```
%matplotlib inline
import numpy as np
import math
import matplotlib.pyplot as plt
import pylab
import pandas as pd

#df = pd.read_excel('/data_ml_v2-03.xlsx')
#mylist = df['data'].tolist()

file = '/data_ml_v2-03.xlsx'

xl = pd.ExcelFile(file)

print(xl.sheet_names)

df1 = xl.parse('data')
```

```
['data']
```

```
data_list = df1['x1,x2,label,score'].values

print(len(data_list))
```

```
350
```

```
data = []
data.clear()
for i in range(len(data_list)):
    string = data_list[i]
    lt = string.split(',')
    new_lt = []
    for j in range(len(lt)):
        new_lt.append(float(lt[j]))
    data.append(new_lt)

print(len(data)) # список со списками вида [x1, x2, label, score]
```

```
350
```

```
N = 350

x1_false = []
x2_false = []
x1_true = []
x2_true = []
for i in range(N):
    if (data[i][2] == -1):
        x1_false.append(data[i][0])
        x2_false.append(data[i][1])
    else:
```

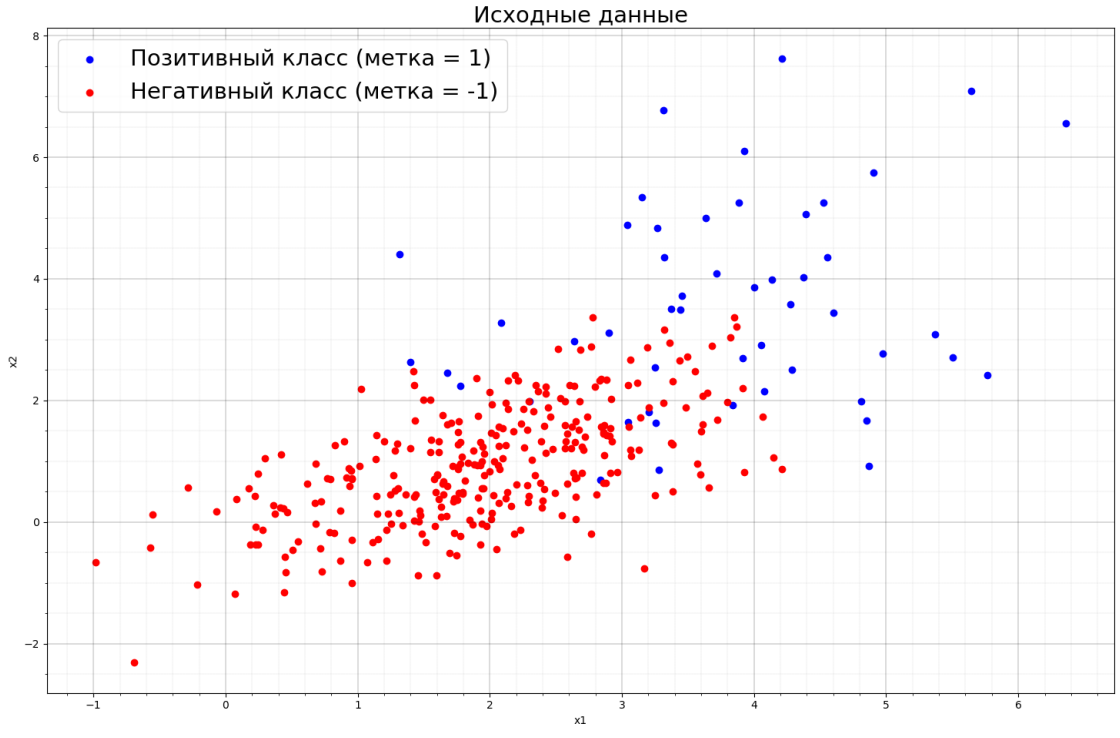
```
        x1_true.append(data[i][0])
        x2_true.append(data[i][1])

print(len(x1_false),',',len(x2_false),' |
',len(x1_true),',',len(x2_true))
```

```
300 , 300  |  50 , 50
```

```
fig, ax = plt.subplots()
fig.set_figwidth(15)
fig.set_figheight(10)
ax.set_title(label="Исходные данные",fontdict={'fontsize': 21})
pylab.scatter(x1_true,x2_true,color='b',label='Позитивный класс (метка
= 1)',lw=1)
pylab.scatter(x1_false,x2_false,color='r',label='Негативный класс
(метка = -1)',lw=1)

ax.set_xlabel(xlabel="x1")
ax.set_ylabel(ylabel="x2")
plt.rcParams.update({'font.size': 21})
ax.minorticks_on()
ax.grid(which='major', color = 'k', linewidth = 0.2)
ax.grid(which='minor', color = 'k', linewidth = 0.1, linestyle = ':')
fig.tight_layout()
plt.legend()
plt.show()
```




```
x1_false = []
x2_false = []
```

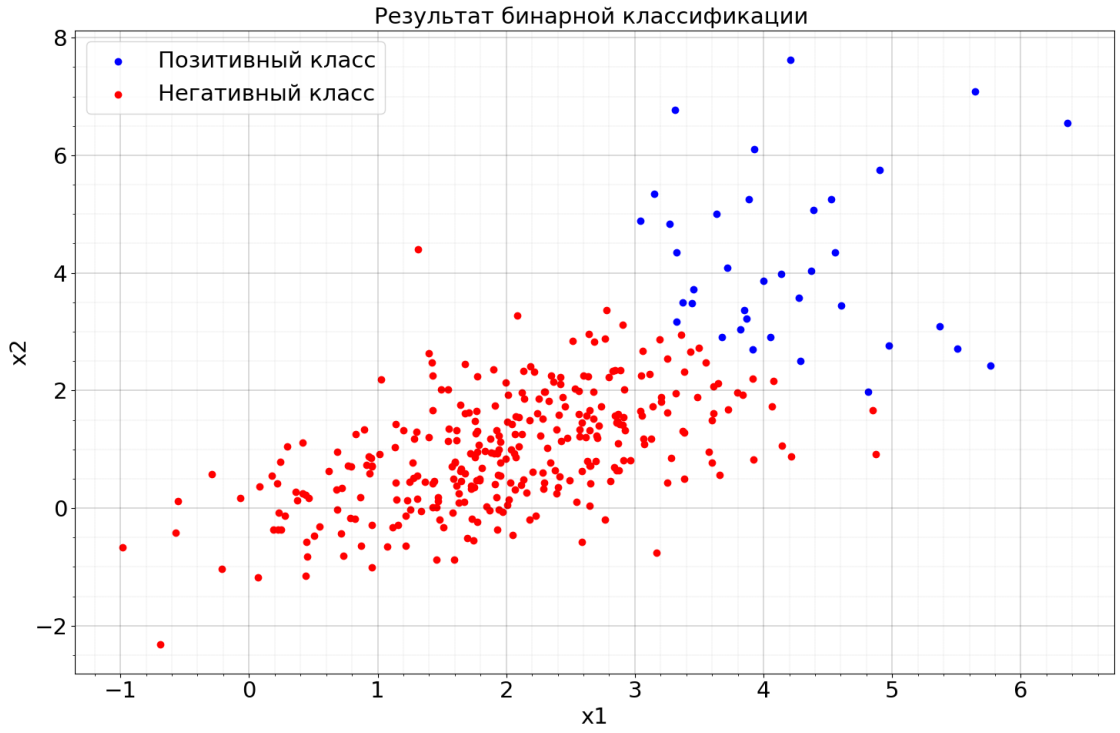
```
x1_true = []
x2_true = []
for i in range(N):
    #print(type(data[i][0]))
    #print(data[i][0] , data[i][1])
    if (data[i][3] <= 0):
        x1_false.append(data[i][0])
        x2_false.append(data[i][1])
    else:
        x1_true.append(data[i][0])
        x2_true.append(data[i][1])

print(len(x1_false),',',len(x2_false),' |
',len(x1_true),',',len(x2_true))
```

```
313 , 313  |  37 , 37
```

```
fig, ax = plt.subplots()
fig.set_figwidth(15)
fig.set_figheight(10)
ax.set_title(label="Результат бинарной классификации
",fontdict={'fontsize': 21})
pylab.scatter(x1_true,x2_true,color='b',label='Позитивный класс',lw=1)
pylab.scatter(x1_false,x2_false,color='r',label='Негативный
класс',lw=1)

ax.set_xlabel(xlabel="x1")
ax.set_ylabel(ylabel="x2")
plt.rcParams.update({'font.size': 21})
ax.minorticks_on()
ax.grid(which='major', color = 'k', linewidth = 0.2)
ax.grid(which='minor', color = 'k', linewidth = 0.1, linestyle = ':')
fig.tight_layout()
plt.legend()
plt.show()
```




```
TP = 0
FN = 0
FP = 0
TN = 0
x1_TP = []
x2_TP = []
x1_FN = []
x2_FN = []
x1_FP = []
x2_FP = []
x1_TN = []
x2_TN = []
lt = []
data_full = []
for i in range(N):
    if (data[i][2] == 1 and data[i][3] > 0):
        x1_TP.append(data[i][0])
        x2_TP.append(data[i][1])
        lt = data[i]
        lt.append(1)
        data_full.append(lt)
        TP += 1
    elif (data[i][2] == 1 and data[i][3] <= 0):
        x1_FN.append(data[i][0])
        x2_FN.append(data[i][1])
        lt = data[i]
        lt.append(2)
        data_full.append(lt)
```

```
        FN += 1
    elif (data[i][2] == -1 and data[i][3] > 0):
        x1_FP.append(data[i][0])
        x2_FP.append(data[i][1])
        lt = data[i]
        lt.append(3)
        data_full.append(lt)
        FP += 1
    elif (data[i][2] == -1 and data[i][3] <= 0):
        x1_TN.append(data[i][0])
        x2_TN.append(data[i][1])
        lt = data[i]
        lt.append(4)
        data_full.append(lt)
        TN += 1

print('Количество:', len(data_full))
print('Confusion matrix')
print('_____________')
print('|', TP, '|', FN, ' |')
print('_____________')
print('|', FP, ' |', TN, '|')
print('_____________')
```

```
Количество: 350
Confusion matrix
_____________
| 32 | 18  |
_____________
| 5  | 295 |
_____________
```

```
print(TP + TN + FP + FN)
```

```
350
```

```
#data_full

from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc, precision_recall_curve
from sklearn.metrics import roc_auc_score

y_true = []
y_score = []
for i in range(len(data_full)):
    y_true.append(data_full[i][2])
    y_score.append(data_full[i][3])
print(y_true, y_score)
```

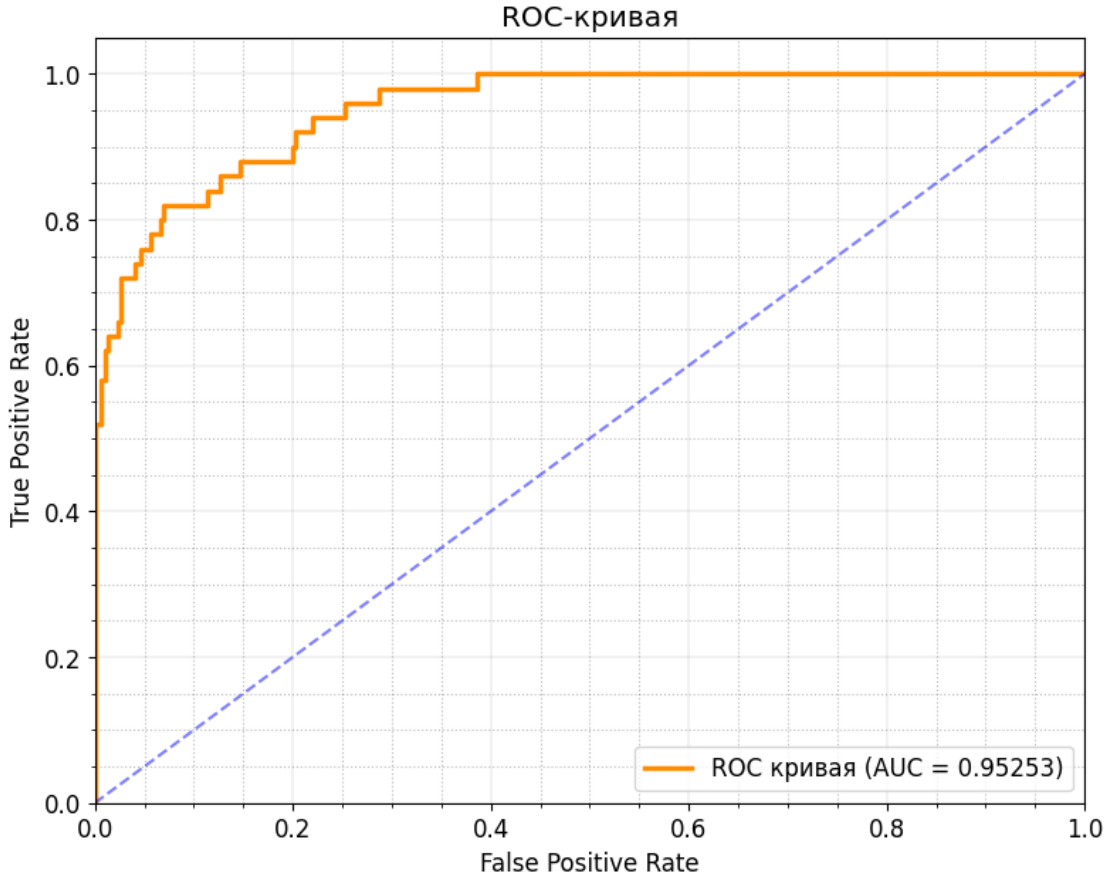
[-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -
1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
-1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0] [-2.9341, -4.7261, -
2.7575, -1.3024, -3.0209, -3.5991, -1.7866, -2.044, -2.4764, -1.5206,
-1.5761, -1.9428, -3.4747, -3.3903, -3.612, -6.7941, -2.3966, -4.2183,
-1.1207, -3.6051, -3.017, -1.6044, -2.7352, -4.2082, -0.78882, -
1.6376, -2.1202, -4.0793, -3.3205, -1.5978, -1.7692, -1.0907, -
0.88981, -5.2411, -5.168, -3.2855, -2.4435, -1.1246, -3.1331, -4.6777,
-1.5709, -2.97, -3.4185, -4.006, -3.271, -3.3522, -1.0002, -2.5132, -
0.63392, -2.4092, -4.037, -3.7122, -2.2366, -2.9955, -3.6294, -1.8691,
-3.1522, -2.2767, -1.9917, -3.0384, -1.6683, -3.3891, -3.2323, -
0.91816, -0.86289, -1.901, -3.6665, -0.87332, -2.7085, -2.4125, -
3.844, -4.4976, -2.8973, -1.4929, -3.613, -1.5451, 0.41757, -2.6268, -
3.1849, -3.3396, -3.3316, -4.8101, -2.0304, -1.3535, -2.9212, -4.7141,
-4.7166, -3.2333, -2.3412, -2.3505, -1.1358, -0.35848, -3.3818, -
3.4916, -1.6565, -0.5912, -2.7599, -2.705, -3.7549, -1.5016, -2.2966,
-1.7411, -4.2862, -2.6305, -2.0144, -4.72, -0.3476, -5.4883, -2.9202,
-1.1924, -1.7636, -4.1656, -3.4901, -2.8424, -4.1408, -1.3293, -
2.8495, -2.6295, -2.9419, -0.20456, -2.0051, -1.6812, -2.2526, -
2.2177, -3.526, -3.2345, -3.7755, -1.9755, -3.8725, -1.8728, -2.3964,
-2.5416, -4.1786, -2.8311, -2.8763, -2.8154, -0.73227, -3.5886, -
2.6249, -2.7551, -1.375, -1.0979, -2.0116, -4.4595, -1.038, -3.154, -
3.0237, -2.1875, -1.3702, -2.7482, -2.1313, -3.0557, -2.0119, -2.9446,

```
-3.3143, -2.8824, -4.0965, 0.05208, -1.6421, -1.7851, -2.1364, -
3.3678, -1.4154, -0.6359, -2.9938, -5.6963, -2.1382, -2.1399, -
0.60469, -3.1397, -2.593, -3.1519, -2.6756, -2.7201, -2.7159, -3.8237,
-4.1703, -3.7797, -1.6032, -3.6606, -0.15357, -4.5895, -2.5092, -
3.6007, -3.2795, -1.2782, -2.5749, -2.2311, -1.7534, -4.4335, -4.0972,
-4.0968, -1.8404, -3.1464, -3.3445, -1.733, -0.61867, -1.6598, -
2.1961, -2.538, -2.2601, -2.5926, 0.028802, -1.9121, -3.8742, -3.497,
-0.29226, -4.3084, -3.4094, -3.4441, -1.6704, -3.3545, -2.6001, -
0.11454, -3.2918, -1.531, -1.4857, -3.5017, -1.2785, -2.976, -1.6773,
-4.3832, -3.193, -1.5291, -2.2753, -4.4665, -1.3864, -5.4212, -
0.53913, -0.91299, -1.3043, -2.07, -1.6159, -2.8118, -3.0455, -3.878,
-3.0121, -1.3897, -2.6133, -3.546, -3.1778, -3.339, -1.2127, -2.728, -
2.6658, -2.8649, -0.93558, -2.5585, -4.6917, -2.6738, -1.4646, -
3.3176, -0.52149, -3.7372, -1.3746, -3.4003, -0.85995, -3.4099, -
4.9048, -1.9712, -2.8404, -1.8712, -2.434, -3.0207, -2.5415, -1.6477,
-2.0286, -1.8876, -1.9957, -2.4907, -3.1572, -4.2076, -2.439, 0.2514,
-2.1535, -1.0435, -2.1437, -3.8702, -2.2157, -4.1225, -4.4452, -
1.7133, -2.298, -3.8537, -4.472, -2.3088, -1.8765, -1.6589, -4.1044, -
0.57547, -4.2778, -3.6009, -1.4005, -2.4616, -1.3685, 0.52394, -
2.2122, -1.6654, -2.6141, -0.28205, 1.7397, -0.60974, 0.95799, -
1.6119, -1.673, 1.0004, -0.55806, -0.27843, 1.3711, -1.7984, -0.27541,
-1.3297, 1.058, 1.2733, -0.27696, 0.76978, 1.6626, 3.0516, 1.0606,
0.037896, -0.17203, 0.54667, 1.6202, -1.1047, 0.99978, 2.1915, 4.0744,
-1.2125, 0.32406, -0.49949, 0.13122, 0.95454, -0.66765, -2.2072, -
0.74071, 1.3006, 1.007, 0.052208, 0.35831, 1.1958, 2.8344, 2.7078,
2.0184, 0.29738, 2.4211, -1.5414, 1.1875, -1.5695, 4.6015, 4.5664]

fpr, tpr, treshold = roc_curve(y_true, y_score)
roc_auc = auc(fpr, tpr)
# строим график
plt.figure(figsize=(9, 7))
plt.plot(fpr, tpr, color='darkorange',linewidth = 2.5,
         label='ROC кривая (AUC = %0.5f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='blue', alpha = 0.5, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-кривая')
plt.legend(loc="lower right")
plt.minorticks_on()
#  Определяем внешний вид линий основной сетки:
plt.grid(which='major',
        color = 'k', alpha = 0.25,
        linewidth = 0.3)

#  Определяем внешний вид линий вспомогательной
#  сетки:
plt.grid(which='minor',
        color = 'k', alpha = 0.25,
```

```
        linestyle = ':')
plt.rcParams['font.size'] = '12'
plt.show()
```



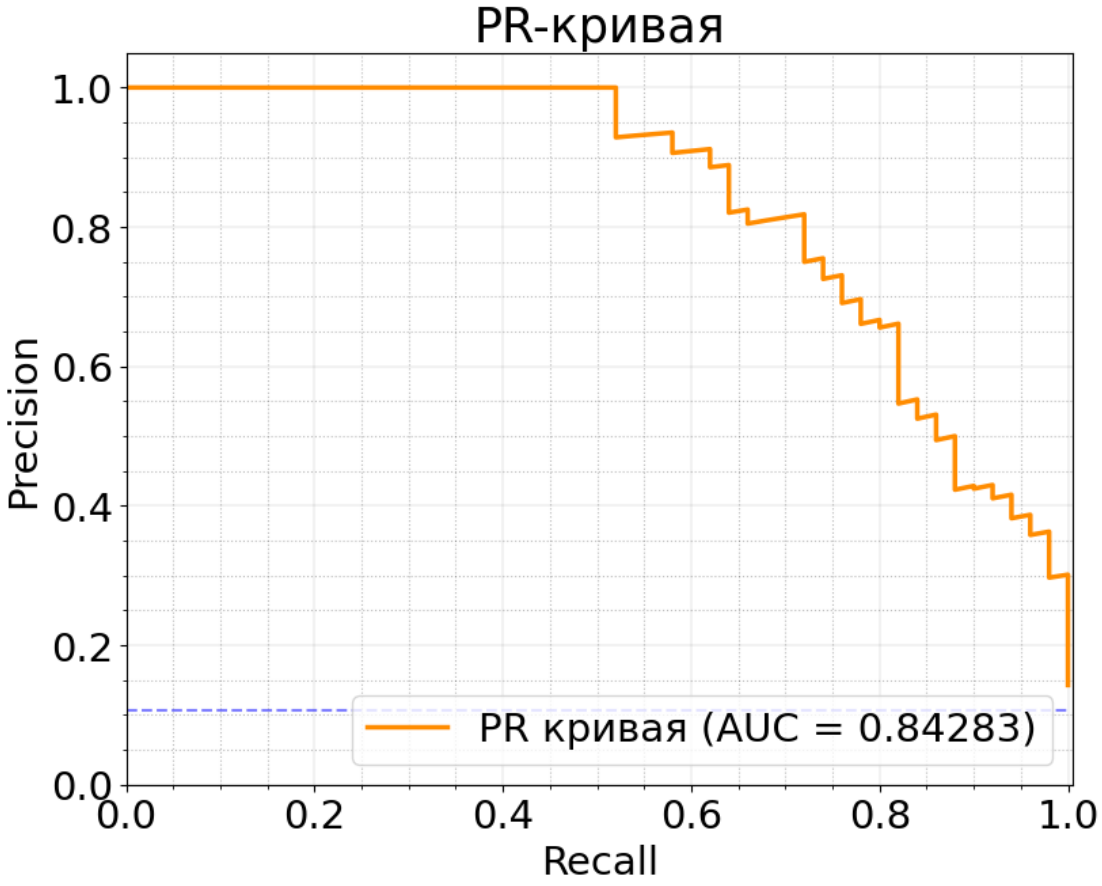

```
roc_auc
```

```
0.9525333333333332
```

```
precision, recall, thresholds = precision_recall_curve(y_true,
y_score)
pr_auc = auc(recall, precision)
# строим график
plt.figure(figsize=(9, 7))
plt.plot(recall, precision, color='darkorange',linewidth = 2.5,
         label='PR кривая (AUC = %0.5f)' % pr_auc)
plt.plot([0, 1], [37/350, 37/350], color='blue', alpha = 0.5,
linestyle='--')
plt.xlim([0.0, 1.005])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PR-кривая')
plt.legend(loc="lower right")
plt.minorticks_on()
```

```
#  Определяем внешний вид линий основной сетки:
plt.grid(which='major',
        color = 'k', alpha = 0.25,
        linewidth = 0.3)

#  Определяем внешний вид линий вспомогательной
#  сетки:
plt.grid(which='minor',
        color = 'k', alpha = 0.25,
        linestyle = ':')
plt.rcParams['font.size'] = '12'
plt.show()
```



Гипотезы U - критерия Манна-Уитни H0: Классификационные очки положительного класса превышают очки отрицательного класса H1: Классификационные очки положительного класса не превышают очки отрицательного класса

```
from scipy.stats import mannwhitneyu

s_pos = []
s_neg = []
Rp = 0
```

```
Rn = 0
for i in range(N):
    if (data_full[i][3] <= 0):
        s_neg.append(abs(data_full[i][3]))
        Rn += abs(data_full[i][3])
    else:
        s_pos.append(data_full[i][3])
        Rp += data_full[i][3]
print(len(s_pos),len(s_neg))
```

```
37 313
```

```
U1, p = mannwhitneyu(s_pos, s_neg)
print(U1,p)
```

```
2644.0 6.468675862682975e-08
```

```
nx, ny = len(s_pos), len(s_neg)
U2 = nx*ny - U1
print(U2)
```

```
8937.0
```

```
Rp
```

```
50.647526000000006
```

```
Up = Rp + ((50*51)/2)
```

```
Up
```

```
1325.647526
```

```
Rn
```

```
792.7434999999997
```

```
Un = Rn + ((300*301)/2)
```

```
Un
```

```
45942.7435
```

```
AUC_r = Un / (Up + Un)
```

```
AUC_r
```

```
0.971954883650031
```

```
SPEC = TN / (TN + FP)
SENS = TP / (TP + FN)
```

```
print(SENS , SPEC)
```

```
0.64 0.9833333333333333
```

```
print(abs(SENS - SPEC))
```

```
0.34333333333333327
```

```
y_true = []
y_score = []
for i in range(len(data_full)):
    y_true.append(data_full[i][2])
    y_score.append(data_full[i][3])

fpr, tpr, treshold = roc_curve(y_true, y_score)
roc_auc = auc(fpr, tpr)
print(fpr)
```

```
[0.         0.         0.         0.00666667 0.00666667 0.01
 0.01       0.01333333 0.01333333 0.02333333 0.02333333 0.02666667
 0.02666667 0.04       0.04       0.04666667 0.04666667 0.05666667
 0.05666667 0.06666667 0.06666667 0.07       0.07       0.11333333
 0.11333333 0.12666667 0.12666667 0.14666667 0.14666667 0.2
 0.2        0.20333333 0.20333333 0.22       0.22       0.25333333
 0.25333333 0.28666667 0.28666667 0.38666667 0.38666667 1.        ]
```

```
criteria = []
cr = []
fc1 = []
fc1_list = []
fc2 = []
fc2_list = []
fc3 = []
fc3_list = []

cr.clear()
fc1.clear()
fc1_list.clear()
fc2.clear()
fc2_list.clear()
fc3.clear()
fc3_list.clear()
for i in range(len(fpr)):
    cr = []
    cr.append(abs(tpr[i] - fpr[i]))
    fc1.append(abs(tpr[i] - fpr[i]))
    cr.append(fpr[i])
    cr.append(tpr[i])
    fc1_list.append(cr)

    cr = []
    cr.append(tpr[i] + fpr[i] - 1)
    fc2.append(tpr[i] + fpr[i] - 1)
    cr.append(fpr[i])
    cr.append(tpr[i])
```

```
    fc2_list.append(cr)

    cr = []
    cr.append((1 - tpr[i])**2 + (fpr[i] - 1)**2)
    fc3.append((1 - tpr[i])**2 + (fpr[i] - 1)**2)
    cr.append(fpr[i])
    cr.append(tpr[i])
    fc3_list.append(cr)
```

```
import numpy as np
optimal_idx1 = np.argmax(tpr - fpr)
optimal_threshold1 = thresholds[optimal_idx1]
print(optimal_idx1, optimal_threshold1)

optimal_idx2 = np.argmin(abs(tpr + fpr - 1))
optimal_threshold2 = thresholds[optimal_idx2]
print(optimal_idx2, optimal_threshold2)

optimal_idx3 = np.argmin((1 - tpr)**2 + (fpr - 1)**2)
optimal_threshold3 = thresholds[optimal_idx3]
print(optimal_idx3, optimal_threshold3)
```

```
22 -4.3084
27 -4.2076
41 -3.8742
```

```
print('Первый ', fc1[fc1.index(max(fc1))], 'позиция',
fc1.index(max(fc1)))
print('Второй ', fc2[fc2.index(max(fc2))], 'позиция',
fc2.index(max(fc2)))
print('Третий ', fc3[fc3.index(min(fc3))], 'позиция',
fc3.index(min(fc3)))
```

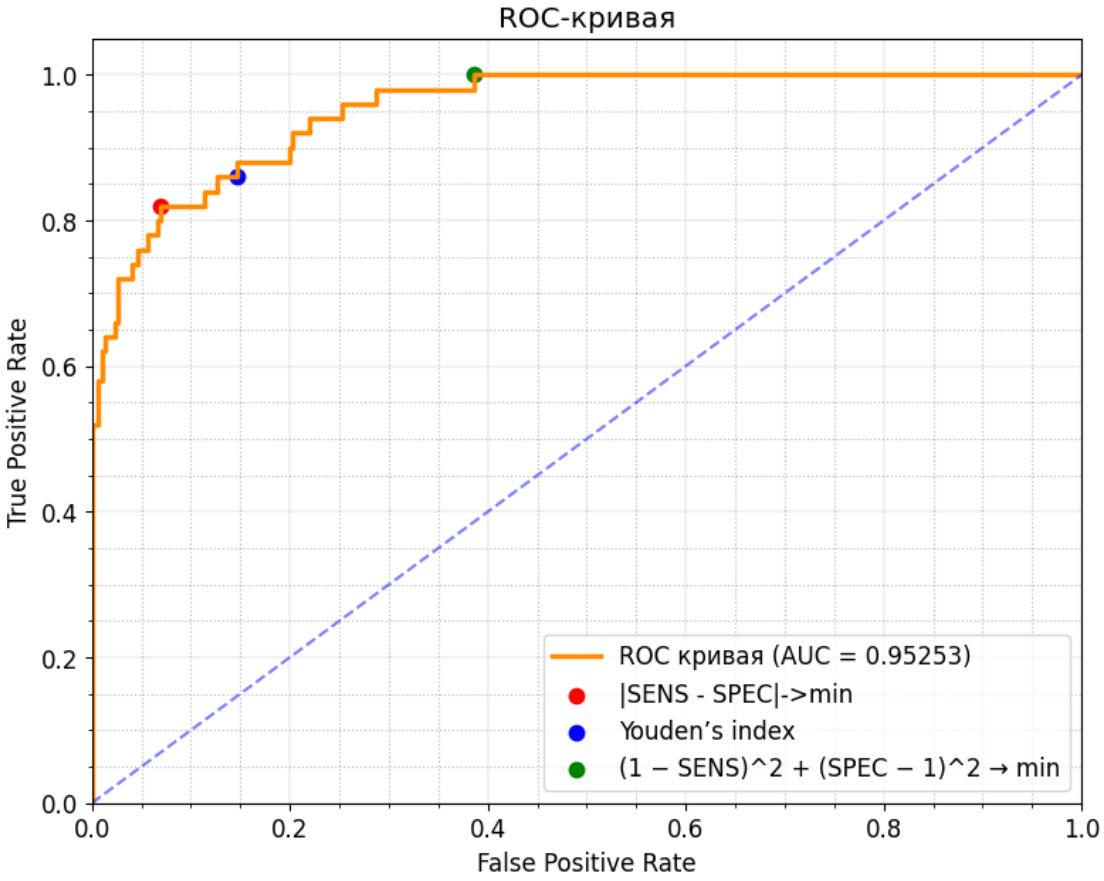
```
Первый  0.75 позиция 22
Второй  1.0 позиция 41
Третий  0.0 позиция 41
```

```
fpr, tpr, treshold = roc_curve(y_true, y_score)
roc_auc = auc(fpr, tpr)
# строим график
plt.figure(figsize=(9, 7))
plt.plot(fpr, tpr, color='darkorange',linewidth = 2.5,
         label='ROC кривая (AUC = %0.5f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='blue', alpha = 0.5, linestyle='--')

plt.scatter(fpr[22], tpr[22], c = 'red',linewidth = 2.5,label='|SENS -
SPEC|->min')    #  цвет точек
plt.scatter(fpr[27], tpr[27], c = 'blue',linewidth =
2.5,label='Youden’s index')    #  цвет точек
plt.scatter(fpr[40], tpr[40], c = 'green',linewidth = 2.5,label='(1 -
SENS)^2 + (SPEC - 1)^2 → min')    #  цвет точек
```

```
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-кривая')
plt.legend(loc="lower right")
plt.minorticks_on()
plt.grid(which='major',
        color = 'k', alpha = 0.25,
        linewidth = 0.3)

plt.grid(which='minor',
        color = 'k', alpha = 0.25,
        linestyle = ':')
plt.rcParams['font.size'] = '12'
plt.show()
```




```
len(data_full)
```

```
350
```

```
data_full_pos = []
data_full_neg = []
```

```
for i in range(len(data_full)):
    if (data_full[i][2] == 1):
        data_full_pos.append(data_full[i])
    else:
        data_full_neg.append(data_full[i])
print(len(data_full_pos), len(data_full_neg))
```

```
50 300
```

```
res_number = []
res_roc_auc = []
res_pr_auc = []

y_pos = []
score_pos = []
lst_pos = []

y_neg = []
score_neg = []
lst_neg = []
for j in range(25):
    y_pos = []
    score_pos = []
    lst_pos = []
    y_neg = []
    score_neg = []
    lst_neg = []
    y_true = []
    y_score = []
    print('Находим для ', 2*(j+1), 2*(j+1)*6)
    lst_pos = np.random.randint(0, 50, 2*(j+1))
    lst_neg = np.random.randint(0, 300, 2*(j+1)*6)
    print(len(lst_pos), len(lst_neg))
    print(lst_pos)
    print(lst_neg)
    for pos in lst_pos:
        y_pos.append(data_full_pos[pos][2])
        score_pos.append(data_full_pos[pos][3])
    for neg in lst_neg:
        y_neg.append(data_full_neg[neg][2])
        score_neg.append(data_full_neg[neg][3])
    y_true = y_pos + y_neg
    y_score = score_pos + score_neg
    print(len(y_true), len(y_score))
    fpr, tpr, treshold = roc_curve(y_true, y_score)
    roc_auc = auc(fpr, tpr)
    precision, recall, thresholds = precision_recall_curve(y_true,
y_score)
    pr_auc = auc(recall, precision)
    res_number.append(2*(j+1) + 2*(j+1)*6)
```

```
    res_roc_auc.append(roc_auc)
    res_pr_auc.append(pr_auc)
```

```
Находим для  2 12
2 12
[31 25]
[259  62 181   9 229 174  45 184 115 287 168  25]
14 14
Находим для  4 24
4 24
[ 5 28 45 46]
[135 192 136 118  36  85  89  79  23 108  40  98 295 232  11 286 266
91
  23  29 217 227 174  26]
28 28
Находим для  6 36
6 36
[46 42 31 23 45  6]
[138  95 164  79 292 101  45 183   0 136 277 205 219  59 121  98 260
177
  29 122  95  14 171 211  34  32  10 236  59  48  42 298 121 224  36
94]
42 42
Находим для  8 48
8 48
[32 47 19 12 16 14 17 33]
[248 162 132 183   2  79  71 113 284 126 134  57 162  29 135 237  13
134
 257  21 207 101 226 282 269 205 222 279 200 230 167 254 256 156 188
58
 232  83 116  34 215  52  87 208 115 274 244 249]
56 56
Находим для  10 60
10 60
[31 25 24 37 48 12 38 32 44 46]
[208 219 116 207 242  84  76  54 185 144 270  10 161 143 119 223 289
52
 108 298 266 117  46 238 217 118  53 113 288 117 282 172  43 140  66
292
 143 206 207 212  82 104  69 233   0 119 235   0 288 200 218 269  17
109
  61   6 243 286 268  97]
70 70
Находим для  12 72
12 72
[27 48 27 47 26 35 41 28 32  2 34 20]
[ 68 201 192 242  22  50  58  66  55   7 141 142 232   4 182  98 238
194
 107 157 251  75  34 168 166 256  16 160  67  17 111  60   2 258   1
247
```

```
 219  20 299 209 232  49 228   4  75 106  97   8  16 294 205  22 112
68
 136  11  66 120 136  74 232 257  75  22 254  63  83 286  20 273 193
148]
84 84
Находим для  14 84
14 84
[28 42  6 46  4  0 11 11 33 25 44 42 43 36]
[ 55  12 180  27 213 244 123 291  22 278  88 110  38 130  44  38 250
13
 249 149   7 213  25 179 254 220 127 187 160 218  14 220 129 138 170
214
  34  26 280 137 202 101 145 250 113  80 191 126 219  20  60 235 143
190
 173  89 183 179 142  29  83 264 279 202 171 115 188 189 274 116  58
36
 217 155 259 108  18  59  89  92 144 228  98 287]
98 98
Находим для  16 96
16 96
[27  0 27 45 32 35 44  4  3 28 23 19 44 22 44 47]
[203 237 146 104  38  49  13 189 247  19 107  77 195 278  91 297  49
198
 198 148  46 155   8  37 106  79 239 178 231  85 116  33 268   7  96
192
 239   5   8 195 129   3  57  14 183 157 280 113   2 230 288 288 161
60
   1   6 205 231 217  97 277 264 169   5  27 222 221  55  81  49 287
44
 298 124 201 133  15  72 154  56 123  76  30 172  87 234 127 182 111
107
 141 279  56 266 262 277]
112 112
Находим для  18 108
18 108
[ 5 28 35  5  2 15 41  4 37 12 12 42 33 35 34  0 18 41]
[146 294  72 183 249 128  64 281  70 242  33  62 296 146 146  57  22
32
  92  73 279  94 248  38 192   7  11  18  83 219 117 216  97  11   6
72
 204 109 207  31 253   0 252 236 190 240  62 230 227  20 177 247 141
276
 237 201 160 112  38 103 292  59 255   3 155 274 288 204  91 115 232
109
 185 226 185  28  54   5 287 227  63  14  64  39  82 217 193 240  13
177
 211 102 287 192  62  11 211   0 188 117 260 140 199 200 165 191  23
250]
126 126
Находим для  20 120
```

20 120
[28 23 26 8 36 30 1 43 24 27 42 14 19 10 36 18 24 25 26 33]
[225 180 73 106 138 80 228 277 99   1 90 295   7 203 25 127 47
286
  38 264 287 18 133 98 146 149 100 77 59 47 121 121 218 173 119
296
 233 87 103 87 119 230 129 221 86 84 38 174 285 114 100 205 100
292
 162 210 147 227 226 210 265 184 19 53 294 63 73 264 67 52 175
100
 126   7 268   4 17 154 55 14 152 209 129 211 71 276 26 254 299
25
 226 70 290 218 107 55 254 54 37 272 24 36 60 132 171 161 290
139
 180 266 39 220 53 158 78 209 92 82   1 13]
140 140
Находим для 22 132
22 132
[33 0 46 1 34 48 21 12 7 8 7 37 34 17 17 38 37 5 15 13 26 35]
[290 47 194 81 13 229 213 185 238 75 174 155 71 61 213 149 144
149
  81 220 87 141 69 70 217 271 17 223 27 254 282 108 211 146 145
33
  79 93 22 292 92 204 12 34 299 14 89 254 167 131 182 282 239
278
  90 212 239 136 58 168 123 194 227 104 226 182 297 201 66 150 260
299
 157 198 283 110 36 207 279 159 299 212 285 242 53 271 255 53 111
248
  64 285 262 222 108 16 247 89 176 187 188 194 280 68 124 88 273
288
 171 259 89 242 193 167 240 204 122 85 89 12 126 192 210 49 80
222
 240 119 215 200 33 22]
154 154
Находим для 24 144
24 144
[46 5 43 4 13 28 47 6 0 15 4 0 41 6 44 31 19 36 13 7 19 29 42
15]
[199 163 67 278 239 222 260 286 266 209 28 175 45 19 142 141 237
223
  33 41 33 79 144 286 154 55 299 209 63 54 182   3 41 216 223
22
 165 24 125 215 106 147 284 220 32 64 182 267 261 83 287 290 10
88
 248 250 117 161 210 234 85 23 174   2 211 75 69 246 186 158 232
285
 214 65 161 43 191 267   2 224 113 149 294 89 180 191 74 254   2
112
  97 240 288 75 122 293 85 144 51 237 269 16 40 168 231 248 40

127
 274 298 287 100 187 136 295 205 292 216 256 76 12 43 41 25 74
266
 113 136 156 63 188   1 207 198 284 16 258 36 87 175 275 35 200
189]
168 168
Находим для 26 156
26 156
[11 29 31 43 18 29 47 11 21 1 9 18 46 17 48 43 20 11 2 16 33 24 45
21
 15 43]
[0 176 180   9 138 85 143 163 120 236 250 193 115 160 241 138 54
266
 193 75 144 134 18 175 116 109 291 47 35 291 159 51 133 188 166
141
 268 117 169 229 244 296 94 222   8 259 197 41 81 167 29 279 181
124
  39 207 24 259 236 154 115 77 80 63 182 176 57 152 175 126 275
238
 101 241 40 19 158 163 209 201 274 213 53 223   1 122 284 65 222
235
 283 266 178   3 143 291 10 237 293 130 258 188 107 121 214 170 157
296
 265 201 240 117 200 83 226 292 277 135 174 140 97 287 90 28 96
32
 182 131 163 168 297 67 71 280 239 74 226 111 106 134 50 239 121
99
 195 154 232 238 176 234 21 88 68 12 105 82]
182 182
Находим для 28 168
28 168
[1 40 14 12 4 18 37 6 44 0 43 1 19 11 28 44 10 21 41 32 17 14 30
40
 19 21 45 0]
[249 251 251 58 104 49 247   2 175 170 20 183 133 216 252 135 262
245
 132 120 22 100 111 258 232 54 289 41 98 248 188 106 237 231 70
169
   2 159 29 30 230 53 48 129 80 83 32 81 195 283   6 281 248
180
   8 102 263 290 230 100 264 56 220 221 42 205 96 41 244 179 226
276
 153 64 202 93 225 221 99 293   6 238 270 19 268 229 161 99 159
85
  24 154 276 77 157   1 215 218 278 225 91 130 255 63 135 112 61
66
 106 53 69 128 52 148 120 14 131 267 36 271 100 188 196 105 105
247
 255 253 41 288 73 251 11 228 10 55   7 222 125 209 258 35 37
174

  74 13 79 197 144 186 38 40 201 183 287 78 97 277 40 122 144
215
 247 165 102   4 66 223]
196 196
Находим для 30 180
30 180
[39 26 0 32 42 43 36 28 38 43 19 35 39 43 12 8 26 43 43 25 44 27 6
10
 31 40 12 7 1 42]
[267 188 83 149 113 37 127 31 273 121 281 100 67 34 37 264 269
176
  68 192 163 156 162 84 239 271 236 225 50   8 292 51 119 67 183
222
 159 86 192 270 169 225 259   0 56 78 279 119 115 111 188 270 117
119
 141 148 104 155 231   4 20 298 258 35 233 250 264 269 50 261 167
100
 246 139 203 46 247   9 100 22 141 53 198 230 167 78 289 36 234
32
 152 80 119 32 36 238 59 199 55 201 10 298 298 142 84 10 198
175
 103 288 145 270 41 129 171 277 60 168 286 88 40 94 117 17 294
74
  98 290 43 174 272 189   5 197 115 297 149 92 19 97 234 25 151
276
 115 173 113 32 297 168   2 118 273 72 296 199 16 166 133 82 218
209
  95 247 198 162 202 96 103 12 110 48 27 242 152 231 242 237 55
29]
210 210
Находим для 32 192
32 192
[4 37 10 8 7 41 32 45 36 28 11 13 0 42 3 25 36 0 2 24 14 21 8
44
 46 2 47 0 48 41 13 0]
[251 30   9 176 206 294 185 86 253 85 35 90 245 155 216 205 290
262
  67   1 279 216   4 186 59 55 93 228 17 294 71 226 71 263 287
54
 145 207 38 20 290 223 199 170 231 53   8 27 56 263 275 137 285
195
   0 218 128 250 170 247 206 24 107 85   2 168 58 238 208 154 171
67
 209 152 295 68 175 68 194 207 11 285 87 85 248 153 105 196 15
202
 150 115 77 211 192 141 216 256 253 200 279 206 282 219 263 103 117
58
 137 179 138 292 15 139 237 208 252 251 221 127 82 265 76 275 162
276
 197 92 252 106 88 16 211 198 121 153 62 61 31 186 194 150 220

```
120
 180 204  86   6 222  14  42 115 247 139 103 275 163 141 232  35 178
107
 276 192  71 281 161  70  54 216 144  58 239 249 103 180 241 226 127
160
  61  13 286 221 218 295  47   7  99 153 228 264]
224 224
Находим для  34 204
34 204
[42 46 23 11 35 28 40 41 22  0 44 48  5 17 20 15  0 16  5 33 41 34 22
1
 35 41 31 14 36  1  1 13  9  7]
[155 203 261  94  41  86 174  13 221  67 167 159  89 216  17  28 113
34
  78 116 143  73  41 212 269 144 216  94 151 273 128 162 107 276 290
253
  51 122 182 132 196 108  96 105  34 224 236 100  16  27  29  72 232
254
 190 123 109  91  73 102 237 209 195 265 181 181  91 200  76 197  61
215
   1 158  72  70 260  41 286 189 256 169 162 140  61 179  23  78 119
235
  15  38 261  68 139 113  87 144  31 107 176   4  58 136 136  66 114
249
 111  87  85  68 245  92  13  12 164  41  78  81  79  26 295 232  30
151
  87 263 290 290 101  76 214  33  30 257 171 285 139  15 198 195 226
80
 110 227  45 254 271 288 160 132 149 285 122  22 246  96  76 128 296
172
   2 249 202 245 231 172  90 268 188 197 180  44  44  64  56   8   3
292
 265 142  67 257 234 122 282  99 229  47  65  84  78  32  87  75 145
294
 250 125 237 151 161 200]
238 238
Находим для  36 216
36 216
[18  9 28 24 29  2 18  1  3  1 30 17 32 28  2 43  1  3 10 30 42 32 31
1
  4 43 11 10 48  8 30 38  0 14  1 37]
[208 158 230 142 148 152 151 256 282  15  68 181 114  99  14  34 278
30
 275 236   7  90 261 182 273 248 127  74 142 288  37 130 123 170 100
87
  59  60 133 285 128 207  79  24  41 129 278 299 179  87 280 184 287
118
 246  67 248   2 237 194  68  50 147 285  37 199  81 257 161 292  88
106
 127  98 175 199  51 226  31 240   1 108  65  34 264  99 225 168 226
```

```
235
 192 290 289 272  85 118 272 117 218 143  65 131 190  48 179 122 267
291
 168  76 128 169   0  20 298 236 278 182  66 177 233   5 160  29 214
230
  78 236 116  79 294 139 267 243 284  59 254 230 287 177 276 226  85
226
 180 251 160 147  69 286 292 155  31   4  72 110 293  48  29 108 291
234
 179   9 289 276 204  55 268 177 211 150 294 299 130 270 208 151 203
296
 127 292 255 225 237  54  75 214 150 185 257   3 281 235   9 110 146
273
 218 145 218 140  30 106 127 258 172 211   8  37 130 224 291 282 170
219]
252 252
Находим для  38 228
38 228
[43 43 38  3 42  0 14 28  2 43 32 30 45 20 41  0 13 15 18 17 32 26 28
31
  7 42 10 10 35 29  8 15 27 48 36 46 15 35]
[167  74  88  57 231   6  50 219  64  51 234 158 170 108 169  87 199
258
  35 135  74   9 185  12  84 275 225 131  57  32  42  24 138 212 105
235
 176 294 151 101 226  46 236 183 199 116 231  26 132 163 171  36 234
94
 127 287  11 201 283 105  59 226 131 278  89  78 235 139  73  80 267
224
 276 142 178 201 219 197 225  72  88 186  18 165 165 148 198  46 295
236
 150 282 215 248 125 293 142  42 112 197 297 103 254  87 110  28 228
150
 178  21 263 144  59 125 141 131 267 262 141 169 172 209  70 210 261
227
 240 269  15 136 250 186 262 225 210 262 141 250  94 280 299 102 212
2
  55 241 168  13 153 122  43  89  91   5  30 154 272 182 139 174 126
198
 239 154 279 240 106 172 270  78 184 219 279 238   2 151 109 233 133
188
 101  16  19 235 161 273 204 261 251  68 172 244  63 224 104 140 214
279
 120 246 169 158 168 204 114 149  83 294 193 143  28 117 115   2 135
132
  68 180 171  25 184 255 138 172 147 245  44  89]
266 266
Находим для  40 240
40 240
[19 31 26 49 30 16 38 20 47 30 27 35  6 35  3 10 46 29 34 29 28 47 10
```

```
9
 11  8 38 36 27 14 31 45 39 11 35 34 12 22 42 33]
[285 193 249 274 120 298 260 174   4  80 258 294  30 226 125 219 145
261
 121 291 235  67 143 289 229  39  41  79 107  10 221 106 216 120  50
67
 266 260  74 256 118 279 222 162 155  13 153 224 160 215 102 221  26
215
 179 198 234 122 122  86 177 148 274  78 224 168 293 154 268  99   2
281
 136  67  52  62 296 125 187 200  22  51 245 119 295 126  38  99  80
130
  69 200 263 133 239  11 197 241 160  64 143 251 239 202 163 279  96
295
 194  85  11 208 143 209 267  28 120 290 122  73  98 290 129 288  38
61
 197  30 256  37 285 194 156 234 129 171 283 269 219 129 164 287 298
18
 145  96  26 179 257  88  74 272 296 234 261  14 156  40 150  77  13
206
  18  13 123 211 269 169 213  44 110 234 153 290  36 132 293 290 213
23
 288 271 277 100 132  20 135 151  14   6 165 129 254 272 201 158 102
115
 115  49  74 228 289 158  70  18  99 261  73 234  37 138   0   8 162
238
  81  14  89 158  14 118  60 295 187 208  68 205 143  65 147  56   6
45
 144 119   3 156 224 117]
280 280
Находим для  42 252
42 252
[46 42 20 29 19 41  0  3  3 37  5 21  7  5  1 30  3 36 36 43  4 33 21
23
 36 48  9 47 15 37 34  0 39 15 24 24 32  1 48 43  0 39]
[105  77 156  31 221  57 176 229 295 110 284 261 239 125 197 141  71
250
 227 185 229  68  31  71 224 282 224  97 203  40 135  95   5 140  17
259
 270 284 178 154 198  23 134  56  98 187 236 142  91 100  91 282 195
92
 197 218 146   8  37 201  10  75  96 240 187  49   3 213 249  28 230
268
  58 112 112 178 155 282  43 128  90 281 246 174 142  89  47 163 285
248
 239 226  80 164 116  83 223 235 297 244 188 292 102 230 271 287  56
154
 141  59  29 216 161 228  78 195  59 269 216 106  65  20 196   7  25
126
  92 149 296  24 131   6 145  38  70 268 138 166  97 167 159 199 174
```

```
125
  60 220  69 292 207 193 120 163  10 236 284 147 295  65 131 111 234
62
 266 150 145  90   6  27 167 203 214 217 217  45 264  35 234 175  75
230
 116  66  23 211 165 204 107 108 279  81 147 220  70  67 104 100 193
134
 165 197 249  81  92  54 147 148 197 285   5 203  58  12  31  32 232
252
 273 143 212 117  87  94 296  13 266 126  21 223 269 208 206  67 112
272
 155 279 258 135  48 285  25 255  82  94 156 205 239  11 205 298 203
37]
294 294
Находим для  44 264
44 264
[35  8 40 22 15 10 44 11 11 33 41 49 45 21 25 41 22  4 30 36  8 25  3
1
 14 39 13  9 21 32 19  7 41 42 18 37 25 48 44 20 27 16 37 38]
[  7 204  94 247 261 203 266 203 252  94 116  15 262 275  99 217 187
147
 243 185   1 150 190 285 244  61 206   5 168 113  11 216 292 100 121
148
 286 188 178 105 150 149  57 232 266 137  19 297 231  35 255 294 187
203
 205 282  79 115  65 184  43 151 103  39  90   1 283  61 192 163 159
238
  99 149 294  68 225 250 253  79 284 173 198 243 233 172  71 177 101
194
  41 180 105 186 158  79  13  58 296 202 145  89 202 257 208  24  50
123
   0  72 169 133  39 188  66 179 186 105 137 185 214  93  22  42 210
142
 288  74 142 216 250 268 142 117  43  73 155  62 169 230 296  48 184
20
  19  71  37  19  49 101 209  21  88 233 230  51  27  26 156 125 183
224
   8 246 113 225  39 173 223  55  96 196  10 299  18 149 123 272 285
223
 178 115 292  47 252  30 118  28 171 185  37  86  96 211   1 105 208
59
 138 163 173 194 191 115 156 113  97 248  66  12 220 263  96  79  11
213
 153  28 219 218 244 183 291  24  85 194 286 140 294 257 299 118 233
103
  83 108  24 207 119  65 278 223  81 240  16 146 237 263 130 288 295
257
 131  81 173 118 143 276 140 155  52  70 136  16]
308 308
Находим для  46 276
```

```
46 276
[13 24 30 12 42 20 30 36 40 10  8  9 38 39 24 42 38 21  2 27  0 22 28
46
 22  3 15 27 30 10 27 43 40  2  4 30 10 40 42  1 28  3 40 32 41 44]
[ 14 248 271 161 285 273 178  71  65 280 179  21 156   9  41  58  18
267
  26 298 177  52  32  61  92 163 269   9  66 252  65  23 288 271 216
88
 233 101  91 181 213 250 117   9 205 181 139 215 222  83 234 197 132
120
 264 177 299  37 271  68  90  21 162 195 117 164  55 155 133  83  36
193
 266 287  78 135 256  43  18 185  67 170 257 224 144 241 278  56  46
66
 260   0 293 111 203  20 128  83 256 249   1 174   1 251 124 250 163
125
  13   5  97 112 225 143  26 265  20 131 165 251   1 246 179 146 214
14
 227 123  56 288   5 139 270   3  90 167 258 206 191  40 137 184 106
101
  69  33 259  76  21 133 145 262 208 223 121 271 113 281 114  50 133
86
  60 119 229 227  47 186  94 187  56 132 183 183 296  37 240 244  17
68
 169 198  83  16 118  77 254 199  11 179  46  33  10 272 163  42 115
154
 194 190 280 273 192  10 115 256 199 167  50 129 268 128 182 236 263
0
 113 253 264 198 192 177 278 299 236 276 220 248 146 290 206 101 132
47
 196  53  27 271 250 118 226 167 156  13 147 193 226  19  10 169   3
261
 105 243  43  67 125  42   6 271 288 156 210 132 236 174 106 287  78
145
   8 284 185 241 196  89]
322 322
Находим для  48 288
48 288
[18 31 19 36 31 25 46 41 30 21 35  9 39 47 43 41  7  6  3 15 14 47 14
48
  3 25 27 27  6 29  7 32  8 41 29  8  9 32 34 15 19  2 19  7 42 19 31
16]
[289  66 239  66 299 179 151 172 242 257 247 135 205 256 150 280 288
193
  29 112 139  81 281  87  39  70  45 231  79   5  18 250 266 190  29
255
  95 234  42  72 193 171 250 298  15  32 160  28  71 258 213 117 144
45
 294 231 242  80  33 103  12  67 249 266 142 197  41 248 294 257 236
174
```

 266 135 71 19 198 172 105 148 10 180 112 86 68 281 153 193 247
222
  16 274 163 187 236 78 43 128 31 81 144 53 299 28 71 249 80
187
  97 242 148 280 128 299 101 133 138 218 275 96 94 82 29 164 187
26
 212 231 17   7 186 140 13 22 143 144 172 141 174 217 157 257 190
74
 156 202 32 45   2 27 136 159 243 70 250 143 245 34   4 102 122
156
 157 134 117 26 16   5 124 107 171 205 230 91 100 136 80 130 237
250
 203 100 29 193   4 161   9 133 54 218 155 200 256 153 233 21 102
70
  43 274 215 60 264 106 171 230 247 96 206 130 101 219 62 272 227
64
 167   4 62 77 174 178 203 84   0 69 69 275 161 107 132 136 226
86
  53 105 56 291 102 246 279 191 194 233 188 255 190 37 177 240 77
178
  92 23 238 237 97 296 238 271 40 17 261 297 211 121 220 30 112
284
 204 152 203 205 137 113   4 273 195 99 66 241 183 263 184 193 138
94]
336 336
Находим для 50 300
50 300
[12 11 38 48 23 43 41 30 2 31 29 4 17 41 18 25 14 20 42 10 1 39 31
43
 43 20 17 19 44 1 7 0 33 4 9 3 13 36 7 26 21 39 9 22 6 28 2
38
 10 29]
[135   8 86 122 187 189 162 196 140 27 255 68 263 170 213 71 105
199
 235 204 91 295 194 215 279 76 213 125 227 256 255 12 104 250   4
0
  50 84 133 215 157 133 169 28 219 264 219 203 121 289 160 272 173
18
  33 115 233 213 76 211 216 288 90 194 32 187 55 114 30 280 220
13
 157 127 99 107 248 45 249 94 159 65 258 174 189 165 234 109 285
172
 147 66 225 173 289 38 178 273 49 296 131 232 16 183 160 86 164
169
 169 225 281 182 188 192 30 144 135 281 250 156 21 106 297 288 106
278
  23 165 70 173 265 159 190 15 32   7 104 256 233 241 73 199 273
155
 273 69 38 62 119 97 17 141 188 44 211 149 129 64 176 233 277
275

```
 118 281 130 133 282 181 122  32  30  15 180 131 156 198 276 119  62
195
  71  20  29  35 297 151 107 282 158 286 201  85 186  29 135 108 155
299
 263 154  31  38 215 151  62  50  65  76 127  44   4 250  80  66 128
234
  68 205  20   6 291  59  10 108 299  72 114 267   4  65 297  72 234
138
 132  55  27   3 252  67  21 181  58  52 159  57 256 253  67 135 268
277
 101 183 195 258 240 221 123  65 191 144 115 183 295 174  91 201 248
128
  63  73 147  98  60 244  34 111 252 203  30 294  98 279 149 272 101
44
  35   0 101 108 228 222  45 276 145 253 156 178]
350 350
```

```
for i in range(len(res_number)):
    print('N = ', res_number[i], 'AUC_roc = ', res_roc_auc[i], 'AUC_pr
= ', res_pr_auc[i])
```

```
N =  14 AUC_roc =  1.0 AUC_pr =  1.0
N =  28 AUC_roc =  0.9583333333333334 AUC_pr =  0.8110119047619048
N =  42 AUC_roc =  0.9490740740740741 AUC_pr =  0.8249158249158248
N =  56 AUC_roc =  0.9270833333333334 AUC_pr =  0.8359035633484163
N =  70 AUC_roc =  0.9883333333333334 AUC_pr =  0.9431150793650794
N =  84 AUC_roc =  0.9305555555555556 AUC_pr =  0.7587742899301868
N =  98 AUC_roc =  0.9345238095238095 AUC_pr =  0.8112766059177312
N =  112 AUC_roc =  0.9505208333333334 AUC_pr =  0.7690231895909501
N =  126 AUC_roc =  0.9614197530864197 AUC_pr =  0.9183107040341085
N =  140 AUC_roc =  0.9579166666666666 AUC_pr =  0.8655275535958681
N =  154 AUC_roc =  0.9724517906336088 AUC_pr =  0.899070997571901
N =  168 AUC_roc =  0.9560185185185186 AUC_pr =  0.8673632360806022
N =  182 AUC_roc =  0.9349112426035503 AUC_pr =  0.8122622019682073
N =  196 AUC_roc =  0.9696003401360545 AUC_pr =  0.8833157379644636
N =  210 AUC_roc =  0.9866666666666666 AUC_pr =  0.9405669221709113
N =  224 AUC_roc =  0.9549153645833334 AUC_pr =  0.8661270833879822
N =  238 AUC_roc =  0.9629469434832757 AUC_pr =  0.8763182730837359
N =  252 AUC_roc =  0.9502314814814814 AUC_pr =  0.8091353324260337
N =  266 AUC_roc =  0.9818790397045245 AUC_pr =  0.9245918316138543
N =  280 AUC_roc =  0.9408333333333332 AUC_pr =  0.7772522388340942
N =  294 AUC_roc =  0.9444444444444444 AUC_pr =  0.831844142412522
N =  308 AUC_roc =  0.9566115702479339 AUC_pr =  0.867208067404442
N =  322 AUC_roc =  0.9639256458727158 AUC_pr =  0.8814291497903424
N =  336 AUC_roc =  0.9559461805555556 AUC_pr =  0.8403726506958965
N =  350 AUC_roc =  0.9392666666666667 AUC_pr =  0.7891118376144272
```

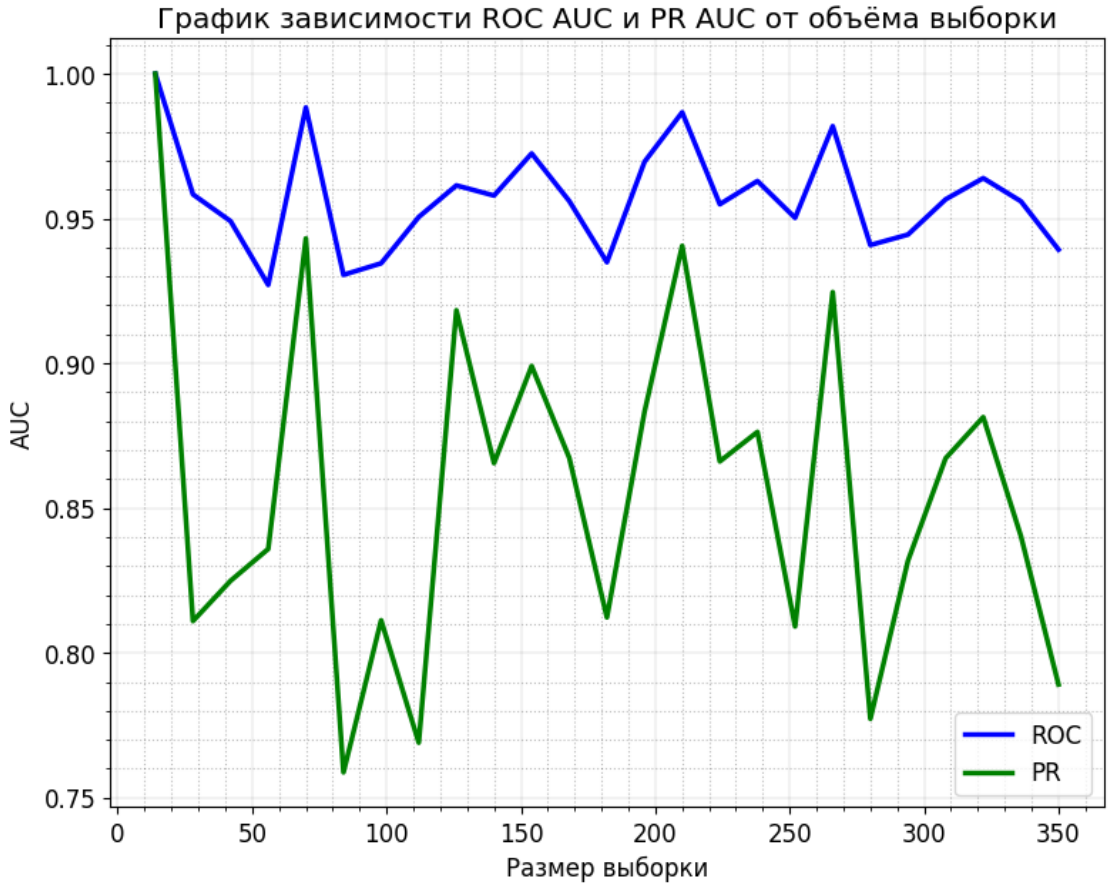
```
print(len(res_number), len(res_roc_auc), len(res_pr_auc))
```

```
25 25 25
```

```
plt.figure(figsize=(9, 7))
plt.plot(res_number, res_roc_auc, color='blue',linewidth = 2.5,
label='ROC')
plt.plot(res_number, res_pr_auc, color='green',linewidth = 2.5,
label='PR')

plt.xlabel('Размер выборки')
plt.ylabel('AUC')
plt.title('График зависимости ROC AUC и PR AUC от объёма выборки')
plt.legend(loc="lower right")
plt.minorticks_on()
plt.grid(which='major',
        color = 'k', alpha = 0.25,
        linewidth = 0.3)

plt.grid(which='minor',
        color = 'k', alpha = 0.25,
        linestyle = ':')
plt.rcParams['font.size'] = '12'
plt.show()
```

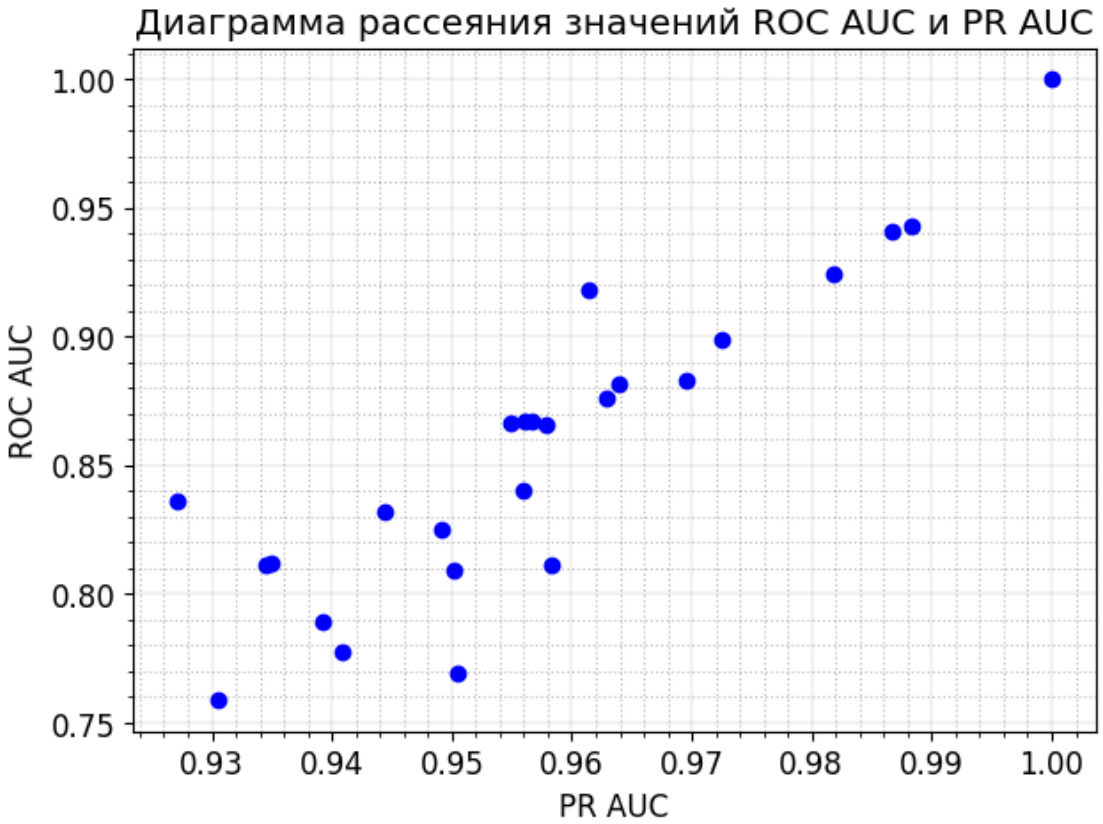



```
plt.figure(figsize=(7,5 ))
```

```
plt.scatter(res_roc_auc, res_pr_auc, color='blue',linewidth = 1)

plt.xlabel('PR AUC')
plt.ylabel('ROC AUC')
plt.title('Диаграмма рассеяния значений ROC AUC и PR AUC')
#plt.legend(loc="lower right")
plt.minorticks_on()
plt.grid(which='major',
        color = 'k', alpha = 0.25,
        linewidth = 0.3)

plt.grid(which='minor',
        color = 'k', alpha = 0.25,
        linestyle = ':')
plt.rcParams['font.size'] = '12'
plt.show()
```




```
np.corrcoef(res_roc_auc, res_pr_auc)

array([[1.        , 0.91564973],
       [0.91564973, 1.        ]])
```

Результаты исследований: В данной работе проводилось исследование показателей качества бинарной классификации, связанных с разделимостью классов. Исследование проводилось с помощью

заданного набора данных и результата бинарной классификации этих данных.

С помощью критерия Манна-Уитни было установлено, что классификационные очки положительного класс превышают очки отрицательного класса. Разница значения нормированного значения статистики Манна-Уитни и ROC AUC составляет менее 2%.

Было показано, что имеет место сильная корреляция величин AUC для ROC и PR кривых.

ROC и PR кривые являются полезным инструментом для сравнения различных моделей классификаторов. Площадь под графиками этих кривых позволяет оценить ту или иную модель в целом, не привязываясь к конкретному порогу.

ROC-кривые более целесообразно использовать, когда наблюдения сбалансированы между каждым классом, тогда как PR-кривые подходят для несбалансированных наборов данных.