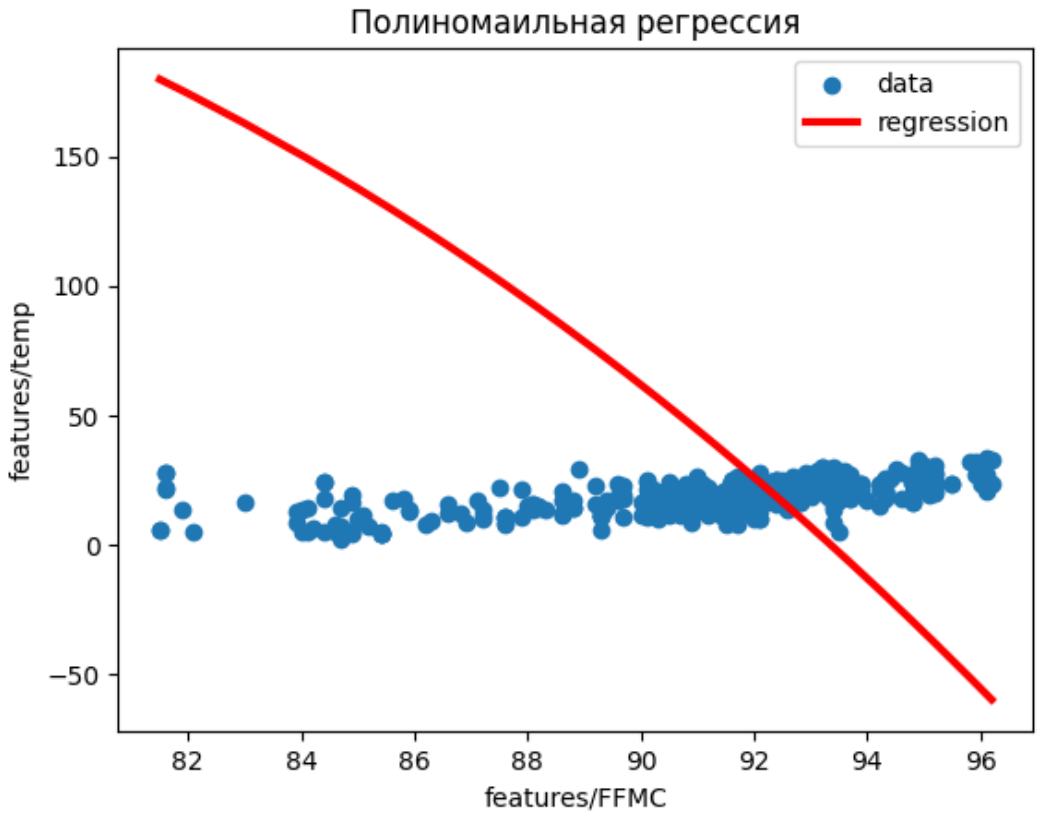
```
x = np.array(x)
y_pred = np.array(y_pred)
```

In [25]:

```
plt.scatter(x, y, label='data')
plt.plot(np.sort(x), y_pred[np.argsort(x)], c='r', lw=3, label='regression');
plt.xlabel('features/FFMC')
plt.ylabel('features/temp')
plt.title('Полиномаильная регрессия')
plt.legend()
```

Out[25]:

```
<matplotlib.legend.Legend at 0x7fea14f24280>
```



ну**,** тут уж как поулчилось**...**

1. Определите в исходном наборе данных признак **(**отличный от независимого и зависимого признаков**),** принимающий непрерывные значения и имеющий свойства**,** указанные в индивидуальном задании**.** Доп**.** признак**:** имеющий минимальную корреляцию с независимой переменной**.**

   area: 0.040062

In [26]:

```
# оставим непрерывные
df_new = df.drop(['features/X', 'features/Y', 'features/day', 'features/month'], axis=1)
```

In [27]:

```
np.abs(df_new.corrwith(df['features/FFMC']))
```

Out[27]:

```
area            0.040062
features/DC     0.443239
```

```
features/DMC     0.490669
features/FFMC    1.000000
features/ISI     0.669626
features/RH      0.252776
features/rain    0.095590
features/temp    0.594506
features/wind    0.096956
dtype: float64
```

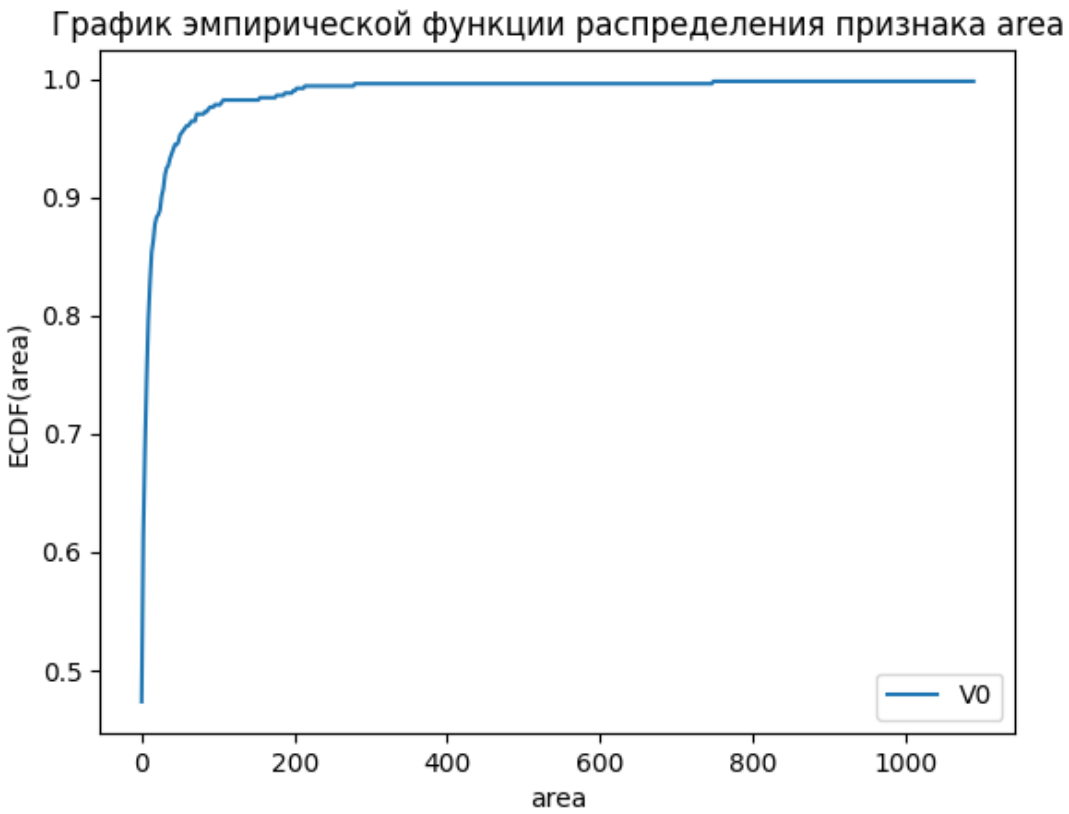
1. Визуализируйте этот признак в соответствии с индивидуальным заданием. Визуализация доп. признака – эмпирическая плотность распределения

In [28]:

```
def ECDF(data, x):
    counter = 0
    for v in data:
        if v <= x:
            counter += 1
    return counter / len(data)

samples = df['area'] # sepal length
npoints = 500
dx = (samples.max()-samples.min())/npoints

xlist = [samples.min()+dx*i for i in range(npoints)]
ylist = [ECDF(samples, x) for x in xlist]

df_ECDF = pd.DataFrame(ylist, columns=['V0'],index=xlist)
df_ECDF.plot.line(title='График эмпирической функции распределения признака area', xlabel
='area', ylabel='ECDF(area)')
```

Out[28]:

```
<Axes: title={'center': 'График эмпирической функции распределения признака area'}, xlabe
l='area', ylabel='ECDF(area)'>
```



1. Сформируйте набор входных данных из двух признаков набора данных (независимый признак и определенный признак), создайте и адаптируйте нормализующий слой **Tensorflow** для двух признаков.

In [29]:

```
X = df[['features/FFMC', 'area']]
X.head()
```

Out[29]:

	features/FFMC	area
0	**96.099998**	**10.820000**
1	**90.500000**	**24.590000**
2	**94.300003**	**0.170000**
3	**96.099998**	**14.680000**
4	**92.900002**	**88.489998**

In [30]:

```
df_normalizer = tf.keras.layers.Normalization()
df_normalizer.adapt(X)
print(df_normalizer.mean.numpy())
print(df_normalizer.variance.numpy())
```

```
[[91.26317  13.101703]]
[[   8.010956 4137.2344  ]]
```

1. Используя созданный нормализующий слой, постройте нейронную сеть (нелинейный регресор) с количеством скрытых слоев, количеством нейронов и функцией активации, указанными в индивидуальном задании, и одним нейроном в выходном слое и обучите ее на наборе данных из двух признаков и отклика.

   Параметры глубокой нейронной сети: кол-во скрытых слоев **– 3,** кол-во нейронов в скрытом слое **– 128,** функция активации **– relu.**

In [31]:

```
model = Sequential([
    df_normalizer,
    Dense(units=128, activation='relu'),
    Dense(units=128, activation='relu'),
    Dense(units=128, activation='relu'),
    Dense(units=1, activation='linear')
])
```

In [32]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 2)                5
 n)

 dense_2 (Dense)             (None, 128)               384

 dense_3 (Dense)             (None, 128)               16512

 dense_4 (Dense)             (None, 128)               16512

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 33,542
Trainable params: 33,537
Non-trainable params: 5
_________________________________________________________________
```

In [33]:

```
model.compile(loss='mse')
history = model.fit(
    X, y,
    epochs=100,
    # уровень выводимой информации
    verbose=1,
    # проверка (валидация) на 20% обучающих данных
    validation_split = 0.2)
```

```
Epoch 1/100
13/13 [==============================] - 1s 17ms/step - loss: 334.4527 - val_loss: 239.16
41
Epoch 2/100
13/13 [==============================] - 0s 5ms/step - loss: 147.4668 - val_loss: 223.067
7
Epoch 3/100
13/13 [==============================] - 0s 5ms/step - loss: 57.0400 - val_loss: 148.0962
Epoch 4/100
13/13 [==============================] - 0s 6ms/step - loss: 40.2599 - val_loss: 151.7744
Epoch 5/100
13/13 [==============================] - 0s 6ms/step - loss: 31.9360 - val_loss: 126.1093
Epoch 6/100
13/13 [==============================] - 0s 5ms/step - loss: 27.2951 - val_loss: 90.8178
Epoch 7/100
13/13 [==============================] - 0s 5ms/step - loss: 24.0217 - val_loss: 81.8294
Epoch 8/100
13/13 [==============================] - 0s 5ms/step - loss: 22.9604 - val_loss: 74.8228
Epoch 9/100
13/13 [==============================] - 0s 7ms/step - loss: 22.1009 - val_loss: 76.2802
Epoch 10/100
13/13 [==============================] - 0s 5ms/step - loss: 22.2750 - val_loss: 76.9588
Epoch 11/100
13/13 [==============================] - 0s 6ms/step - loss: 21.4737 - val_loss: 66.4640
Epoch 12/100
13/13 [==============================] - 0s 4ms/step - loss: 21.5010 - val_loss: 67.5268
Epoch 13/100
13/13 [==============================] - 0s 5ms/step - loss: 21.3040 - val_loss: 85.3022
Epoch 14/100
13/13 [==============================] - 0s 4ms/step - loss: 20.7810 - val_loss: 81.1236
Epoch 15/100
13/13 [==============================] - 0s 6ms/step - loss: 21.5006 - val_loss: 85.7689
Epoch 16/100
13/13 [==============================] - 0s 4ms/step - loss: 21.0245 - val_loss: 50.7942
Epoch 17/100
13/13 [==============================] - 0s 6ms/step - loss: 20.3426 - val_loss: 65.7635
Epoch 18/100
13/13 [==============================] - 0s 4ms/step - loss: 20.3105 - val_loss: 76.5151
Epoch 19/100
13/13 [==============================] - 0s 5ms/step - loss: 20.8641 - val_loss: 53.0607
Epoch 20/100
13/13 [==============================] - 0s 6ms/step - loss: 19.9272 - val_loss: 82.8452
Epoch 21/100
13/13 [==============================] - 0s 4ms/step - loss: 20.3985 - val_loss: 65.9838
Epoch 22/100
13/13 [==============================] - 0s 5ms/step - loss: 20.1984 - val_loss: 47.3065
Epoch 23/100
13/13 [==============================] - 0s 7ms/step - loss: 20.4412 - val_loss: 41.2476
Epoch 24/100
13/13 [==============================] - 0s 5ms/step - loss: 20.5829 - val_loss: 66.8237
Epoch 25/100
13/13 [==============================] - 0s 5ms/step - loss: 20.6466 - val_loss: 50.9244
Epoch 26/100
13/13 [==============================] - 0s 4ms/step - loss: 20.2902 - val_loss: 70.2206
Epoch 27/100
13/13 [==============================] - 0s 6ms/step - loss: 20.6389 - val_loss: 56.8671
Epoch 28/100
13/13 [==============================] - 0s 4ms/step - loss: 19.9974 - val_loss: 71.4359
Epoch 29/100
13/13 [==============================] - 0s 4ms/step - loss: 20.4831 - val_loss: 58.2693
Epoch 30/100
13/13 [==============================] - 0s 4ms/step - loss: 20.3071 - val_loss: 43.2517
Epoch 31/100
```

```
13/13 [==============================] - 0s 4ms/step - loss: 19.7657 - val_loss: 59.6811
Epoch 32/100
13/13 [==============================] - 0s 5ms/step - loss: 20.8339 - val_loss: 39.6871
Epoch 33/100
13/13 [==============================] - 0s 4ms/step - loss: 19.8012 - val_loss: 45.0088
Epoch 34/100
13/13 [==============================] - 0s 5ms/step - loss: 20.3039 - val_loss: 50.4676
Epoch 35/100
13/13 [==============================] - 0s 6ms/step - loss: 20.1164 - val_loss: 48.2886
Epoch 36/100
13/13 [==============================] - 0s 7ms/step - loss: 19.8722 - val_loss: 57.8296
Epoch 37/100
13/13 [==============================] - 0s 6ms/step - loss: 20.8148 - val_loss: 36.4553
Epoch 38/100
13/13 [==============================] - 0s 4ms/step - loss: 20.1579 - val_loss: 47.1891
Epoch 39/100
13/13 [==============================] - 0s 5ms/step - loss: 20.0379 - val_loss: 63.0229
Epoch 40/100
13/13 [==============================] - 0s 6ms/step - loss: 20.4783 - val_loss: 48.2867
Epoch 41/100
13/13 [==============================] - 0s 7ms/step - loss: 19.9503 - val_loss: 42.9794
Epoch 42/100
13/13 [==============================] - 0s 5ms/step - loss: 20.2053 - val_loss: 52.6338
Epoch 43/100
13/13 [==============================] - 0s 6ms/step - loss: 19.5246 - val_loss: 50.2342
Epoch 44/100
13/13 [==============================] - 0s 8ms/step - loss: 20.3800 - val_loss: 41.3574
Epoch 45/100
13/13 [==============================] - 0s 5ms/step - loss: 20.2559 - val_loss: 46.5726
Epoch 46/100
13/13 [==============================] - 0s 5ms/step - loss: 20.2743 - val_loss: 34.7265
Epoch 47/100
13/13 [==============================] - 0s 6ms/step - loss: 20.0194 - val_loss: 40.5808
Epoch 48/100
13/13 [==============================] - 0s 5ms/step - loss: 20.1670 - val_loss: 49.8863
Epoch 49/100
13/13 [==============================] - 0s 6ms/step - loss: 20.4859 - val_loss: 39.3477
Epoch 50/100
13/13 [==============================] - 0s 6ms/step - loss: 20.3290 - val_loss: 49.0058
Epoch 51/100
13/13 [==============================] - 0s 6ms/step - loss: 20.1374 - val_loss: 43.7762
Epoch 52/100
13/13 [==============================] - 0s 6ms/step - loss: 19.9993 - val_loss: 39.7544
Epoch 53/100
13/13 [==============================] - 0s 5ms/step - loss: 19.8305 - val_loss: 51.8732
Epoch 54/100
13/13 [==============================] - 0s 5ms/step - loss: 19.8408 - val_loss: 43.0935
Epoch 55/100
13/13 [==============================] - 0s 4ms/step - loss: 19.6739 - val_loss: 48.8845
Epoch 56/100
13/13 [==============================] - 0s 5ms/step - loss: 20.1427 - val_loss: 32.0438
Epoch 57/100
13/13 [==============================] - 0s 5ms/step - loss: 20.0137 - val_loss: 37.5315
Epoch 58/100
13/13 [==============================] - 0s 5ms/step - loss: 19.6730 - val_loss: 42.8393
Epoch 59/100
13/13 [==============================] - 0s 6ms/step - loss: 19.5162 - val_loss: 36.9753
Epoch 60/100
13/13 [==============================] - 0s 5ms/step - loss: 19.6242 - val_loss: 37.9165
Epoch 61/100
13/13 [==============================] - 0s 6ms/step - loss: 20.2260 - val_loss: 40.1858
Epoch 62/100
13/13 [==============================] - 0s 6ms/step - loss: 19.8022 - val_loss: 33.7392
Epoch 63/100
13/13 [==============================] - 0s 6ms/step - loss: 20.4679 - val_loss: 33.8564
Epoch 64/100
13/13 [==============================] - 0s 6ms/step - loss: 20.1035 - val_loss: 38.6225
Epoch 65/100
13/13 [==============================] - 0s 5ms/step - loss: 19.9717 - val_loss: 33.6067
Epoch 66/100
13/13 [==============================] - 0s 6ms/step - loss: 19.6648 - val_loss: 38.0719
Epoch 67/100
```

```
13/13 [==============================] - 0s 6ms/step - loss: 20.2445 - val_loss: 49.8462
Epoch 68/100
13/13 [==============================] - 0s 5ms/step - loss: 19.3933 - val_loss: 56.5650
Epoch 69/100
13/13 [==============================] - 0s 6ms/step - loss: 20.1561 - val_loss: 33.8759
Epoch 70/100
13/13 [==============================] - 0s 5ms/step - loss: 19.8733 - val_loss: 34.6892
Epoch 71/100
13/13 [==============================] - 0s 10ms/step - loss: 19.5274 - val_loss: 40.5808
Epoch 72/100
13/13 [==============================] - 0s 6ms/step - loss: 19.3771 - val_loss: 45.7072
Epoch 73/100
13/13 [==============================] - 0s 8ms/step - loss: 19.4411 - val_loss: 37.6523
Epoch 74/100
13/13 [==============================] - 0s 10ms/step - loss: 19.6195 - val_loss: 34.0401
Epoch 75/100
13/13 [==============================] - 0s 8ms/step - loss: 19.7160 - val_loss: 38.7936
Epoch 76/100
13/13 [==============================] - 0s 6ms/step - loss: 19.6977 - val_loss: 63.3216
Epoch 77/100
13/13 [==============================] - 0s 6ms/step - loss: 20.5602 - val_loss: 35.3805
Epoch 78/100
13/13 [==============================] - 0s 7ms/step - loss: 19.7305 - val_loss: 30.1655
Epoch 79/100
13/13 [==============================] - 0s 6ms/step - loss: 19.3109 - val_loss: 48.8983
Epoch 80/100
13/13 [==============================] - 0s 7ms/step - loss: 19.6168 - val_loss: 51.7130
Epoch 81/100
13/13 [==============================] - 0s 7ms/step - loss: 19.9135 - val_loss: 36.7082
Epoch 82/100
13/13 [==============================] - 0s 7ms/step - loss: 19.4505 - val_loss: 36.3852
Epoch 83/100
13/13 [==============================] - 0s 9ms/step - loss: 19.1536 - val_loss: 50.7242
Epoch 84/100
13/13 [==============================] - 0s 7ms/step - loss: 19.7649 - val_loss: 41.2187
Epoch 85/100
13/13 [==============================] - 0s 11ms/step - loss: 19.5971 - val_loss: 39.0127
Epoch 86/100
13/13 [==============================] - 0s 7ms/step - loss: 19.0907 - val_loss: 36.6324
Epoch 87/100
13/13 [==============================] - 0s 7ms/step - loss: 20.1808 - val_loss: 30.1883
Epoch 88/100
13/13 [==============================] - 0s 9ms/step - loss: 19.5275 - val_loss: 40.7477
Epoch 89/100
13/13 [==============================] - 0s 9ms/step - loss: 19.2685 - val_loss: 33.9546
Epoch 90/100
13/13 [==============================] - 0s 7ms/step - loss: 19.1867 - val_loss: 29.4800
Epoch 91/100
13/13 [==============================] - 0s 6ms/step - loss: 19.5212 - val_loss: 31.0856
Epoch 92/100
13/13 [==============================] - 0s 5ms/step - loss: 19.5476 - val_loss: 42.3579
Epoch 93/100
13/13 [==============================] - 0s 7ms/step - loss: 19.2822 - val_loss: 47.8922
Epoch 94/100
13/13 [==============================] - 0s 6ms/step - loss: 19.2308 - val_loss: 32.4583
Epoch 95/100
13/13 [==============================] - 0s 7ms/step - loss: 18.6375 - val_loss: 33.0452
Epoch 96/100
13/13 [==============================] - 0s 8ms/step - loss: 19.5013 - val_loss: 48.6390
Epoch 97/100
13/13 [==============================] - 0s 6ms/step - loss: 19.8242 - val_loss: 42.3702
Epoch 98/100
13/13 [==============================] - 0s 7ms/step - loss: 19.5932 - val_loss: 30.2149
Epoch 99/100
13/13 [==============================] - 0s 5ms/step - loss: 19.5541 - val_loss: 35.8491
Epoch 100/100
13/13 [==============================] - 0s 6ms/step - loss: 19.1938 - val_loss: 37.2671
```

Качество заметно лучше, чем в первом варианте

1. Визуализируйте набор данных в виде точечного графика и прогноз нейронной сети в виде поверхности в

1. Визуализируйте набор данных в виде точечного графика и прогноз нейронной сети в виде поверхности в трехмерном пространстве.

In [34]:

```
y_pred = model.predict(X)
```

```
16/16 [==============================] - 0s 2ms/step
```

In [35]:

```
y_pred = [x[0] for x in y_pred]
```

In [36]:

```
X = np.array(X)
X
```

Out[36]:

```
array([[96.1 , 10.82],
       [90.5 , 24.59],
       [94.3 ,  0.17],
       ...,
       [90.  ,  0.  ],
       [89.4 ,  0.  ],
       [91.6 , 42.87]], dtype=float32)
```

In [40]:

```
n_plot = 51

x_plot = np.linspace(np.min(xs), np.max(xs), n_plot)
y_plot = np.linspace(np.min(ys), np.max(ys), n_plot)

x_plot, y_plot = np.meshgrid(x_plot, y_plot)
```

In [41]:

```
x_plot2 = np.reshape(x_plot, [n_plot**2,1])
y_plot2 = np.reshape(y_plot, [n_plot**2,1])
xy_2 = np.hstack([x_plot2, y_plot2])
xy_2.shape
```

Out[41]:

```
(2601, 2)
```

In [43]:

```
z = model.predict(xy_2)
z.shape
```

```
82/82 [==============================] - 0s 1ms/step
```

Out[43]:

```
(2601, 1)
```

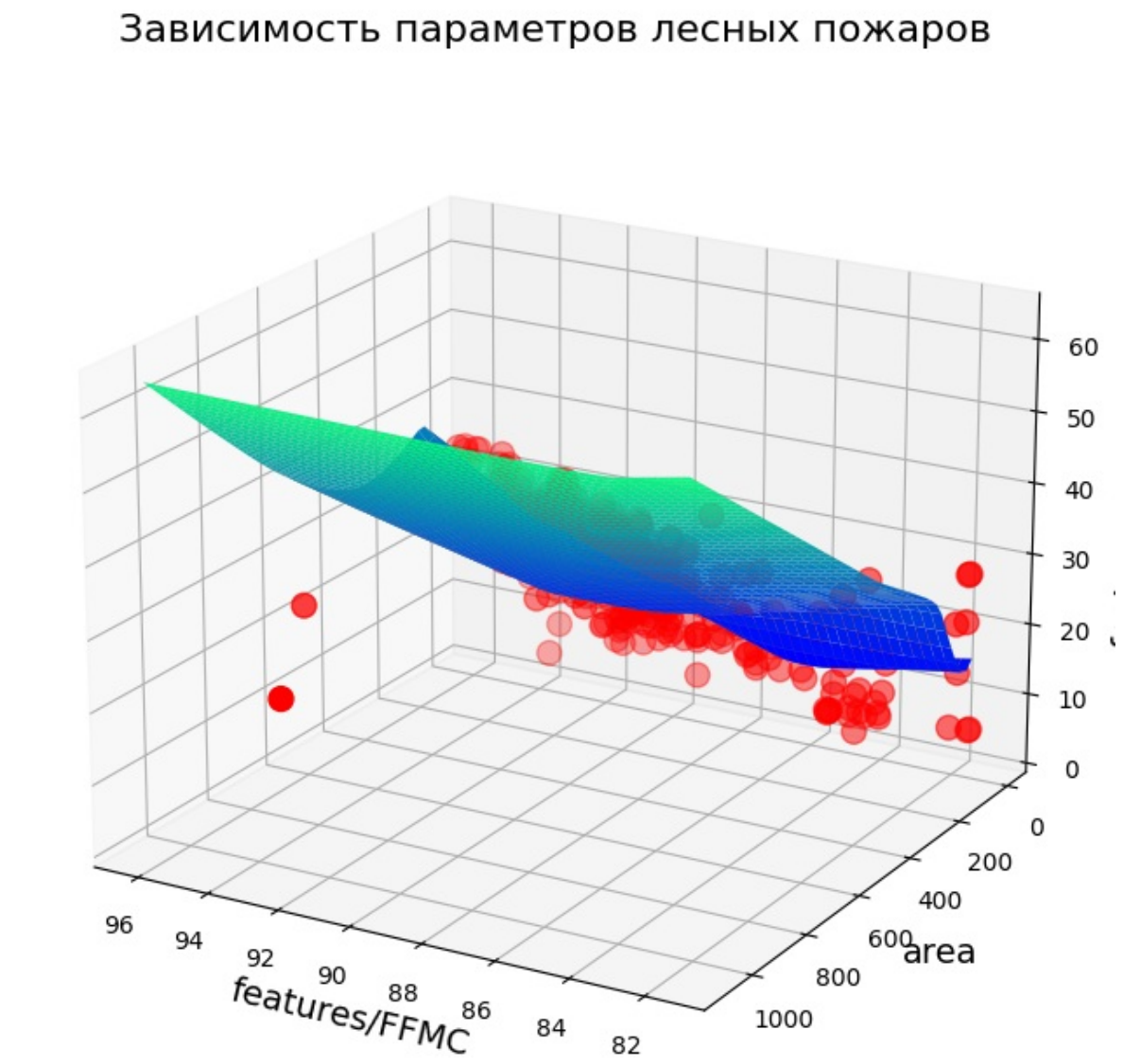
In [46]:

```
z_plot = z.reshape((n_plot, n_plot))
```

In [52]:

```
from matplotlib import cm

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(x_plot, y_plot, z_plot, \
       rstride=1, cstride=1, linewidth=0.05, cmap=cm.winter, antialiased=True, \
```

```
       edgecolors='gray')
ax.scatter( xs, ys, zs, s=100, c='r' )

ax.set_xlabel('features/FFMC', fontsize=14)
ax.set_ylabel('area', fontsize=14)
ax.set_zlabel('features/temp', fontsize=14)
ax.set_title('Зависимость параметров лесных пожаров', fontsize=16)

ax.set_zlim(0., z_plot.max())
ax.view_init(elev = 20, azim = 120)
```



1. Разбейте набор данных из двух признаков и отклика на обучающую и тестовую выборки и постройте кривые обучения для заданного показателя качества в зависимости от количества точек в обучающей выборке, подписывая оси и рисунок и создавая легенду.

   Показатель качества регрессии **– MSE (mean squared error)**

In [91]:

```
def data_split(ratio=20):
  train, test = tfds.load("forest_fires", split= [f'train[:{ratio}%]', f'train[{ratio}%:
]'])
  train = tfds.as_dataframe(train)
  test = tfds.as_dataframe(test)
  y_train = train['features/temp']
  y_test = test['features/temp']
  X_train = train[['features/FFMC', 'area']]
  X_test = test[['features/FFMC', 'area']]
  return X_train, y_train, X_test, y_test
```

In [92]:

```
def mse(y, y_):
  return np.sum((y - y_)**2) / y.shape[0]
```

In [101]:

```
train_score = []
test_score = []
for i in range(10, 100, 10):
  X_train, y_train, X_test, y_test = data_split(i)
  print(X_train.shape, y_train.shape)
  model_ = Sequential([
      df_normalizer,
      Dense(units=128, activation='relu'),
      # Dense(units=128, activation='relu'),
      # Dense(units=128, activation='relu'),
      Dense(units=1, activation='linear')
  ])
  model_.compile(loss='mse')

  model_.fit(X_train, y_train, epochs=50, verbose=0)

  y_train_predict = model_.predict(X_train)
  y_train_predict = [x[0] for x in y_train_predict]
  train_score.append(mse(y_train, y_train_predict))

  y_test_predict = model_.predict(X_test)
  y_test_predict = [x[0] for x in y_test_predict]
  test_score.append(mse(y_test, y_test_predict))
  print('-->', i, ' done')
```

```
(52, 2) (52,)
2/2 [==============================] - 0s 10ms/step
15/15 [==============================] - 0s 2ms/step
--> 10  done
(103, 2) (103,)
4/4 [==============================] - 0s 4ms/step
13/13 [==============================] - 0s 1ms/step
--> 20  done
(155, 2) (155,)
5/5 [==============================] - 0s 2ms/step
12/12 [==============================] - 0s 2ms/step
--> 30  done
(207, 2) (207,)
7/7 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
--> 40  done
(258, 2) (258,)
9/9 [==============================] - 0s 2ms/step
9/9 [==============================] - 0s 2ms/step
--> 50  done
(310, 2) (310,)
10/10 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
--> 60  done
(362, 2) (362,)
12/12 [==============================] - 0s 1ms/step
5/5 [==============================] - 0s 2ms/step
--> 70  done
(414, 2) (414,)
13/13 [==============================] - 0s 1ms/step
4/4 [==============================] - 0s 4ms/step
--> 80  done
(465, 2) (465,)
15/15 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 8ms/step
--> 90  done
```

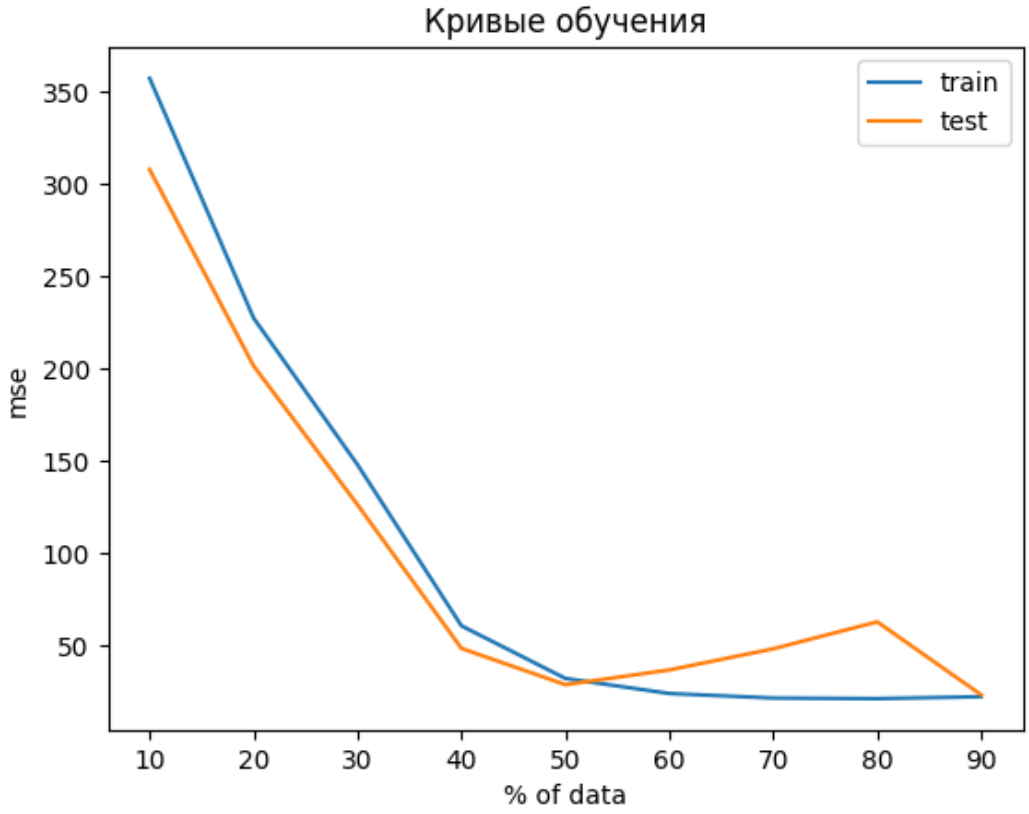
In [104]:

```
plt.plot(
```

```
plt.plot(
    range(10, 100, 10),
    train_score, label='train'
)
plt.plot(
    range(10, 100, 10),
    test_score, label='test'
)
plt.xlabel('% of data')
plt.ylabel('mse')
plt.title('Кривые обучения')
plt.legend()
```

Out[104]:

```
<matplotlib.legend.Legend at 0x7fea0b804ee0>
```



In [ ]:

In [ ]: