

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра математического моделирования и искусственного интеллекта

## ОТЧЕТ ПО КОНТРОЛЬНОЙ РАБОТЕ № 4

Дисциплина: Методы машинного обучения

Студент: Петров Артем Евгеньевич

Группа: НКНбд-01-21

Москва 2024

---

### Вариант №18

1. Загрузите заданный в индивидуальном задании набор данных из Tensorflow Datasets, включая указанные в задании независимые признаки и метку класса. Оставьте в наборе признаки, принимающие числовые значения.
2. Визуализируйте точки набора данных на плоскости с координатами, соответствующими двум независимым признакам, отображая точки различных классов разными цветами. Подпишите оси и рисунок, создайте легенду для классов набора данных.
3. Если признак с метками классов содержит более двух классов, то объедините некоторые классы, чтобы получить набор для бинарной классификации. Объединяйте классы таким образом, чтобы положительный и отрицательный классы были сопоставимы по количеству точек.
4. Разбейте набор данных из двух признаков и бинарных меток класса на обучающую и тестовую выборки. Постройте нейронные сети с нормализующим слоем и параметрами, указанными в индивидуальном задании, для бинарной классификации и обучите их на обучающей выборке, контролируя процесс обучения нейронных сетей. Определите нейронную сеть с более высоким качеством бинарной классификации по показателю бинарной классификации, указанному в индивидуальном задании.
5. Визуализируйте границы принятия решений построенных нейронных сетей на отдельных рисунках на всем наборе данных из двух признаков и бинарных меток

классов.

6. Визуализируйте на одном рисунке ROC-кривые для построенных классификаторов на основе нейронных сетей, вычислите площади под ROC-кривыми методом трапеций или иным методом и создайте легенду с указанием площадей кривых.
7. Определите исходном наборе данных дополнительный признак, отличный от указанных в задании двух независимых признаков, принимающий непрерывные значения и имеющий максимальную дисперсию.
8. Визуализируйте точки набора данных в трехмерном пространстве с координатами, соответствующими трем независимым признакам, отображая точки различных классов разными цветами. Подпишите оси и рисунок, создайте легенду для классов набора данных.
9. Разбейте исходный набор данных на обучающую и тестовую выборки. Постройте нейронную сеть для многоклассовой классификации с нормализующим слоем и параметрами, соответствующими лучшей нейронной сети для бинарной классификации из п.4, и обучите ее на обучающей выборке, контролируя процесс ее обучения.
10. Постройте кривые обучения в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду.

#### Контрольная работа 4 – Вариант 18

1. Набор данных: diamonds
2. Независимые признаки: carat, depth
3. Метка класса: cut
4. Показатель бинарной классификации:

*FN* (False Negatives) – число точек в положительном классе, ошибочно спрогнозированных в отрицательный класс

5. Параметры нейронных сетей: Кол-во скрытых слоев: 3 Кол-во нейронов в слое: 128  
Оптимизатор: Adadelta Функции активации в скрытых слоях: leaky\_relu, selu, swish

**1. Загрузите заданный в индивидуальном задании набор данных из Tensorflow Datasets, включая указанные в задании независимые признаки и метку класса. Оставьте в наборе признаки, принимающие числовые значения.**

Набор данных: diamonds

Независимые признаки: carat, depth

Метка класса: cut

```
In [44]: import pandas as pd
```

```
In [45]: df = pd.read_csv('https://drive.google.com/uc?export=download&id=1Mu4s1dgU_xG1DA1JI')
```

```
In [46]: df.head()
```

```
Out[46]:
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
In [47]: df.isna().sum()
```

```
Out[47]: Unnamed: 0    0
carat          0
cut            0
color          0
clarity        0
depth          0
table          0
price          0
x              0
y              0
z              0
dtype: int64
```

```
In [48]: df.columns
```

```
Out[48]: Index(['Unnamed: 0', 'carat', 'cut', 'color', 'clarity', 'depth', 'table',
               'price', 'x', 'y', 'z'],
              dtype='object')
```

```
In [ ]: df = df.drop(['Unnamed: 0', 'color', 'clarity'], axis = 1)
```

```
In [56]: df.head()
```

```
Out[56]:
```

	carat	cut	depth	table	price	x	y	z
0	0.23	Ideal	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	63.3	58.0	335	4.34	4.35	2.75

## 2. Визуализируйте точки набора данных на плоскости с координатами, соответствующими двум независимым признакам, отображая точки различных классов разными цветами. Подпишите оси и рисунок, создайте легенду для классов набора данных.

Независимые признаки: carat, depth

Метка класса: cut

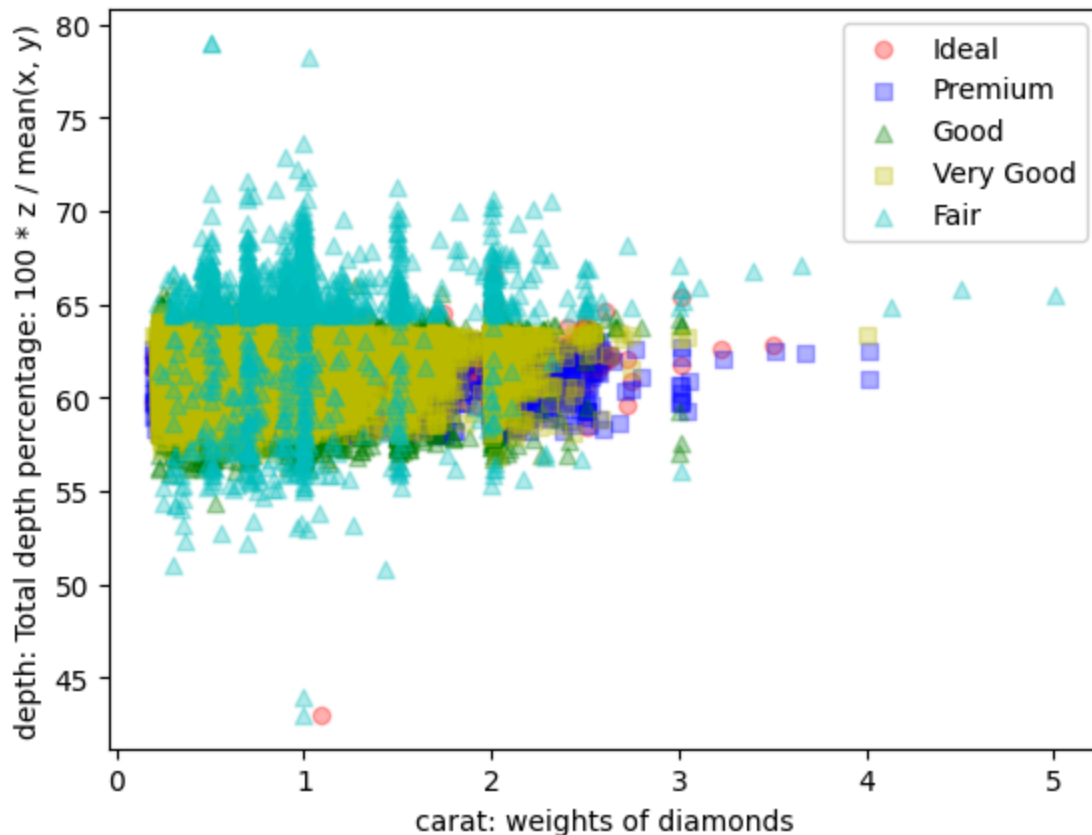
```
In [57]: df['cut'].unique()
```

```
Out[57]: array(['Ideal', 'Premium', 'Good', 'Very Good', 'Fair'], dtype=object)
```

```
In [58]: import matplotlib.pyplot as plt
```

```
In [59]: plt.scatter(df[df['cut'] == 'Ideal']['carat'], df[df['cut'] == 'Ideal']['depth'], c =
plt.scatter(df[df['cut'] == 'Premium']['carat'], df[df['cut'] == 'Premium']['depth'], c =
plt.scatter(df[df['cut'] == 'Good']['carat'], df[df['cut'] == 'Good']['depth'], c =
plt.scatter(df[df['cut'] == 'Very Good']['carat'], df[df['cut'] == 'Very Good']['depth'], c =
plt.scatter(df[df['cut'] == 'Fair']['carat'], df[df['cut'] == 'Fair']['depth'], c =
plt.xlabel("carat: weights of diamonds")
plt.ylabel("depth: Total depth percentage: 100 * z / mean(x, y)")
plt.legend()
```

```
Out[59]: <matplotlib.legend.Legend at 0x20f2a060bc0>
```



3. Если признак с метками классов содержит более двух классов, то объедините некоторые классы, чтобы получить набор для бинарной классификации. Объединяйте классы таким образом, чтобы положительный и отрицательный классы были сопоставимы по количеству точек.

```
In [60]: df.head()
```

```
Out[60]:
```

	carat	cut	depth	table	price	x	y	z
0	0.23	Ideal	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	63.3	58.0	335	4.34	4.35	2.75

```
In [62]: df['cut'].value_counts()
```

```
Out[62]: cut
Ideal      21551
Premium    13791
Very Good  12082
Good       4906
Fair       1610
Name: count, dtype: int64
```

```
In [65]: df['res'] = [1 if cut == 'Ideal' or cut == 'Premium' else 0 for cut in df['cut']]
df.head()
```

```
Out[65]:
```

	carat	cut	depth	table	price	x	y	z	res
0	0.23	Ideal	61.5	55.0	326	3.95	3.98	2.43	1
1	0.21	Premium	59.8	61.0	326	3.89	3.84	2.31	1
2	0.23	Good	56.9	65.0	327	4.05	4.07	2.31	0
3	0.29	Premium	62.4	58.0	334	4.20	4.23	2.63	1
4	0.31	Good	63.3	58.0	335	4.34	4.35	2.75	0

```
In [66]: df['res'].value_counts()
```

```
Out[66]: res
1      35342
0      18598
Name: count, dtype: int64
```

**4. Разбейте набор данных из двух признаков и бинарных меток класса на обучающую и тестовую выборки. Постройте нейронные сети с нормализующим слоем и параметрами, указанными в индивидуальном задании, для бинарной классификации и обучите их на обучающей выборке, контролируя процесс обучения нейронных сетей. Определите нейронную сеть с более высоким качеством бинарной классификации по показателю бинарной**

# классификации, указанному в индивидуальном задании.

Показатель бинарной классификации:

*FN* (False Negatives) – число точек в положительном классе, ошибочно спрогнозированных в отрицательный класс

Параметры нейронных сетей: Кол-во скрытых слоев: 3 Кол-во нейронов в слое: 128  
Оптимизатор: Adadelata Функции активации в скрытых слоях: leaky\_relu, selu, swish

```
In [99]: def train_test_split(X, y, test_ratio = 0.2, seed = None):
    assert X.shape[0] == y.shape[0], "not equal shapes of x and y"
    assert 0.0 <= test_ratio <= 1.0, "wrong test ration val"

    if seed:
        np.random.seed(seed)

    shuffled_indexes = np.random.permutation(len(X))

    test_size = int(len(X) * test_ratio)

    test_indexes = shuffled_indexes[:test_size]
    train_indexes = shuffled_indexes[test_size:]

    X_train = X.iloc[train_indexes]
    y_train = y.iloc[train_indexes]

    X_test = X.iloc[test_indexes]
    y_test = y.iloc[test_indexes]

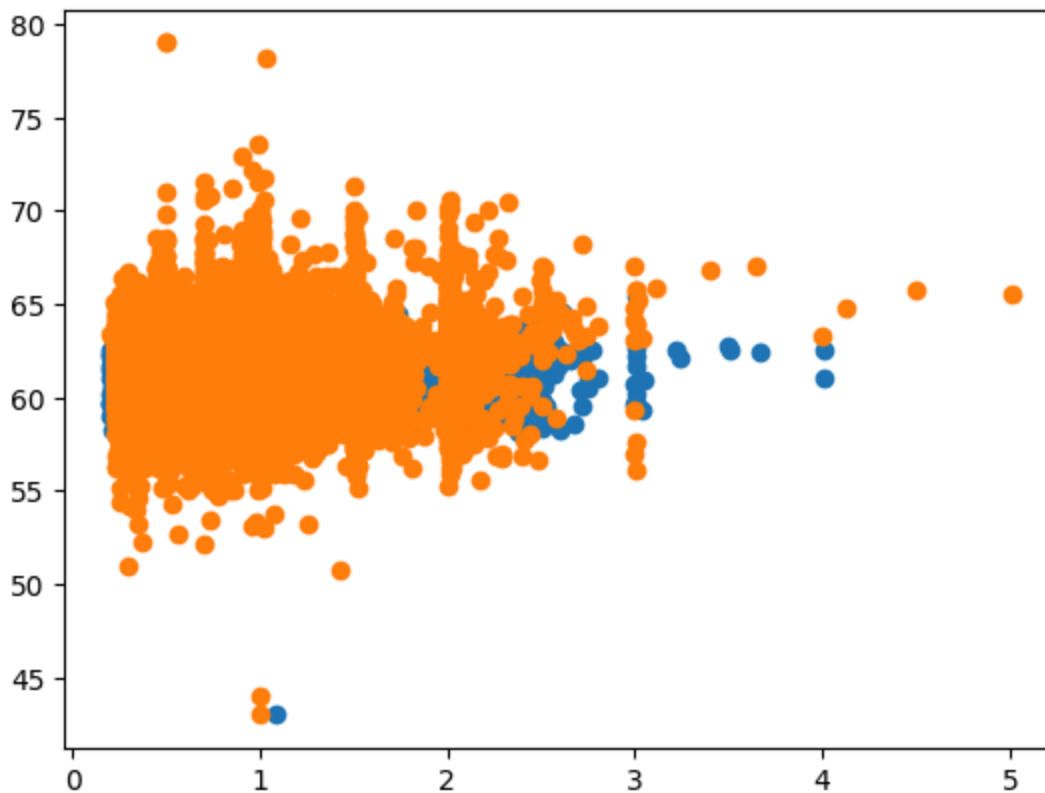
    return X_train, y_train, X_test, y_test
```

```
In [100... X = df[['carat', 'depth']]
y = df['res']
X.head(), X.shape, y.head(), y.shape, len(X)
```

```
Out[100... (   carat  depth
0   0.23   61.5
1   0.21   59.8
2   0.23   56.9
3   0.29   62.4
4   0.31   63.3,
(53940, 2),
0    1
1    1
2    0
3    1
4    0
Name: res, dtype: int64,
(53940,),
53940)
```

```
In [139... plt.scatter(df[df['res'] == 1]['carat'], df[df['res'] == 1]['depth'])
plt.scatter(df[df['res'] == 0]['carat'], df[df['res'] == 0]['depth'])
```

```
Out[139... <matplotlib.collections.PathCollection at 0x20f5c9d0200>
```



```
In [101... import numpy as np
```

```
In [102... X_train, y_train, X_test, y_test = train_test_split(X, y, test_ratio = 0.2)
```

```
In [140... # X_train.head(), y_train.head(), X_test.head(), y_test.head()
```

```
In [141... import tensorflow as tf
```

```
In [142... feature_normalizer = tf.keras.layers.Normalization(axis=None, input_shape=(X.shape[1]
feature_normalizer.adapt(X_train.values)
```

C:\Python312\Lib\site-packages\keras\src\layers\preprocessing\normalization.py:99: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(**kwargs)
```

```
In [158... def leaky_relu(x):
    return tf.keras.activations.leaky_relu(x, negative_slope=0.1)
```

```
In [159... model = tf.keras.Sequential([
    feature_normalizer,
    tf.keras.layers.Dense(128, activation = leaky_relu),
    # tf.keras.layers.ReLU(negative_slope=0.1),
```



```
tf.keras.layers.Dense(128, activation='selu'),
tf.keras.layers.Dense(128, activation='swish'),
tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

Model: "sequential\_6"

Layer (type)	Output Shape	
normalization_3 (Normalization)	(None, 2)	
dense_24 (Dense)	(None, 128)	
dense_25 (Dense)	(None, 128)	
dense_26 (Dense)	(None, 128)	
dense_27 (Dense)	(None, 1)	

Total params: 33,540 (131.02 KB)

Trainable params: 33,537 (131.00 KB)

Non-trainable params: 3 (16.00 B)

```
In [160... model.compile(loss = tf.keras.losses.binary_crossentropy,
               optimizer = tf.keras.optimizers.Adadelta(learning_rate=0.01),
               metrics=[tf.keras.metrics.FalseNegatives(name = 'FN')])
               # metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')]
               )
```

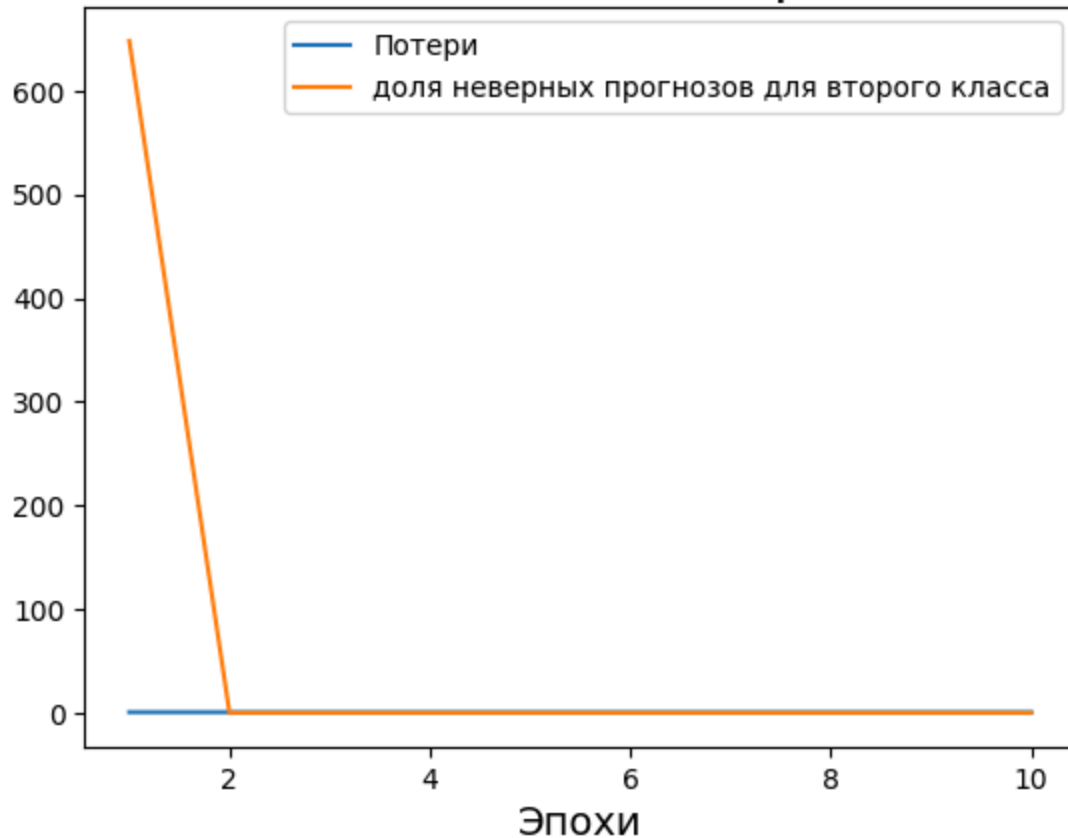
```
In [161... hist = model.fit(X_train, y_train, epochs = 10)
```

```
Epoch 1/10
1349/1349 ————— 4s 1ms/step - FN: 640.5304 - loss: 0.6609
Epoch 2/10
1349/1349 ————— 4s 3ms/step - FN: 0.0000e+00 - loss: 0.6484
Epoch 3/10
1349/1349 ————— 5s 3ms/step - FN: 0.0000e+00 - loss: 0.6442
Epoch 4/10
1349/1349 ————— 2s 1ms/step - FN: 0.0000e+00 - loss: 0.6463
Epoch 5/10
1349/1349 ————— 2s 2ms/step - FN: 0.0000e+00 - loss: 0.6419
Epoch 6/10
1349/1349 ————— 4s 3ms/step - FN: 0.0000e+00 - loss: 0.6446
Epoch 7/10
1349/1349 ————— 3s 1ms/step - FN: 0.0000e+00 - loss: 0.6407
Epoch 8/10
1349/1349 ————— 1s 1ms/step - FN: 0.0000e+00 - loss: 0.6429
Epoch 9/10
1349/1349 ————— 3s 3ms/step - FN: 0.0000e+00 - loss: 0.6374
Epoch 10/10
1349/1349 ————— 4s 1ms/step - FN: 0.0000e+00 - loss: 0.6393
```

```
In [162... plt.plot(np.arange(1, 11), hist.history['loss'], label='Потери')
# plt.plot(np.arange(1, 11), hist.history['accuracy'], label='доля неверных прогнозов
plt.plot(np.arange(1, 11), hist.history['FN'], label='доля неверных прогнозов для в

plt.title('Показатели качества нейронной сети', size=20)
plt.xlabel('Эпохи', size=14)
plt.legend();
```

## Показатели качества нейронной сети



```
In [163... prediction = model.predict(X_test)
prediction
```

338/338 ————— 1s 2ms/step

```
Out[163... array([[0.66179705],
        [0.6714588 ],
        [0.65332043],
        ...,
        [0.6598264 ],
        [0.66083145],
        [0.6537504 ]], dtype=float32)
```

```
In [164... y_pred = np.array([1 if prob > 0.5 else 0 for prob in np.ravel(prediction)])
```

```
In [166... loss, false_negatives = model.evaluate(X_test, y_test)
loss, false_negatives
```

338/338 ————— 1s 2ms/step - FN: 0.0000e+00 - loss: 0.6321

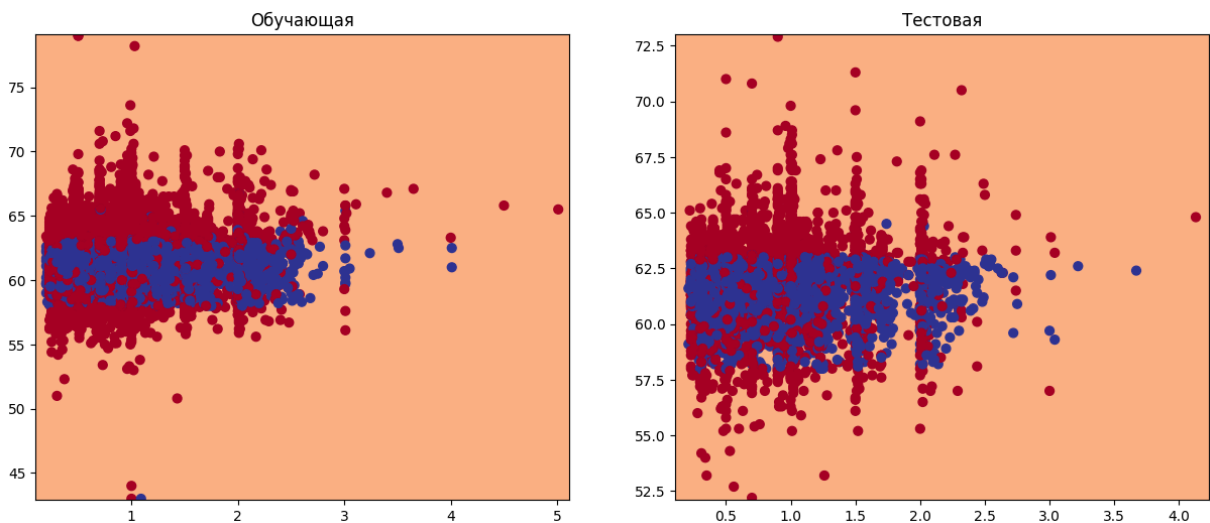
Out[166... (0.6378081440925598, 0.0)

```
In [167... def plot_decision_boundary(model, X, y):
    # Найдем диапазоны изменения по осям и построим сетку
    x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
    y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                          np.linspace(y_min, y_max, 100))

    # Набор данных для прогнозирования
    X_in = np.c_[xx.ravel(), yy.ravel()]
    # Прогноз при помощи обученной модели
    y_pred = model.predict(X_in)
    # Проверка мультиклассовости
    if len(y_pred[1]) > 1:
        # мультиклассовая классификация
        # изменяем форму прогноза для визуализации
        y_pred = np.argmax(y_pred, axis=1).reshape(xx.shape)
    else:
        # бинарная классификация
        y_pred = np.round(y_pred).reshape(xx.shape)
    # Рисуем границу решения
    plt.contourf(xx, yy, y_pred, cmap=plt.cm.RdYlBu, alpha=0.7)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.RdYlBu)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
```

```
In [171... plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.title("Обучающая")
plot_decision_boundary(model, X=X_train.values, y = y_train.values)
plt.subplot(1, 2, 2)
plt.title("Тестовая")
plot_decision_boundary(model, X=X_test.values, y = y_test.values)
```

313/313 ————— 1s 2ms/step  
 313/313 ————— 1s 2ms/step



## 6. Визуализируйте на одном рисунке ROC-кривые для построенных классификаторов на основе нейронных сетей, вычислите площади под ROC-кривыми методом трапеций или иным методом и создайте легенду с указанием площадей кривых.

```
In [172... def TN(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 0) & (y_predict == 0))
```

```
In [173... def FP(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 0) & (y_predict == 1))
```

```
In [174... def FN(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 1) & (y_predict == 0))
```

```
In [175... def TP(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 1) & (y_predict == 1))
```

```
In [176... def confusion_matrix(y_true, y_predict):
    return np.array([
        [TP(y_true, y_predict), FN(y_true, y_predict)],
        [FP(y_true, y_predict), TN(y_true, y_predict)]
    ])
```

```
In [178... confusion_matrix(y_test, y_pred)
```

```
Out[178... array([[7105,    0],
        [3683,    0]], dtype=int64)
```

```
In [179... def true_false_positive(threshold_vector, y_test):
    true_positive = np.equal(threshold_vector, 1) & np.equal(y_test, 1)
    true_negative = np.equal(threshold_vector, 0) & np.equal(y_test, 0)
    false_positive = np.equal(threshold_vector, 1) & np.equal(y_test, 0)
    false_negative = np.equal(threshold_vector, 0) & np.equal(y_test, 1)

    tpr = true_positive.sum() / (true_positive.sum() + false_negative.sum())
    fpr = false_positive.sum() / (false_positive.sum() + true_negative.sum())

    return tpr, fpr
```

```
In [180... def roc_from_scratch(probabilities, y_test, partitions=100):
    roc = np.array([])
    for i in range(partitions + 1):

        threshold_vector = np.greater_equal(probabilities, i / partitions).astype(int)
        tpr, fpr = true_false_positive(threshold_vector, y_test)
        roc = np.append(roc, [fpr, tpr])

    return roc.reshape(-1, 2)
```

```
In [246... prediction = model.predict(X)
prediction[0:5]
```

1686/1686 ————— 3s 2ms/step

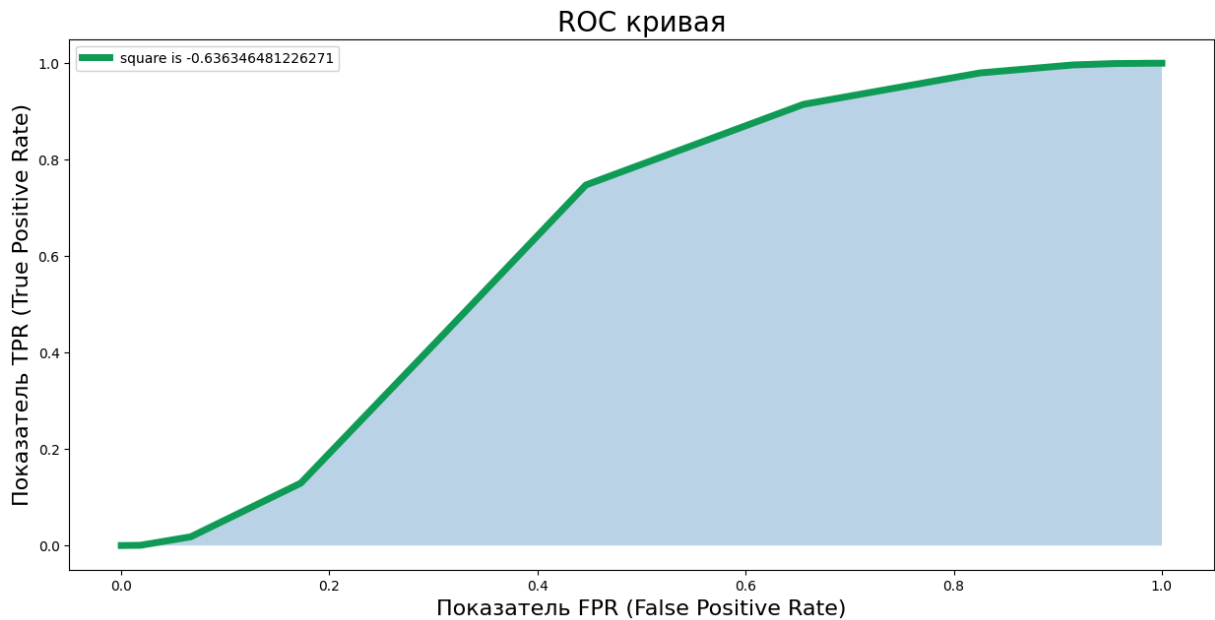
```
Out[246... array([[0.6589062 ],
        [0.6653069 ],
        [0.6754862 ],
        [0.65325296],
        [0.6478472 ]], dtype=float32)
```

```
In [270... def trapezoid(x, y):
    area = 0
    for i in range(len(x) - 1):
        dx = x[i+1]-x[i]
        # Calculate the area of the trapezoid on this subinterval
        area += (y[i] + y[i + 1]) / 2 * dx
    return area
```

In [ ]:

```
In [276... plt.figure(figsize=(15,7))

ROC = roc_from_scratch(prediction.reshape(-1),y,partitions=200)
#plt.scatter(ROC[:,0],ROC[:,1],color='#0F9D58',s=100)
plt.plot(ROC[:,0],ROC[:,1],color='#0F9D58',lw=5, label = "square is " + str(trapezoid(
plt.fill_between(ROC[:, 0], ROC[:, 1], alpha = 0.3)
plt.legend()
plt.title('ROC кривая',fontsize=20)
plt.xlabel('Показатель FPR (False Positive Rate)',fontsize=16)
plt.ylabel('Показатель TPR (True Positive Rate)',fontsize=16);
```



**7. Определите исходном наборе данных дополнительный признак, отличный от указанных в задании двух независимых признаков, принимающий непрерывные значения и имеющий максимальную дисперсию.**

In [281... `df.columns`

Out[281... `Index(['carat', 'cut', 'depth', 'table', 'price', 'x', 'y', 'z', 'res'], dtype='object')`

In [284... `df[['table', 'price', 'x', 'y', 'z']].var()`

Out[284... `table 4.992948e+00`  
`price 1.591563e+07`  
`x 1.258347e+00`  
`y 1.304472e+00`  
`z 4.980109e-01`  
`dtype: float64`

**Максимальная дисперсия у признака 'price'**

**8. Визуализируйте точки набора данных в трехмерном пространстве с координатами, соответствующими трем**

независимым признакам, отображая точки различных классов разными цветами. Подпишите оси и рисунок, создайте легенду для классов набора данных.

```
In [285... from matplotlib import cm
```

```
In [286... df['cut'].value_counts()
```

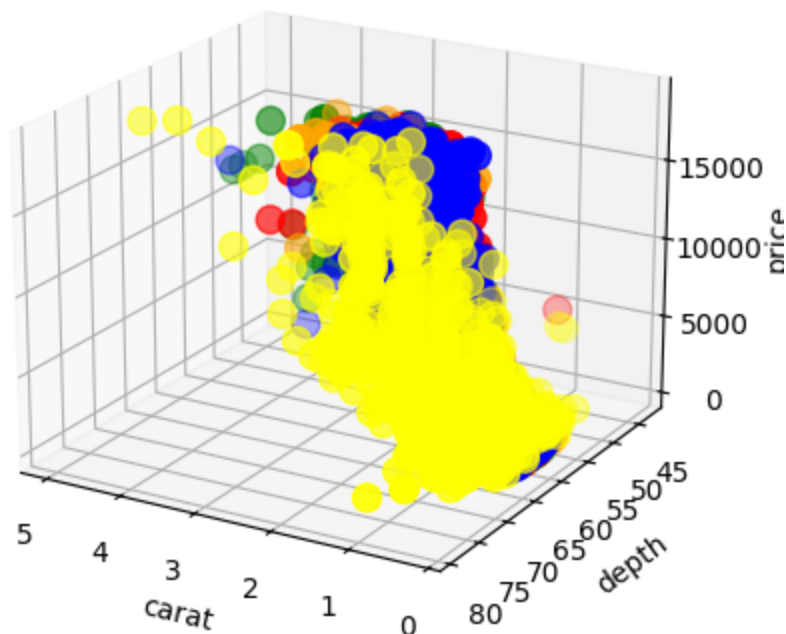
```
Out[286... cut
Ideal          21551
Premium        13791
Very Good      12082
Good           4906
Fair           1610
Name: count, dtype: int64
```

```
In [292... fig = plt.figure()
ax = plt.axes(projection='3d')

ax.set_xlabel('carat')
ax.set_ylabel('depth')
ax.set_zlabel('price')

ax.scatter(df[df['cut'] == 'Ideal']['carat'], df[df['cut'] == 'Ideal']['depth'],
           df[df['cut'] == 'Ideal']['price'], s=100, c='r')
ax.scatter(df[df['cut'] == 'Premium']['carat'], df[df['cut'] == 'Premium']['depth'],
           df[df['cut'] == 'Premium']['price'], s=100, c='g')
ax.scatter(df[df['cut'] == 'Very Good']['carat'], df[df['cut'] == 'Very Good']['depth'],
           df[df['cut'] == 'Very Good']['price'], s=100, c='b')
ax.scatter(df[df['cut'] == 'Good']['carat'], df[df['cut'] == 'Good']['depth'],
           df[df['cut'] == 'Good']['price'], s=100, c='orange')
ax.scatter(df[df['cut'] == 'Fair']['carat'], df[df['cut'] == 'Fair']['depth'],
           df[df['cut'] == 'Fair']['price'], s=100, c='yellow')

ax.view_init(azim = 120, elev = 20)
```



9. Разбейте исходный набор данных на обучающую и тестовую выборки. Постройте нейронную сеть для многоклассовой классификации с нормализующим слоем и параметрами, соответствующими лучшей нейронной сети для бинарной классификации из п.4, и обучите ее на обучающей выборке, контролируя процесс ее обучения.

```
In [293... X = df[['carat', 'depth', 'price']]
           y = df['cut']
```

```
In [294... X_train, y_train, X_test, y_test = train_test_split(X, y, test_ratio=0.2, seed = 51
```

```
In [297... X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
Out[297... ((43152, 3), (43152,)), (10788, 3), (10788,))
```

```
In [298... def to_one_hot(labels, dimension=5):
           results = np.zeros((len(labels), dimension))
           for i, label in enumerate(labels):
```



```

        results[i, label] = 1.
    return results

```

In [299... `list(enumerate(['Ideal', 'Premium', 'Very Good', 'Good', 'Fair']))`

Out[299... `[(0, 'Ideal'), (1, 'Premium'), (2, 'Very Good'), (3, 'Good'), (4, 'Fair')]`

In [304... `dictionary = {`  
 `'Fair':0,`  
 `'Good':1,`  
 `'Very Good':2,`  
 `'Premium':3,`  
 `'Ideal':4`  
`}`

In [306... `y_train = y_train.apply(lambda x: dictionary[x])`  
`y_test = y_test.apply(lambda x: dictionary[x])`  
`y_train.head(), y_test.head()`

Out[306... `(26368 2`  
`14952 2`  
`11606 2`  
`6425 3`  
`7784 3`  
`Name: cut, dtype: int64,`  
`50361 3`  
`20942 3`  
`24143 2`  
`4313 4`  
`8727 4`  
`Name: cut, dtype: int64)`

In [307... `y_train = to_one_hot(y_train.values)`  
`y_test = to_one_hot(y_test.values)`  
`y_train.shape, y_test.shape`

Out[307... `((43152, 5), (10788, 5))`

In [308... `y_test, y_train`


Out[308... `(array([[0., 0., 0., 1., 0.],`  
 `[0., 0., 0., 1., 0.],`  
 `[0., 0., 1., 0., 0.],`  
 `...,`  
 `[0., 0., 0., 0., 1.],`  
 `[0., 0., 1., 0., 0.],`  
 `[0., 0., 1., 0., 0.])),`  
`array([[0., 0., 1., 0., 0.],`  
 `[0., 0., 1., 0., 0.],`  
 `[0., 0., 1., 0., 0.],`  
 `...,`  
 `[0., 1., 0., 0., 0.],`  
 `[0., 0., 0., 1., 0.],`  
 `[0., 0., 0., 0., 1.]])`


```
In [310... feature_normalizer = tf.keras.layers.Normalization(axis=None, input_shape=(X.shape[1],))
feature_normalizer.adapt(X_train.values)


In [317... model = tf.keras.Sequential([
    feature_normalizer,
    tf.keras.layers.Dense(128, activation=leaky_relu),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dense(5, activation="softmax")
])


In [318... model.compile(optimizer="rmsprop",
    loss="categorical_crossentropy",
    metrics=["accuracy"])


In [319... history = model.fit(X_train,
    y_train,
    epochs=20,
    # уровень выводимой информации
    verbose=1,
    # проверка (валидация) на 20% обучающих данных
    validation_split = 0.2)
```


Epoch 1/20  
**1079/1079**  5s 3ms/step - accuracy: 0.3963 - loss: 1.3803 - val\_accuracy: 0.4037 - val\_loss: 1.3521


Epoch 2/20  
**1079/1079**  2s 2ms/step - accuracy: 0.3947 - loss: 1.3620 - val\_accuracy: 0.4070 - val\_loss: 1.3449


Epoch 3/20  
**1079/1079**  2s 2ms/step - accuracy: 0.3914 - loss: 1.3632 - val\_accuracy: 0.4038 - val\_loss: 1.3421


Epoch 4/20  
**1079/1079**  2s 2ms/step - accuracy: 0.3959 - loss: 1.3590 - val\_accuracy: 0.4061 - val\_loss: 1.3413


Epoch 5/20  
**1079/1079**  2s 2ms/step - accuracy: 0.3941 - loss: 1.3550 - val\_accuracy: 0.4025 - val\_loss: 1.3417


Epoch 6/20  
**1079/1079**  2s 2ms/step - accuracy: 0.3939 - loss: 1.3518 - val\_accuracy: 0.4051 - val\_loss: 1.3411


Epoch 7/20  
**1079/1079**  2s 2ms/step - accuracy: 0.4026 - loss: 1.3545 - val\_accuracy: 0.4019 - val\_loss: 1.3419


Epoch 8/20  
**1079/1079**  3s 2ms/step - accuracy: 0.3918 - loss: 1.3579 - val\_accuracy: 0.4077 - val\_loss: 1.3422


Epoch 9/20  
**1079/1079**  2s 2ms/step - accuracy: 0.4004 - loss: 1.3557 - val\_accuracy: 0.4073 - val\_loss: 1.3396

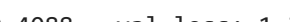
Epoch 10/20  
**1079/1079**  2s 2ms/step - accuracy: 0.4027 - loss: 1.3462 - val\_accuracy: 0.4098 - val\_loss: 1.3450

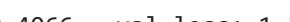
Epoch 11/20  
**1079/1079**  3s 2ms/step - accuracy: 0.4010 - loss: 1.3494 - val\_accuracy: 0.4134 - val\_loss: 1.3389

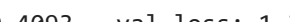
Epoch 12/20  
**1079/1079**  3s 2ms/step - accuracy: 0.3962 - loss: 1.3526 - val\_accuracy: 0.4112 - val\_loss: 1.3388


Epoch 13/20  
**1079/1079**  3s 2ms/step - accuracy: 0.4049 - loss: 1.3464 - val\_accuracy: 0.4070 - val\_loss: 1.3384


Epoch 14/20  
**1079/1079**  3s 2ms/step - accuracy: 0.4049 - loss: 1.3493 - val\_accuracy: 0.4063 - val\_loss: 1.3414

Epoch 15/20  
**1079/1079**  2s 2ms/step - accuracy: 0.4041 - loss: 1.3487 - val\_accuracy: 0.4088 - val\_loss: 1.3386

Epoch 16/20  
**1079/1079**  3s 2ms/step - accuracy: 0.3996 - loss: 1.3536 - val\_accuracy: 0.4066 - val\_loss: 1.3376

Epoch 17/20  
**1079/1079**  2s 2ms/step - accuracy: 0.4005 - loss: 1.3490 - val\_accuracy: 0.4093 - val\_loss: 1.3389

Epoch 18/20  
**1079/1079**  2s 2ms/step - accuracy: 0.4037 - loss: 1.3472 - val\_accuracy: 0.4084 - val\_loss: 1.3394

Epoch 19/20  
**1079/1079**  2s 2ms/step - accuracy: 0.4031 - loss: 1.3519 - val\_accuracy: 0.4031 - val\_loss: 1.3519

ccuracy: 0.4124 - val\_loss: 1.3380

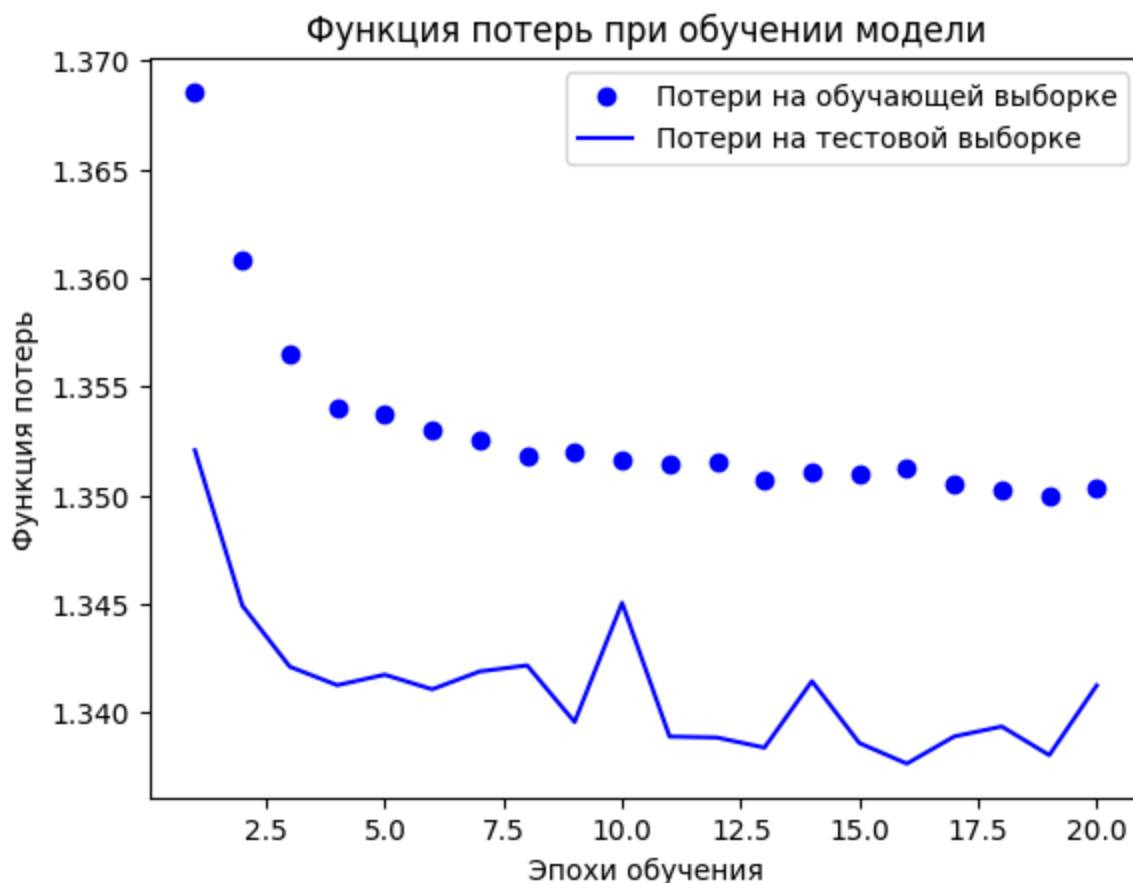
Epoch 20/20

1079/1079 ————— 2s 2ms/step - accuracy: 0.4026 - loss: 1.3522 - val\_a  
ccuracy: 0.4101 - val\_loss: 1.3412

## 10. Постройте кривые обучения в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду.

In [320...

```
loss = history.history["loss"]  
val_loss = history.history["val_loss"]  
epochs = range(1, len(loss) + 1)  
plt.plot(epochs, loss, "bo", label="Потери на обучающей выборке")  
plt.plot(epochs, val_loss, "b", label="Потери на тестовой выборке")  
plt.title("Функция потерь при обучении модели")  
plt.xlabel("Эпохи обучения")  
plt.ylabel("Функция потерь")  
plt.legend();
```



К сожалению, нейронная сеть плохо обучается из-за того, что наши признаки

никак не коррелируют с качеством алмазов, что заметно еще в самом начале(точки всех классов алмаза наплывают друг на друга, из-за чего провести классификацию не получается)

In [ ]: