# **Задача классификации**

Для пример сгенерируем случайные данные и постараемся разобраться, как решать задачу классификации данных:

import matplotlib

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

TEXT\_COLOR = 'black'

matplotlib.rcParams['figure.figsize'] = (15, 10)

matplotlib.rcParams['text.color'] = 'black'

matplotlib.rcParams['font.size'] = 14

matplotlib.rcParams['axes.labelcolor'] = TEXT\_COLOR

matplotlib.rcParams['xtick.color'] = TEXT\_COLOR

matplotlib.rcParams['ytick.color'] = TEXT\_COLOR

RANDOM\_STATE = 0

np.random.seed(RANDOM\_STATE)

from sklearn.datasets import make\_classification

X\_data, y\_data = make\_classification(

n\_samples=100,

n\_features=2,

n\_redundant=0,

n\_informative=2,

n\_clusters\_per\_class=2,

random\_state=RANDOM\_STATE,

shift=[4, 4]

)

X\_new = np.array([[4.2, 4.2]])

pnts\_scatter = plt.scatter(X\_data[:, 0], X\_data[:, 1], marker='o', c=y\_data, s=50, edgecolor='k')

plt.scatter(X\_new[:, 0], X\_new[:, 1], marker='x', s=100, edgecolor='k', c='green')

plt.xlabel('$x\_1$')

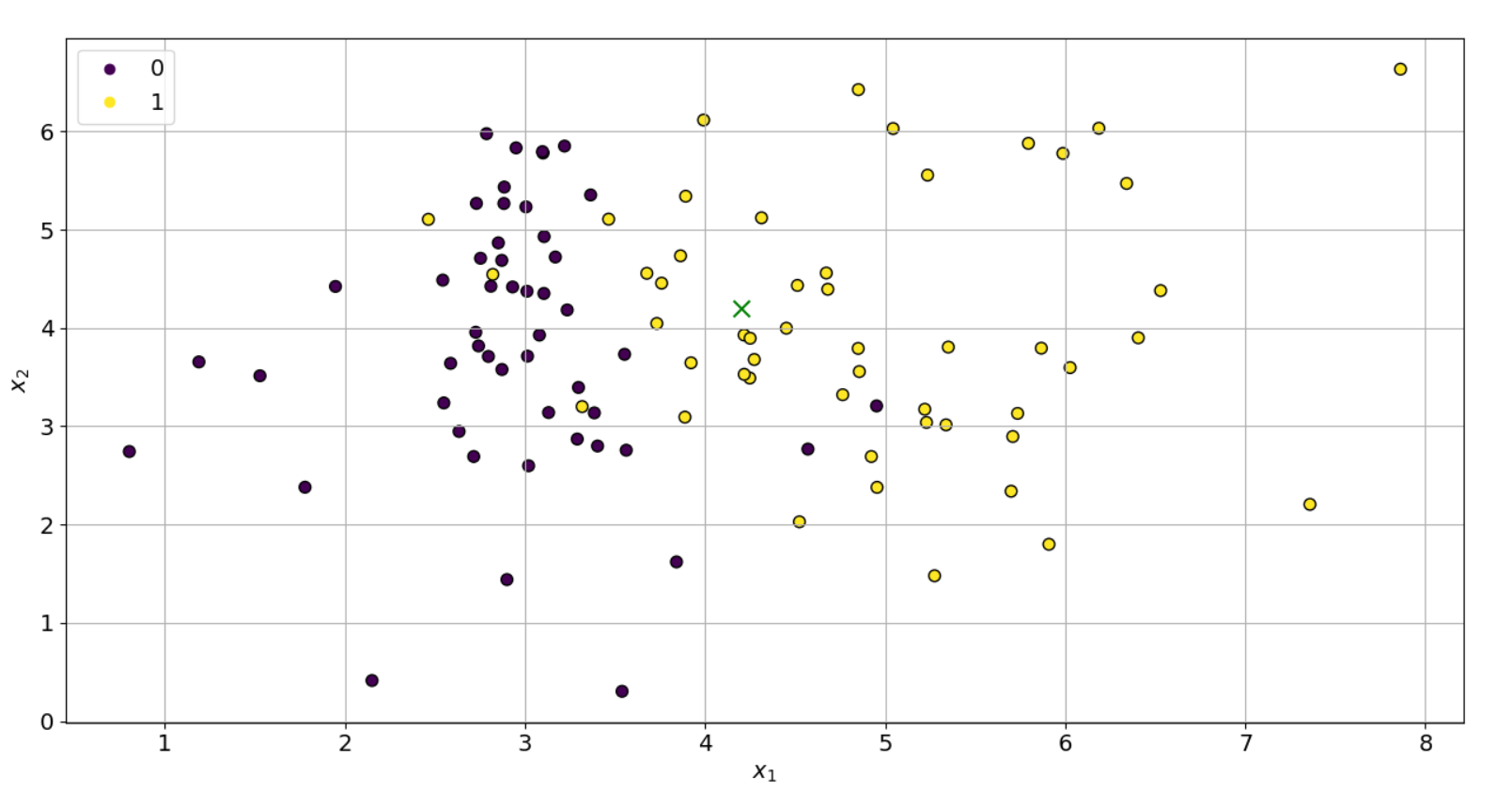
plt.ylabel('$x\_2$')

plt.grid(True)

plt.legend(handles=pnts\_scatter.legend\_elements()[0], labels=['0', '1'])

plt.show()

Результат



Также взглянем на данные X\_data и разметку y\_data, которые мы имеем:

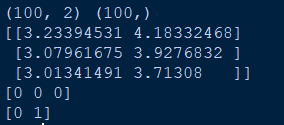
print(X\_data.shape, y\_data.shape)

print(X\_data[:3])

print(y\_data[:3])

print(np.unique(y\_data))

Результат



**Логистическая регрессия**

Для отображения функции сигмоиды реализуем недостающую функцию sigmoid():

def sigmoid(z):

sgm=1/(1+np.exp(-z))

return sgm

X = np.linspace(-10, 10, 100)

Y=sigmoid(X)

plt.plot(X, Y)

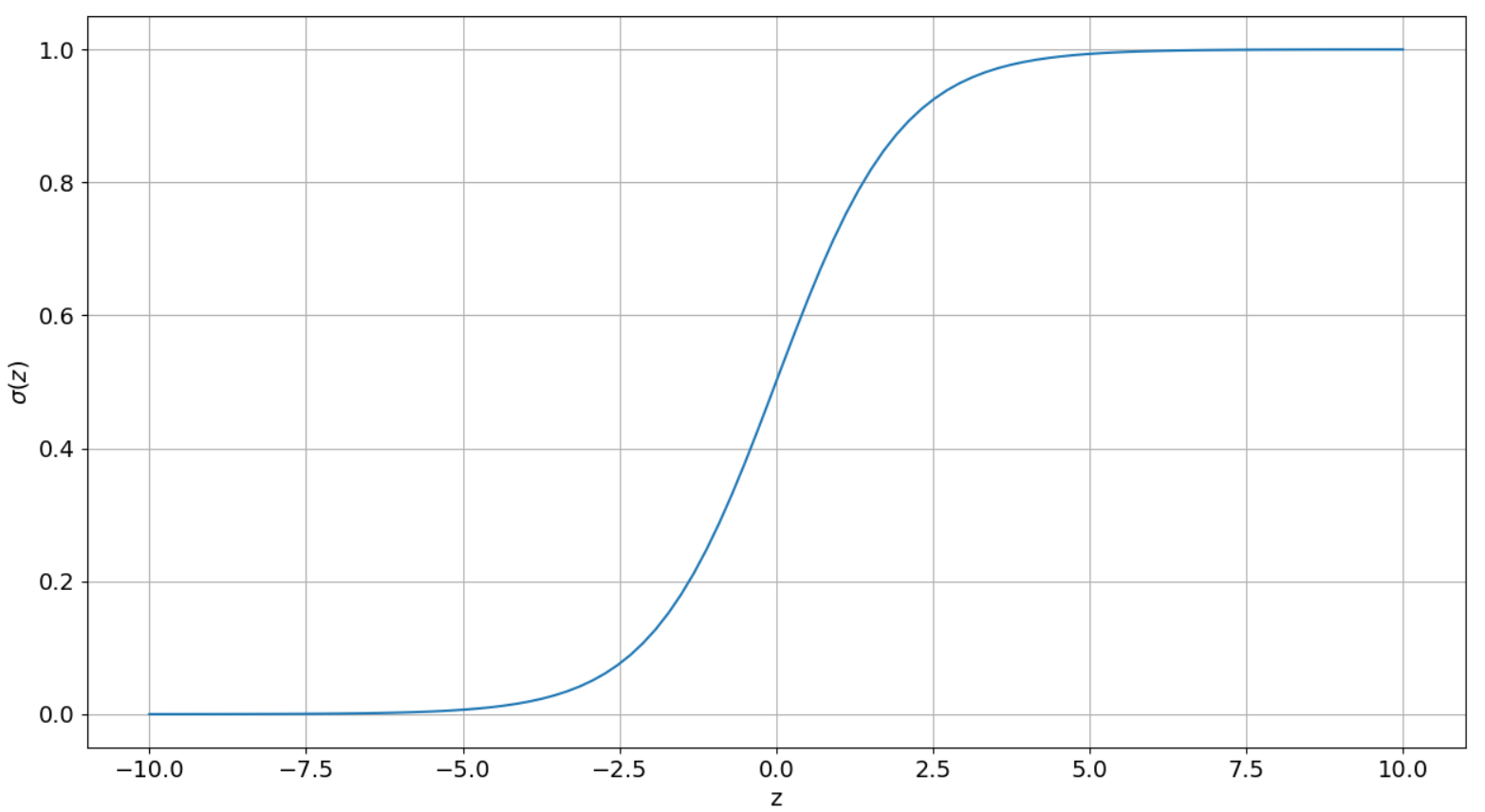
plt.xlabel('z')

plt.ylabel('$\sigma(z)$')

plt.grid()

plt.show()

Результат



# **Функция предсказания**

Начало реализации модели происходит с реализации функции предсказания. Для начала вновь вспомним и реализуем функцию предсказания линейной регрессии:

# функция линейной регрессии

def predict\_linreg(X, W):

X=np.concatenate((np.ones((X.shape[0],1),int),X),axis=1)

z=X@W

return z

# Проверим на паре тестов

W = np.array([1, 2, 3])

X = np.array([

[3, 4],

[2, 3],

[0, 1]

])

y\_true = np.array([19, 14, 4])

y\_pred = predict\_linreg(X, W)

assert np.all(y\_true == y\_pred)

После этого, напишем реализацию функции предсказания степени уверенности:

def predict\_proba(X, W):

z=predict\_linreg(X,W)

y=sigmoid(z)

return y

# Проверим на паре тестов

W = np.array([1, 2, 3])

X = np.array([

[-3, 4],

[2, -3],

[0, -1]

])

y\_true = np.array([0.999088, 0.01798, 0.1192])

y\_pred = predict\_proba(X, W)

# Проверка сделана через близость значений из-за округлений

assert np.all(np.abs(y\_true-y\_pred) < 1e-5)

# Для примера отобразим порог вместе с функцией сигмоиды

X = np.linspace(-10, 10, 100)

Y = sigmoid(X)

Y\_thrsh = np.full\_like(X, 0.5)

plt.plot(X, Y)

plt.plot(X, Y\_thrsh, 'r--', label='threshold = 0.5')

plt.xlabel('z')

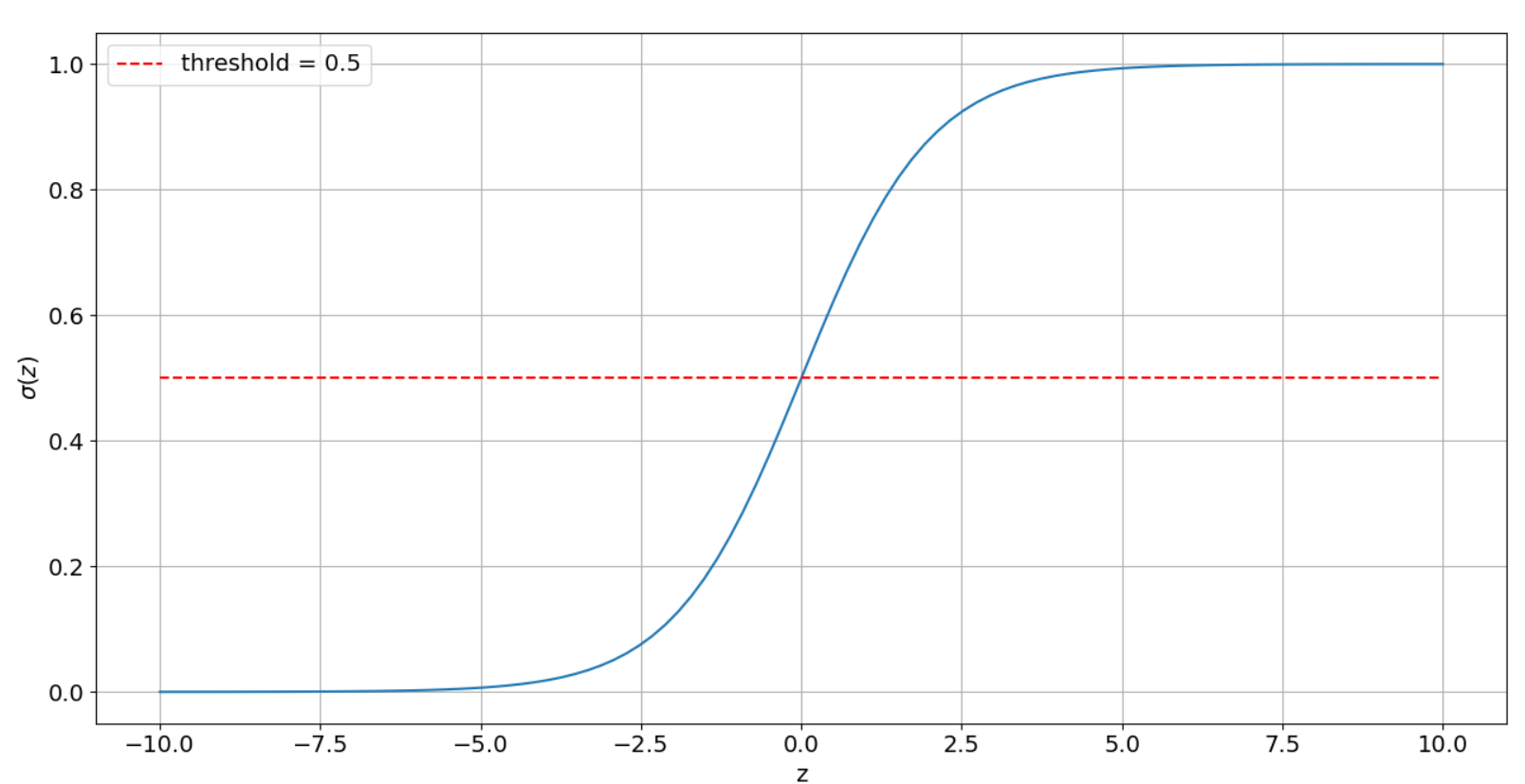
plt.ylabel('$\sigma(z)$')

plt.legend()

plt.grid()

plt.show()

Результат



# Функция определения класса

def predict(X, W, threshold):

y\_cls=[]

z=predict\_linreg(X,W)

y=sigmoid(z)

a=y>threshold

y\_cls=a.astype(int)

return y\_cls

# Проверка функции определения класса

W = np.array([1, 2, 3])

X = np.array([

[-3, 4],

[2, -3],

[0, -1]

])

y\_true = np.array([1, 0, 0])

y\_pred = predict(X, W, 0.5)

assert np.all(y\_true == y\_pred)

# **Визуализация предсказаний**

Для случая классификации с двумя признаками полезно для понимания провести визуализацию границы решения (decision boundary) модели:

def plot\_2d\_decision\_boundary(X, W, y\_true):

x1\_vals = np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 100)

x2\_vals = np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 100)

xx, yy = np.meshgrid(x1\_vals, x2\_vals)

y\_pred = predict\_proba(np.c\_[xx.ravel(), yy.ravel()], W)

y\_pred = y\_pred.reshape(xx.shape)

plt.contourf(xx, yy, y\_pred)

pnts\_scatter = plt.scatter(X[:, 0], X[:, 1], c=y\_true, s=30, edgecolor='k')

plt.xlabel("$x\_1$")

plt.ylabel("$x\_2$")

plt.grid(True)

plt.legend(handles=pnts\_scatter.legend\_elements()[0], labels=['0', '1'])

plt.show()

# Выберем случайные веса и установим переменные

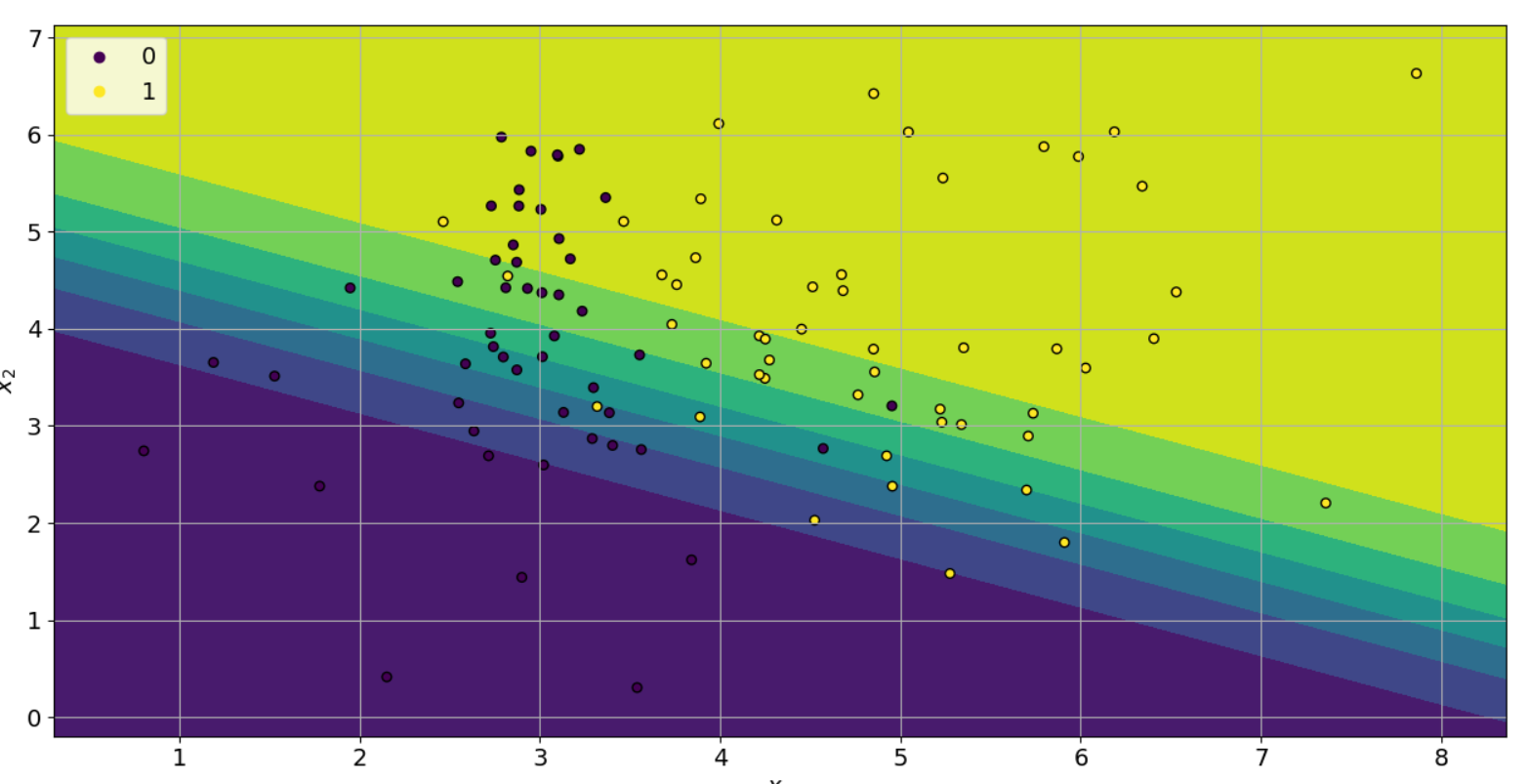
rand\_W = [-10, 1, 2]

X = X\_data

y\_true = y\_data

plot\_2d\_decision\_boundary(X\_data, rand\_W, y\_data)

Результат



## **Задание**

Замените функцию predict\_proba() на predict() (создав новую функцию отображения), отобразите данные и постарайтесь описать, почему изменился результат отображения:

#Заменим predict\_proba() на predict()

def plot\_2d\_decision\_boundary1(X, W, y\_true):

x1\_vals = np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 100)

x2\_vals = np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 100)

xx, yy = np.meshgrid(x1\_vals, x2\_vals)

y\_pred = predict(np.c\_[xx.ravel(), yy.ravel()], W, 0.5)

y\_pred = y\_pred.reshape(xx.shape)

plt.contourf(xx, yy, y\_pred)

pnts\_scatter = plt.scatter(X[:, 0], X[:, 1], c=y\_true, s=30, edgecolor='k')

plt.xlabel("$x\_1$")

plt.ylabel("$x\_2$")

plt.grid(True)

plt.legend(handles=pnts\_scatter.legend\_elements()[0], labels=['0', '1'])

plt.show()

# Выберем случайные веса и установим переменные

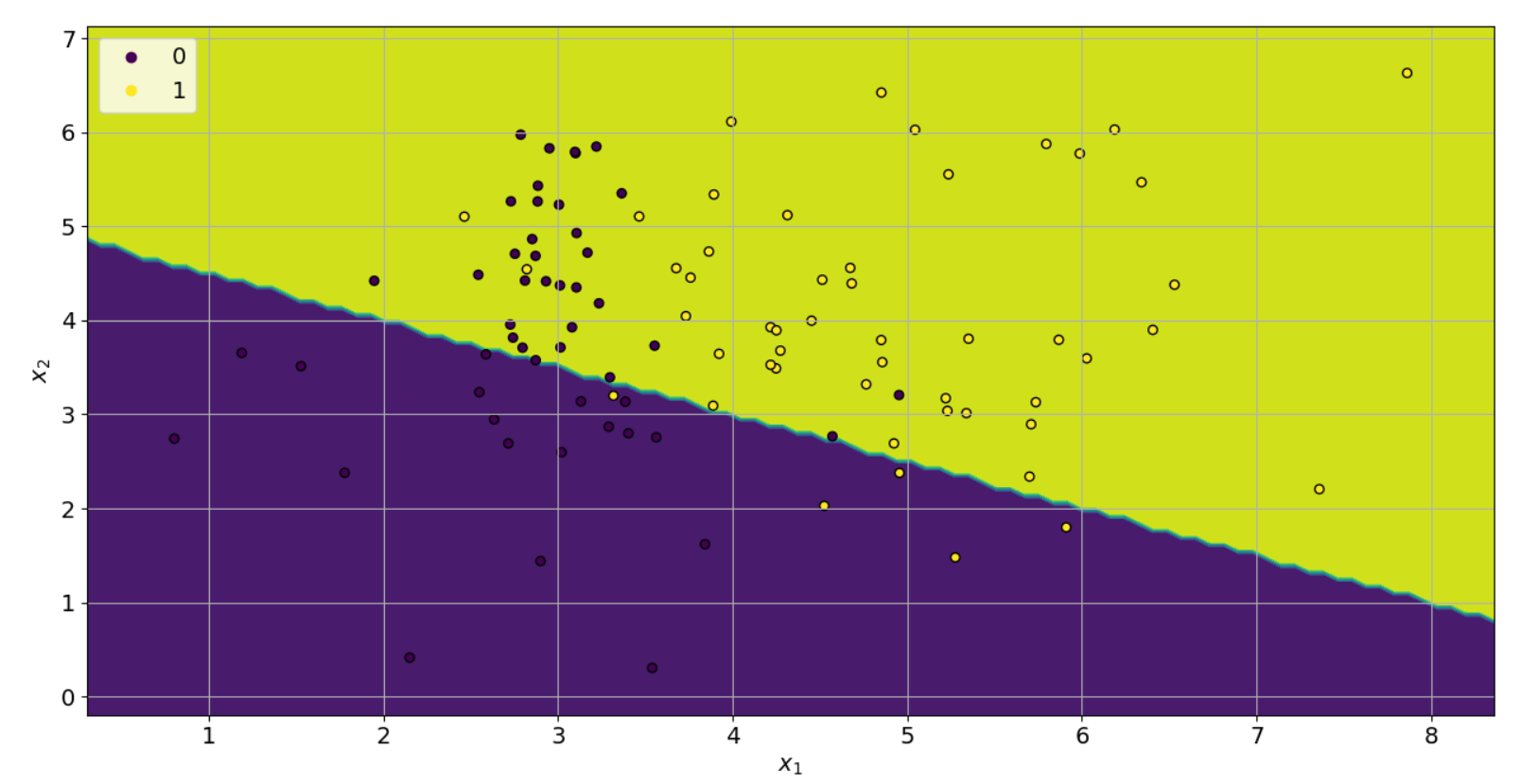
rand\_W = [-10, 1, 2]

X = X\_data

y\_true = y\_data

plot\_2d\_decision\_boundary1(X\_data, rand\_W, y\_data)

Результат



**Метрики**  
Расчет данных показателей заключается в сравнении вектора присвоенных классов и вектора размеченных классов.

# Функция получения показателей

def get\_truefalse\_metrics(y\_true, y\_pred):

TP=np.sum((y\_true==1)&(y\_pred==1))

TN=np.sum((y\_true==0)&(y\_pred==0))

FP=np.sum((y\_true==0)&(y\_pred==1))

FN=np.sum((y\_true==1)&(y\_pred==0))

return TP, TN, FP, FN

#Проверка

y\_true = np.array([0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1])

y\_pred = np.array([0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0])

TP, TN, FP, FN = get\_truefalse\_metrics(y\_true, y\_pred)

assert np.all([TP == 3, TN == 4, FP == 2, FN == 5])

## **Матрица ошибок**

Для визуализации матрицы полезно построить 2D массив и разместить в нем элементы:

# Матрица ошибок

def get\_confusion\_matrix(y\_true, y\_pred):

TP, TN, FP, FN = get\_truefalse\_metrics(y\_true, y\_pred)

conf\_matrix=([TN,FP],[FN,TP])

return conf\_matrix

# Небольшой тест

y\_true = np.array([0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1])

y\_pred = np.array([0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0])

conf\_matrix = get\_confusion\_matrix(y\_true, y\_pred)

assert np.all(conf\_matrix == np.array([[4, 2], [5, 3]]))

def show\_confusion\_matrix(conf\_matrix):

ax = sns.heatmap(conf\_matrix, annot=True)

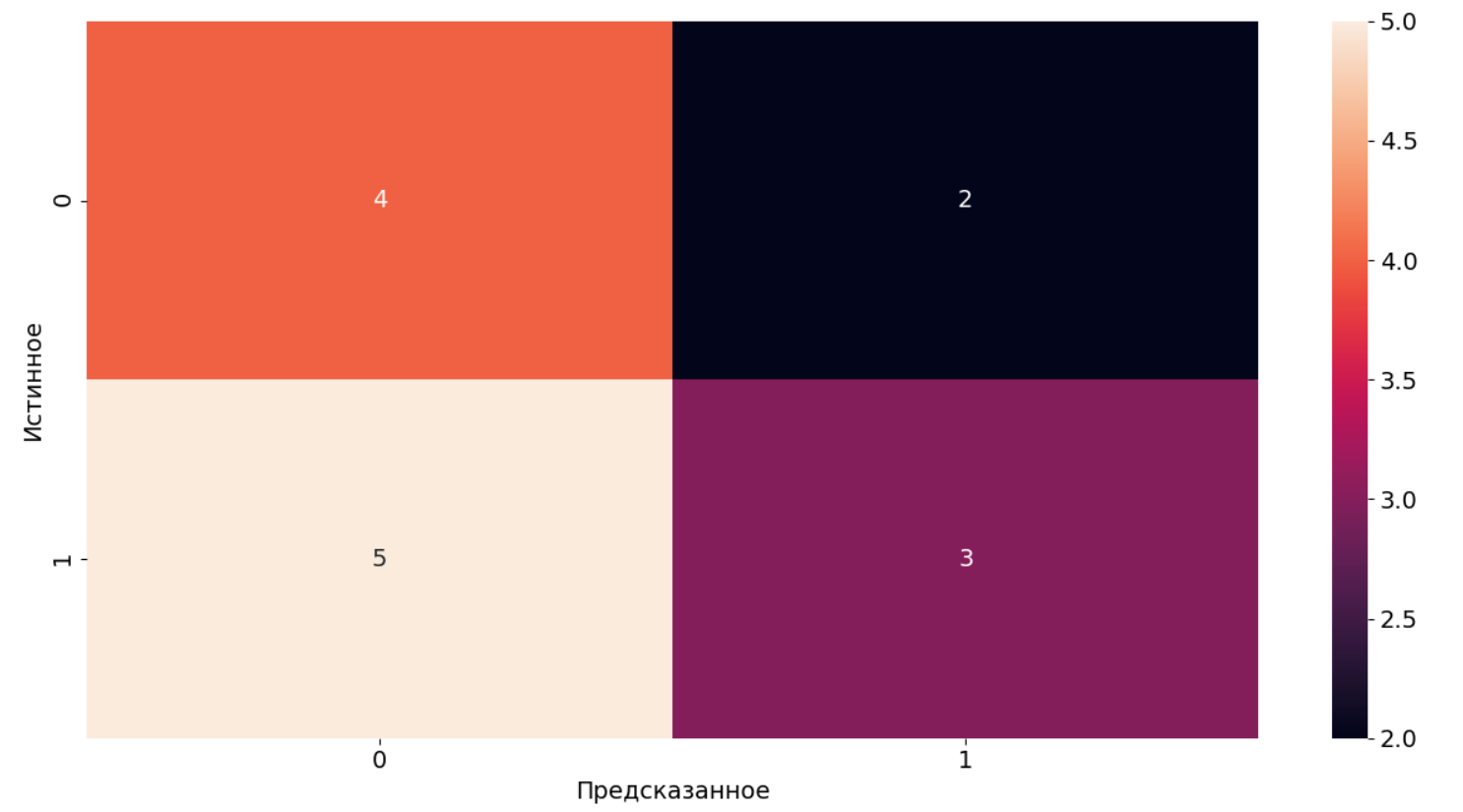
plt.xlabel('Предсказанное')

plt.ylabel('Истинное')

plt.show()

show\_confusion\_matrix(conf\_matrix)

Результат



## 

## **Задание**

Произведите предсказание классов на данных X\_data и отобразите результаты предсказания в виде матрицы ошибок.

rand\_W = [-10, 1, 2]

X = X\_data

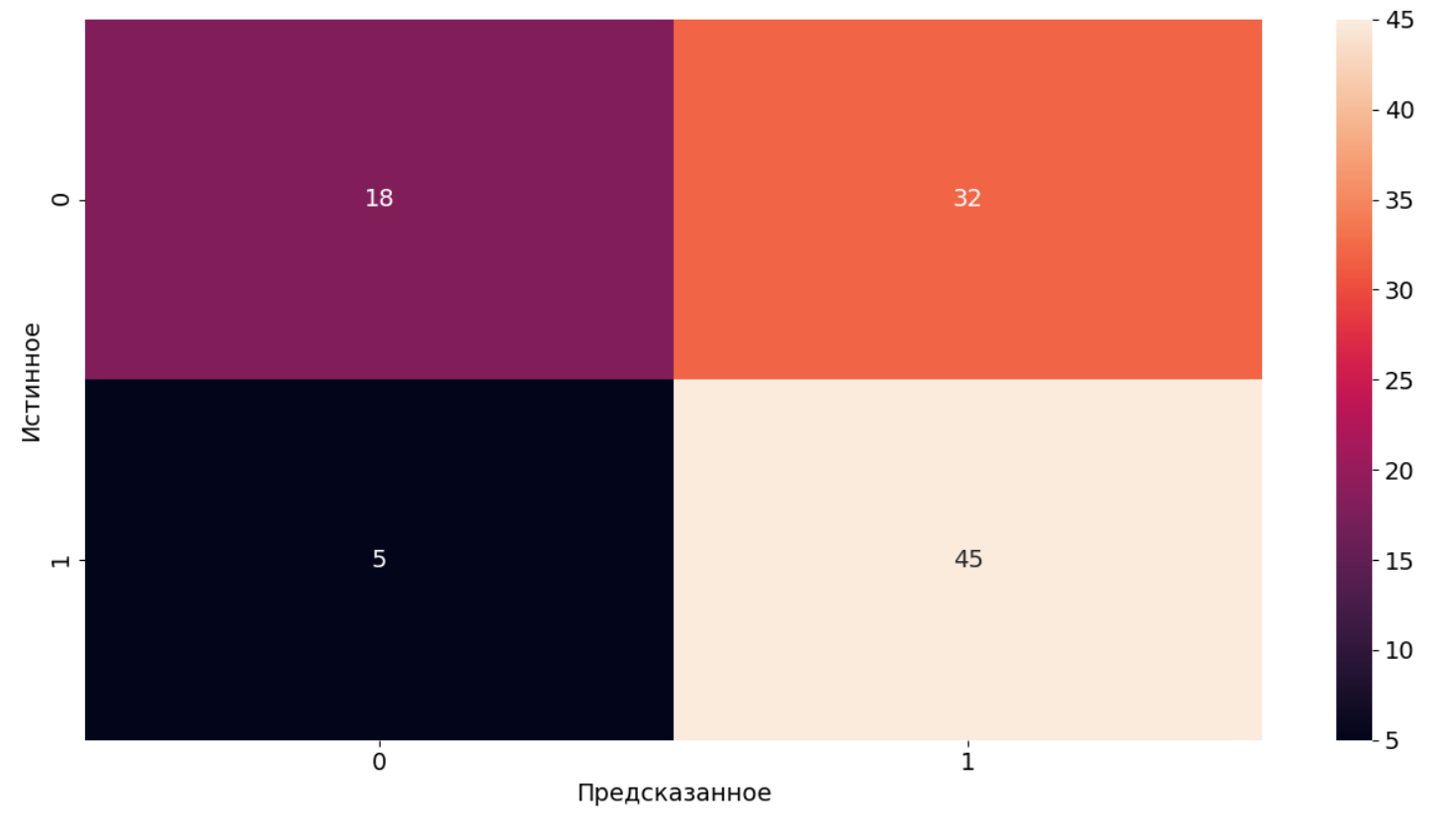
y\_true = y\_data

y\_pred = predict(X, rand\_W, 0.5)

conf\_matrix=get\_confusion\_matrix(y\_true, y\_pred)

show\_confusion\_matrix(conf\_matrix)

Результат



**Достоверность, точность, полнота, F-мера (Accuracy, Precision, Recall, F1)**

Пока что отдохнем и напишем их реализации:

def calculate\_accuracy(y\_true, y\_pred):

TP, TN, FP, FN = get\_truefalse\_metrics(y\_true, y\_pred)

acc=(TP+TN)/(TP+TN+FP+FN)

return acc

# Функция точности, полноты и F-мера

def calculate\_recall\_precision\_f1(y\_true, y\_pred, class\_index):

TP, TN, FP, FN = get\_truefalse\_metrics(y\_true, y\_pred)

'''

class\_index - Индекс класса для которого делается оценка показателей

'''

if class\_index==1:

precision=TP/(TP+FP)

recall=TP/(TP+FN)

if class\_index==0:

precision=TN/(TN+FN)

recall=TN/(TN+FP)

f1=2\*(recall\*precision)/(recall+precision)

return recall, precision, f1

y\_true = np.array([0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1])

y\_pred = np.array([0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0])

acc = calculate\_accuracy(y\_true, y\_pred)

recall\_1, precision\_1, f1\_1 = calculate\_recall\_precision\_f1(y\_true, y\_pred, 1)

recall\_0, precision\_0, f1\_0 = calculate\_recall\_precision\_f1(y\_true, y\_pred, 0)

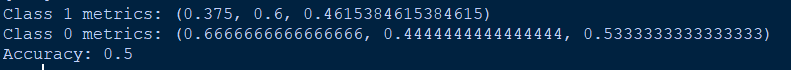
print(f'Class 1 metrics: {recall\_1, precision\_1, f1\_1}')

print(f'Class 0 metrics: {recall\_0, precision\_0, f1\_0}')

print(f'Accuracy: {acc}')

assert np.all([acc == 0.5, recall\_1 == 0.375, precision\_1 == 0.6])

Результат



Когда требование определено, давайте оценим такой случай:

def show\_classification\_metrics(y\_true, y\_pred):

acc = calculate\_accuracy(y\_true, y\_pred)

recall\_1, precision\_1, f1\_1 = calculate\_recall\_precision\_f1(y\_true, y\_pred, 1)

recall\_0, precision\_0, f1\_0 = calculate\_recall\_precision\_f1(y\_true, y\_pred, 0)

print(f'Class 1 metrics: {recall\_1, precision\_1, f1\_1}')

print(f'Class 0 metrics: {recall\_0, precision\_0, f1\_0}')

print(f'Accuracy: {acc}')

y\_true = np.array([0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])

y\_pred = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

show\_classification\_metrics(y\_true, y\_pred)

Результат



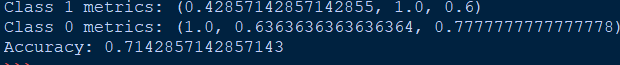
Для проверки проверим второй пример:

y\_true = np.array([0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])

y\_pred = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1])

show\_classification\_metrics(y\_true, y\_pred)

Результат



# **Разделение данных на обучение/тест со стратификацией**

Таким образом, необходимо написать функцию разделения на выборки, которая явно сохранит соотношение классов в выборках.

# Функция сплита со стратификацией

def train\_test\_split\_stratified(X, y, test\_size):

# Объединим X и y, чтобы работать с одним массивом

Xy=np.concatenate((X,y.reshape(len(y),1)),axis=1)

# Создадим листы Xy\_train и Xy\_test, в которые будем формировать выборки

Xy\_train=list()

Xy\_test=list()

#Получение уникальных значений (классов)

uniq=np.unique(Xy[:,-1])

# Создаем цикл по уникальным значениям (классам) массива y

# (так как стратифицируем по целевым классам)

for i in range(len(uniq)):

# Получаем индексы записей, у которых значение равно уникальному

# значению в итерации (равно выбранному классу в итерации)

# numpy.where()

a1=np.where((Xy[:,-1]==uniq[i]),True,False)

ind=np.array(range(len(y)))

ind=ind[a1]

# Перемешиваем индексы, чтобы выборка была случайной

# numpy.random.permutation()

ind=np.random.permutation(ind)

# Умножаем test\_size на количество записей класса, чтобы получить количество записе этого класса, которое уходит на тест

# Не забудьте, что количество должно быть int

cls\_n\_test=(int(TEST\_SIZE\*(len(ind))))

# Выбираем индексы записей, которые пойдут на тест [:cls\_n\_test]

# Остальные индексы пойдут на обучение

index\_test=ind[:cls\_n\_test]

index\_train=ind[cls\_n\_test:]

# Расширяем массив Xy\_train данными по индексам для обучения

Xy\_train.extend(Xy[index\_train,:])

# Расширяем массив Xy\_test данными по индексам для теста

Xy\_test.extend(Xy[index\_test,:])

# Перемешаем еще раз полученные Xy\_train и Xy\_test

Xy\_train=np.random.permutation(Xy\_train)

Xy\_test=np.random.permutation(Xy\_test)

# Разделим обратно на X\_train, X\_test, y\_train, y\_test

# X - все колонки до последней, y - последняя колонка

X\_train=Xy\_train[:,:-1]

X\_test=Xy\_test[:,:-1]

y\_train=Xy\_train[:,-1]

y\_test=Xy\_test[:,-1]

return X\_train, X\_test, y\_train, y\_test

TEST\_SIZE = 0.3

X\_train, X\_test, y\_train, y\_test = train\_test\_split\_stratified(X\_data, y\_data, TEST\_SIZE)

assert np.all([X\_train.shape[0] == 70, X\_test.shape[1] == X\_data.shape[1], y\_test.shape[0] == 30])

print(X\_test.shape)

Результат



После разделения можно провести оценку показателей на тестовой выборке для дальнейшего сравнения:

# TODO - выполните предсказание с порогом 0.5 и отобразите метрики

rand\_W = [-10, 1, 2]

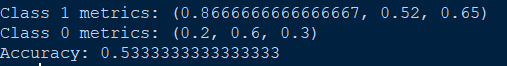
X = X\_test

y\_true = y\_test

y\_pred = predict(X, rand\_W, 0.5)

show\_classification\_metrics(y\_true, y\_pred)

Результат



# **Функция потерь**

Графики функций представлены ниже.

Y = np.linspace(0, 1, 100)

J\_1 = -np.log(Y)

J\_2 = -np.log(1-Y)

plt.plot(Y, J\_1, 'k', Y, J\_2, 'k--')

plt.grid()

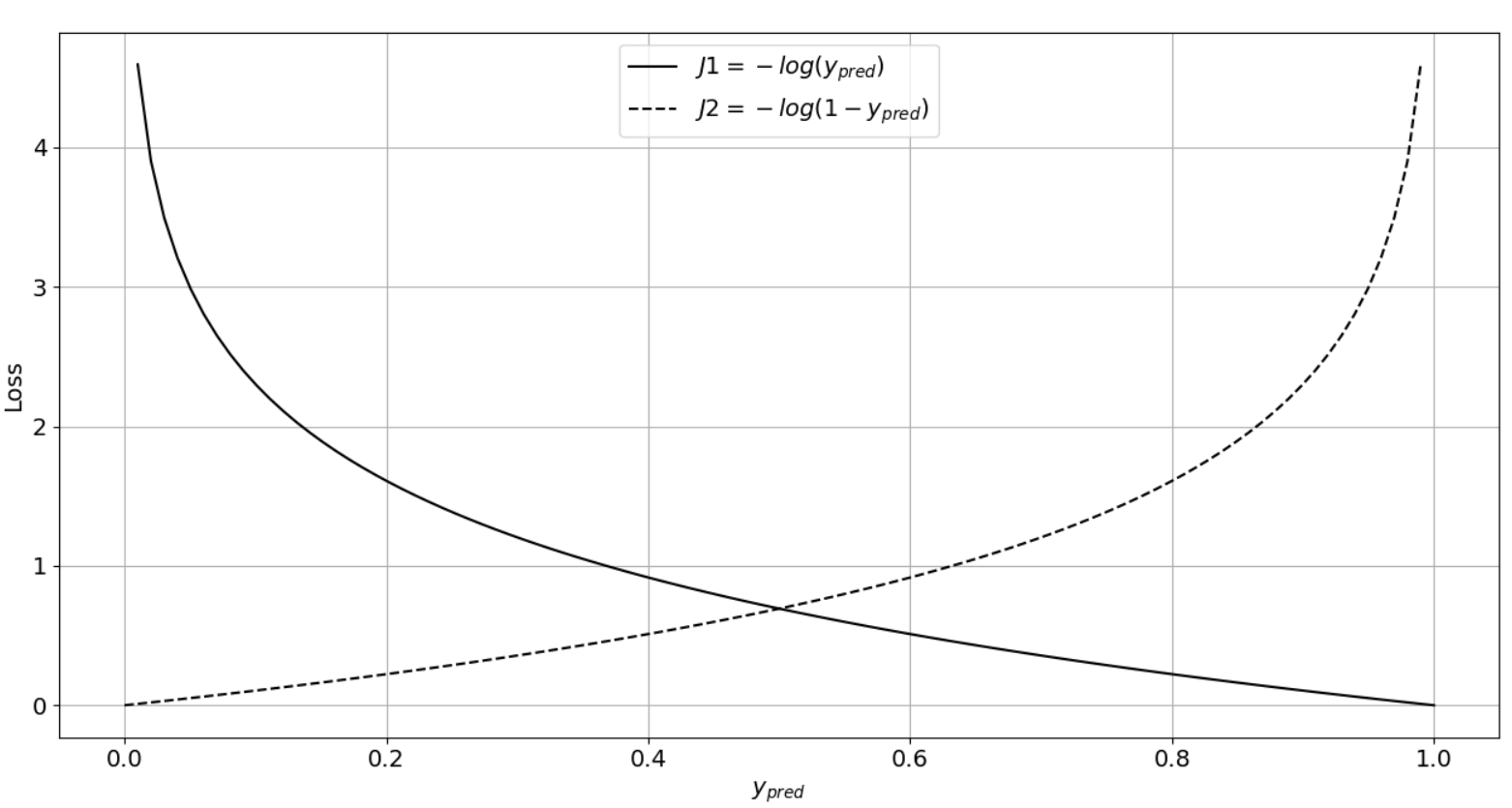
plt.xlabel('$y\_{pred}$')

plt.ylabel('Loss')

plt.legend(['$J1=-log(y\_{pred})$', '$J2=-log(1-y\_{pred})$'])

plt.show()

Результат



# **Производная функции потерь**

Таким образом получена формула для вычисления производной функции потерь IMG_256, самое время реализовать как саму фукнцию потерь, так и ее производную:

def bce\_loss(y\_true, y\_pred):

loss=np.mean(-y\_true\*(np.log(y\_pred))-(1-y\_true)\*(np.log(1-y\_pred)))

return loss

# Производная функции потерь

def bce\_loss\_deriv(X, y\_true, y\_pred):

dJ=np.mean(np.transpose(X)\*(y\_pred-y\_true),axis=1)

return dJ

# Проверка

y\_true = np.array([0, 0, 1, 1])

y\_pred = np.array([0.98, 0.3333, 0.2, 0.8])

X = np.array([

[1, 2, 3],

[2, 3, 4],

[8, 9, 0],

[7, 6, 5],

])

loss = bce\_loss(y\_true, y\_pred)

dJ = bce\_loss\_deriv(X, y\_true, y\_pred)

dJ\_true = np.array([-1.53835, -1.360025, 0.8183])

assert loss == 1.5375048946336445

assert np.all(np.abs(dJ-dJ\_true) < 1e-8)

**Обучение модели  
Задание**

Настало время обучить нашу модель и эта задача переходит вам! Давайте напишем функцию обучения модели. Учтите следующие особенности:

* в ходе обучения применяется степень уверенности, а не конечный предсказанный класс;
* функция predict\_proba() принимает на вход исходные данные и сама добавляет колонку, функция bce\_loss\_deriv() более общая и не добавляет колонку единиц.

# Функция модели обучения

def fit\_model(X, y, lr, n\_iter):

y\_true=y

W=np.zeros(X.shape[1]+1)

loss\_history=[]

y\_pred=predict\_proba(X,W)

X1=np.concatenate((np.ones((X.shape[0],1),int),X),axis=1)

for i in range(n\_iter):

dJ=bce\_loss\_deriv(X1, y\_true, y\_pred)

W=W-lr\*dJ

y\_pred=predict\_proba(X,W)

loss=bce\_loss(y\_true, y\_pred)

loss\_history.append(loss)

return W, loss\_history

def show\_loss(loss\_history):

plt.plot(loss\_history)

plt.grid()

plt.xlabel('iter')

plt.ylabel('loss')

plt.show()

trained\_W, loss\_history = fit\_model(

lr=0.1,

n\_iter=5000,

X=X\_train,

y=y\_train

)

show\_loss(loss\_history)

# TODO - Отобразите значения метрик

y\_pred=predict(X\_train,trained\_W,0.5)

show\_classification\_metrics(y\_train, y\_pred)

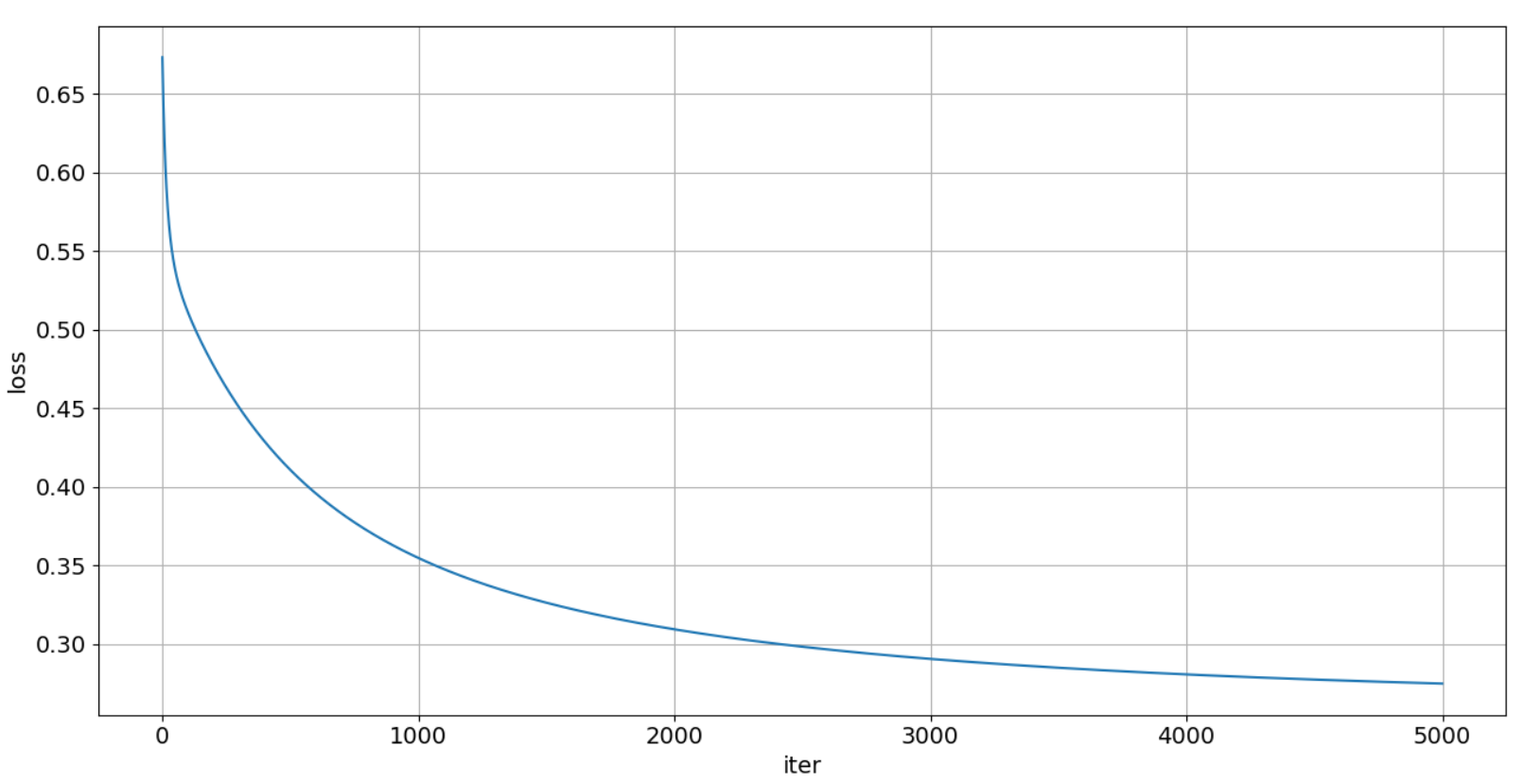
# TODO - Отобразите матрицу ошибок

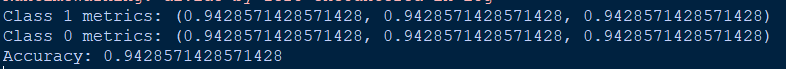
conf\_matrix=get\_confusion\_matrix(y\_train, y\_pred)

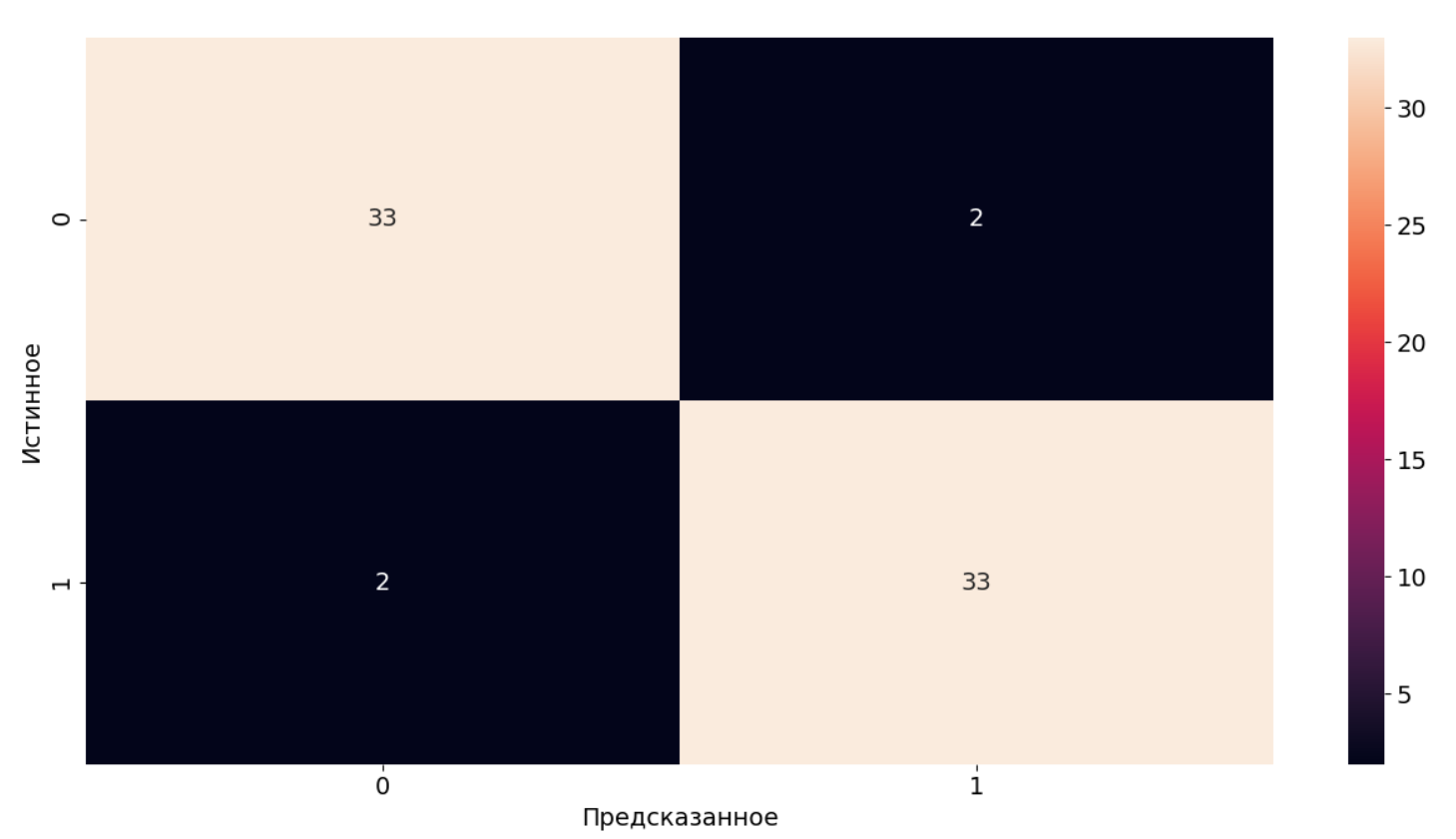
show\_confusion\_matrix(conf\_matrix)

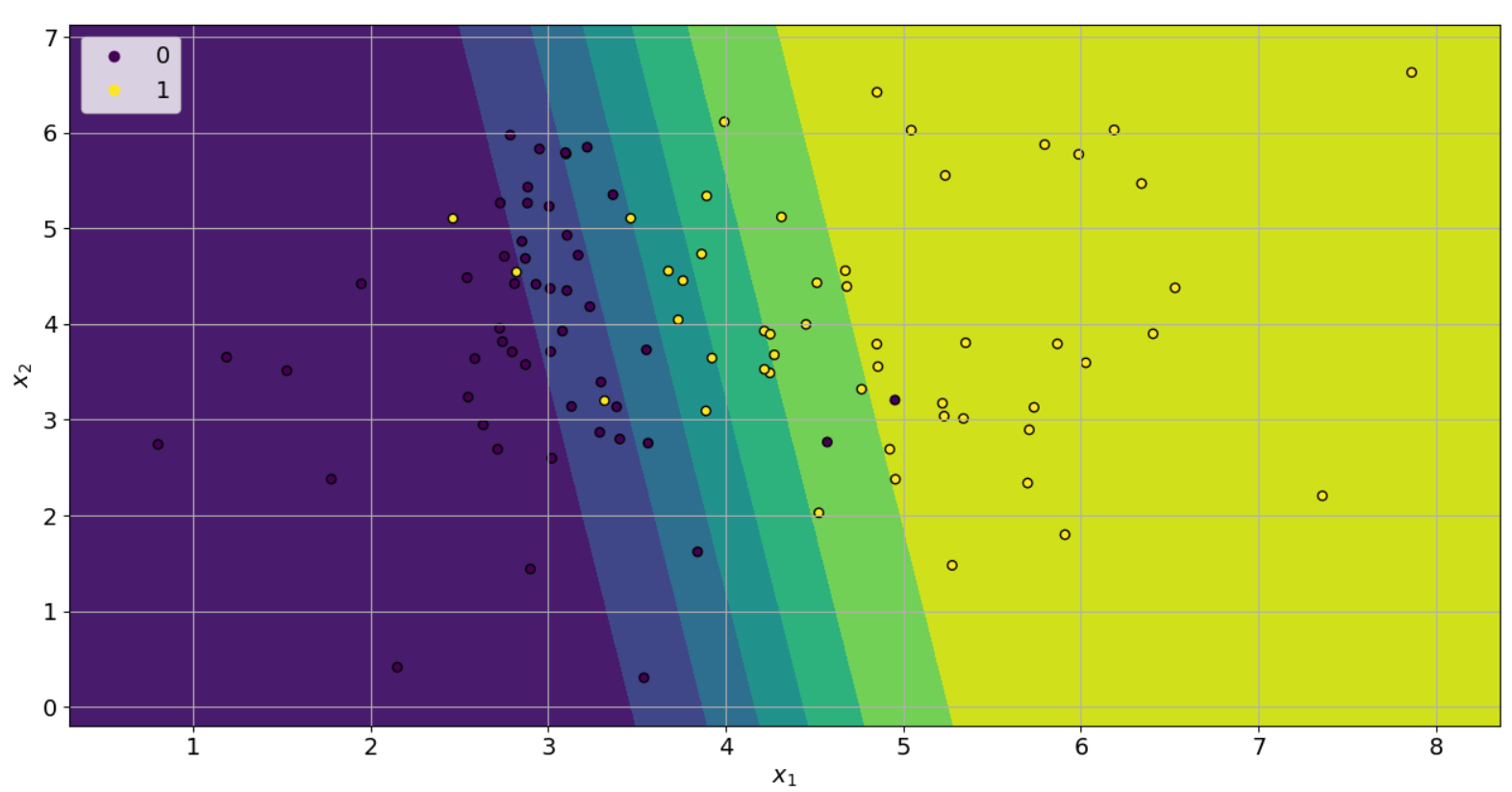
plot\_2d\_decision\_boundary(X\_data, trained\_W, y\_data)

Результат









## **Задание**

Изучите, как меняются показатели в зависимости от выбранного порога (7 разных значений в диапазоне IMG_256.

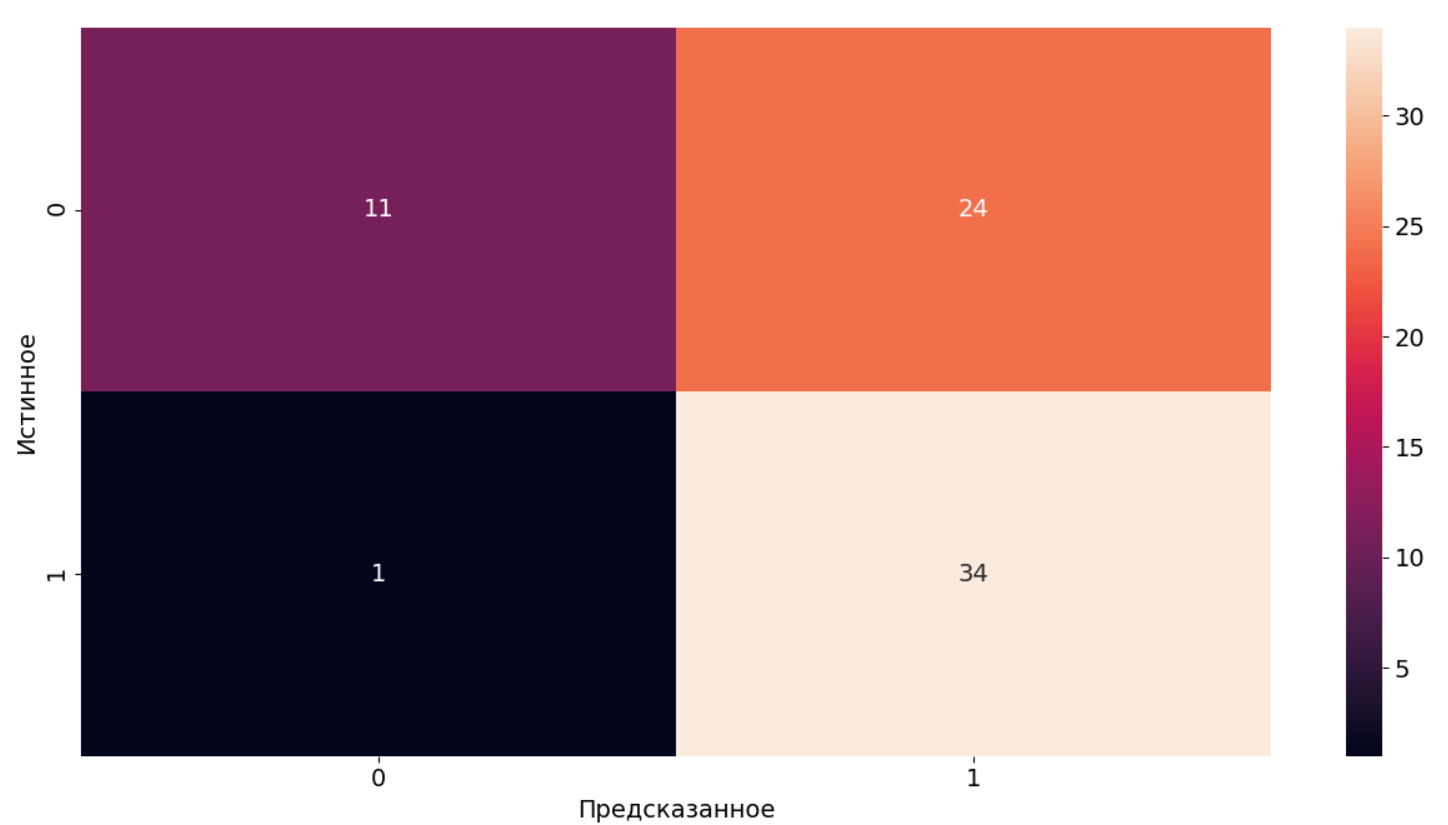
for i in np.linspace(0.1,0.95,7):

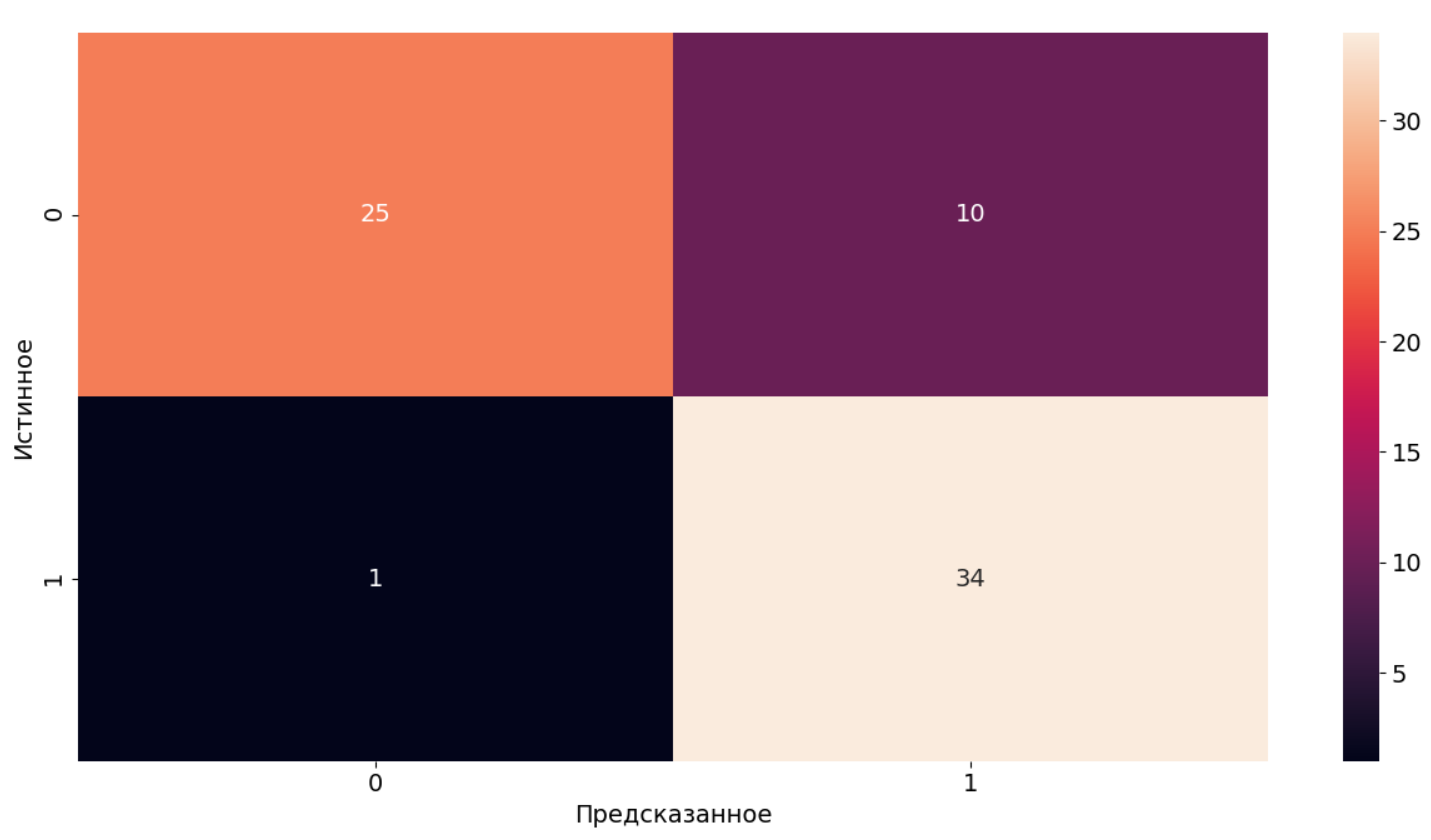
y\_pred=predict(X\_train,trained\_W,i)

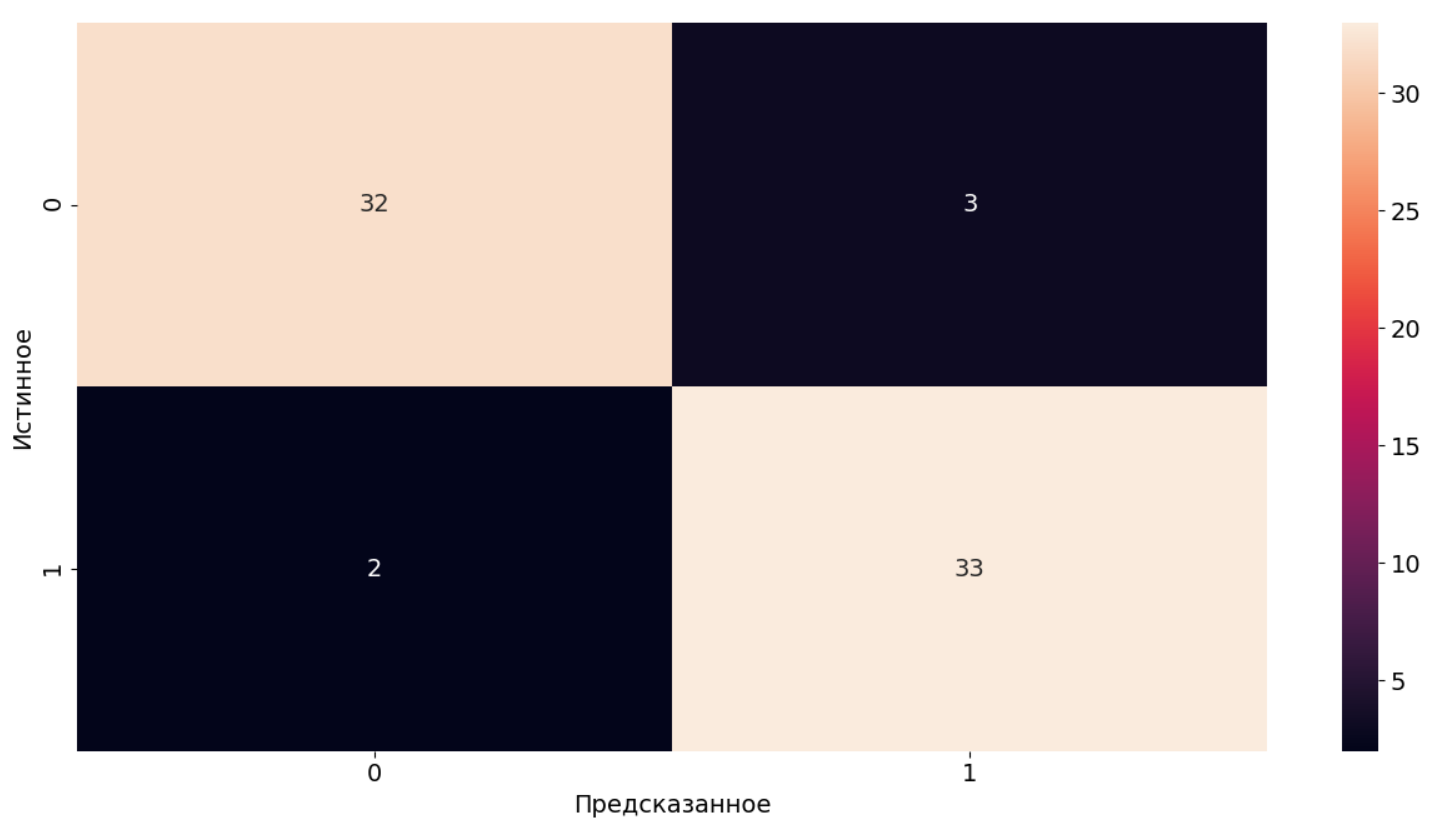
conf\_matrix=get\_confusion\_matrix(y\_train, y\_pred)

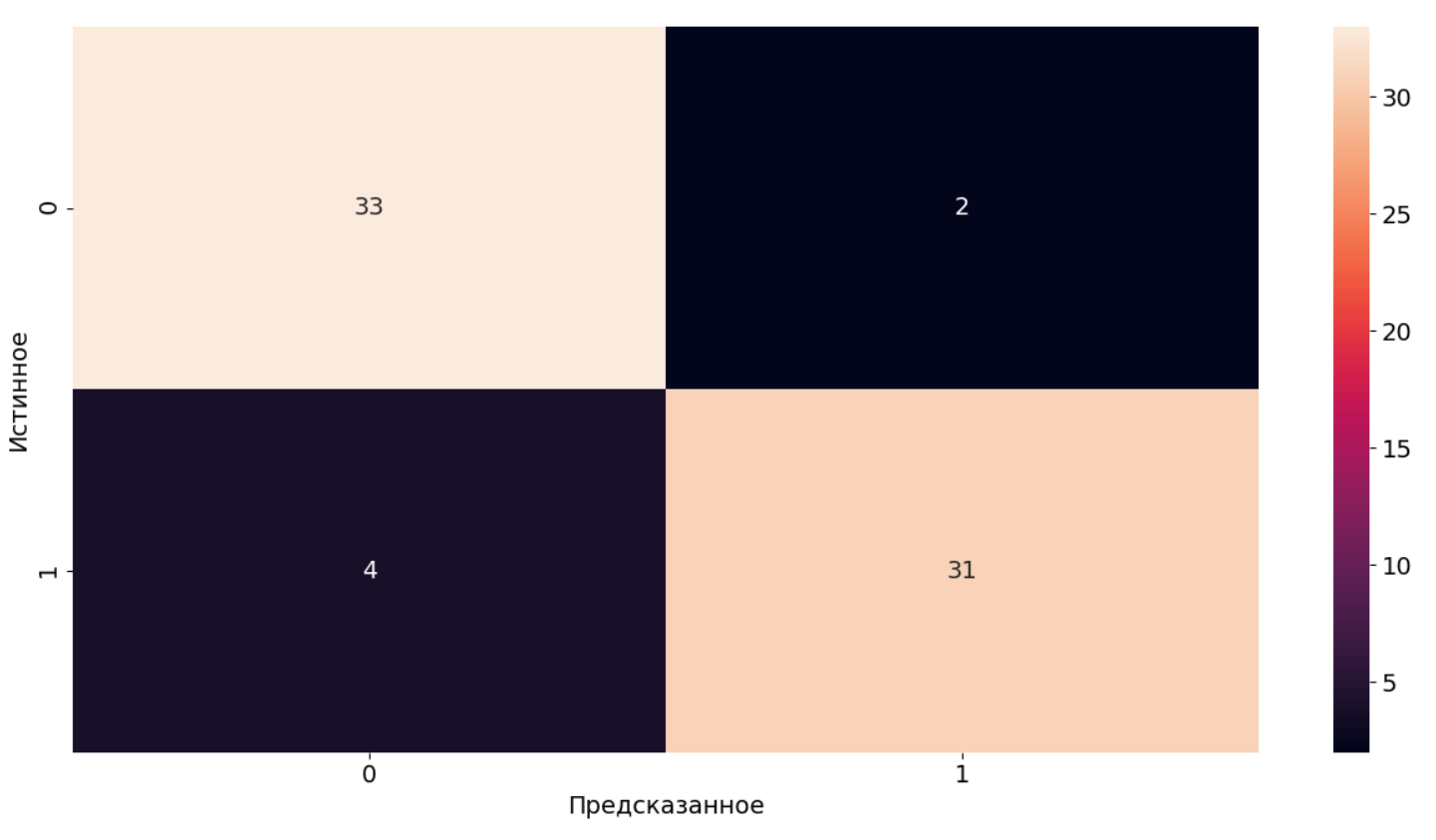
show\_confusion\_matrix(conf\_matrix)

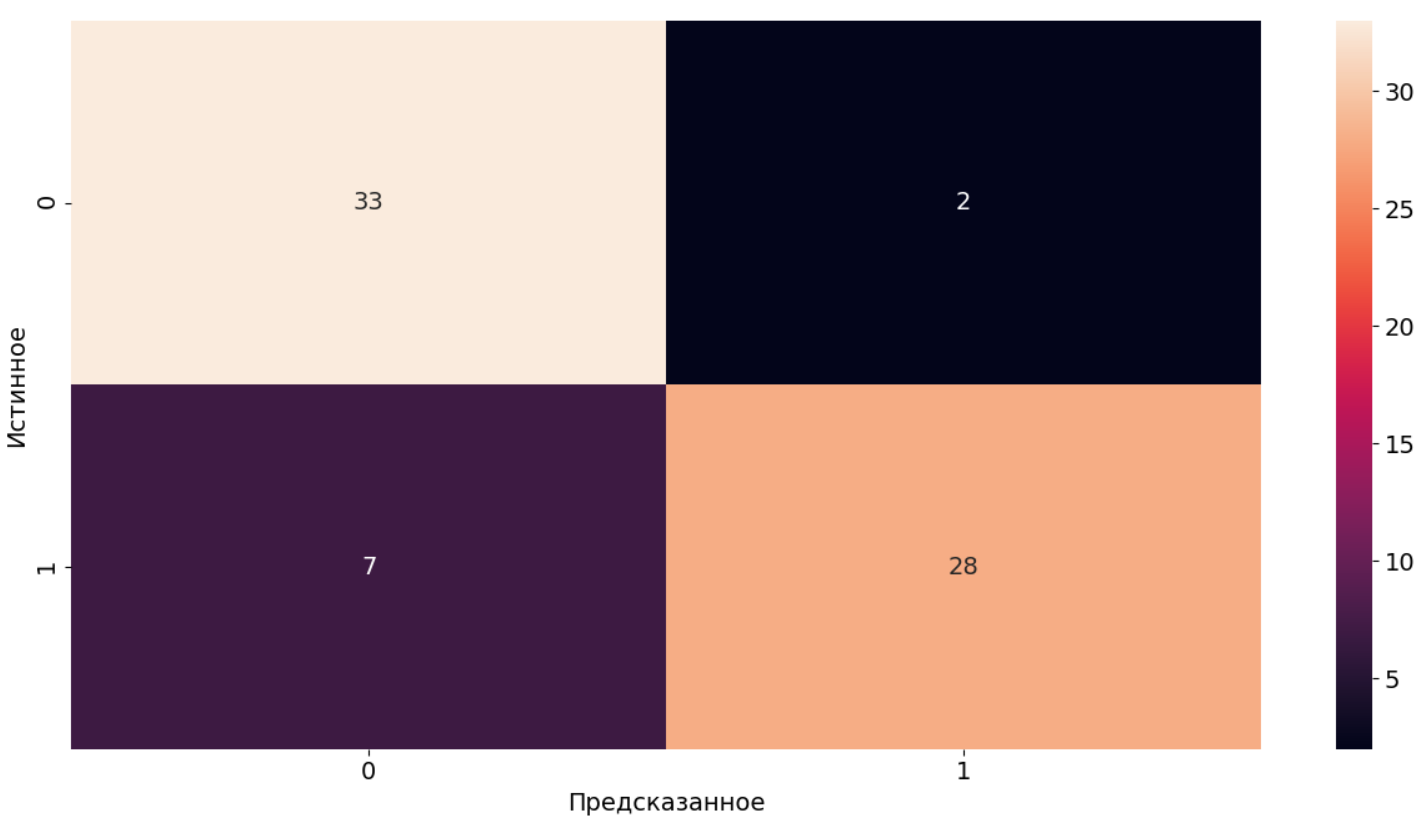
Результат

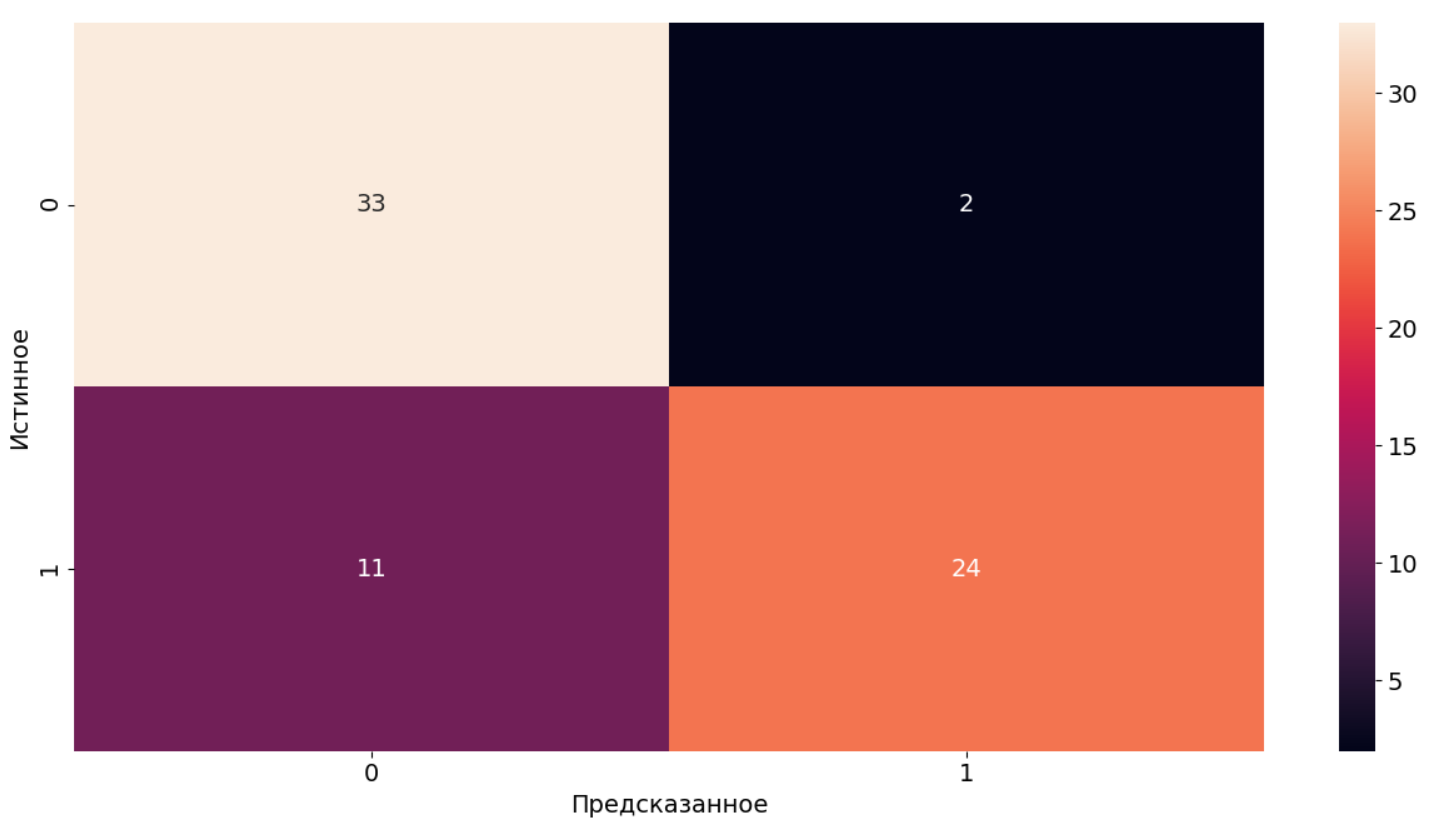


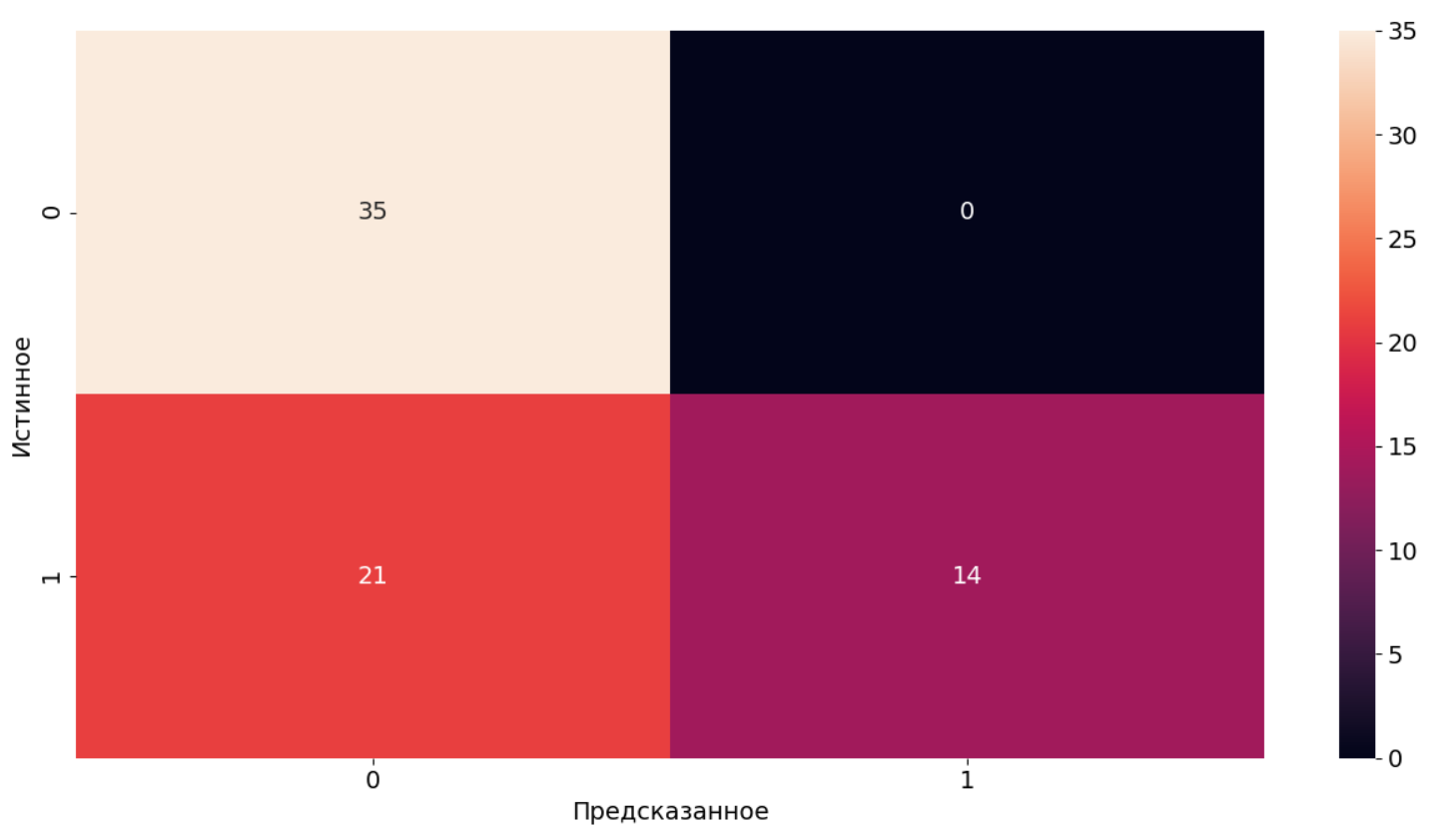












## **ROC кривая и AUC**

# функции получения TRP и FPR

def get\_tpr\_fpr\_metrics(y\_true, y\_pred, class\_index):

TP, TN, FP, FN = get\_truefalse\_metrics(y\_true, y\_pred)

if class\_index==1:

TPR=TP/(TP+FN)

FPR=FP/(FP+TN)

if class\_index==0:

TPR=FP/(FP+TN)

FPR=TP/(TP+FN)

return TPR, FPR

# TEST

y\_true = np.array([0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0])

y\_pred = np.array([0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0])

tpr\_0, fpr\_0 = get\_tpr\_fpr\_metrics(y\_true, y\_pred, 0)

assert np.isclose(tpr\_0, 0.5)

assert np.isclose(fpr\_0, 0.5)

Теперь самое время пробежаться по 11 значениям порога (с шагом 0.1) и отобразить кривую, которая получается в осях значений (x ~ FPR и y ~ TPR).

def show\_roc\_curve(X, y, W, class\_index):

diag\_values = np.array([

[0, 0],

[1, 1],

])

render\_n\_points = 10

threshold\_values = np.arange(render\_n\_points+1)/render\_n\_points

tpr\_values = []

fpr\_values = []

for threshold\_value in threshold\_values:

y\_pred = predict(X, W, threshold=threshold\_value)

tpr\_0, fpr\_0 = get\_tpr\_fpr\_metrics(y, y\_pred, class\_index)

tpr\_values.append(tpr\_0)

fpr\_values.append(fpr\_0)

plt.plot(fpr\_values, tpr\_values)

plt.plot(diag\_values[:,0], diag\_values[:,1], 'b--')

plt.xlabel('FPR')

plt.ylabel('TPR')

plt.grid(True)

plt.figure(figsize=[17,10])

plt.subplot(121)

show\_roc\_curve(X\_train, y\_train, trained\_W, class\_index=0)

plt.title('Обученная модель')

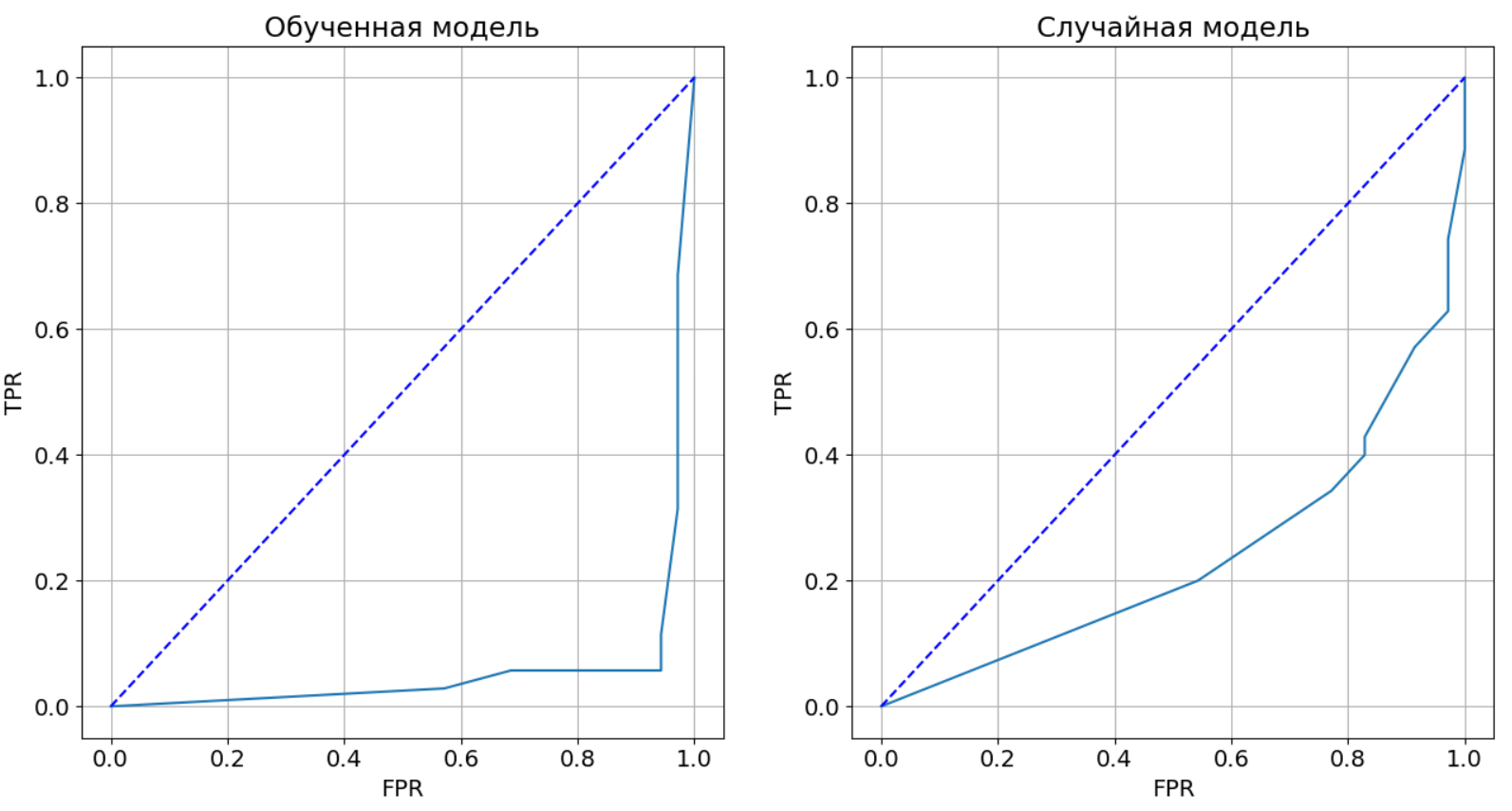
plt.subplot(122)

show\_roc\_curve(X\_train, y\_train, rand\_W, class\_index=0)

plt.title('Случайная модель')

plt.show()

Результат



## **Задание - PR-кривая**

Теперь, когда вы освоили отображение ROC-кривой, реализуйте и сравните модели по другому показателю под названием PR-кривая. Суть и идея все те же, смотрим на значения Precision и Recall на разных порогах и рисуем кривую в осях (x ~ Recall, y ~ Precision).

def show\_pr\_curve(X, y, W, class\_index):

render\_n\_points = 10

threshold\_values = np.arange(render\_n\_points+1)/render\_n\_points

recall\_values = []

precision\_values = []

for threshold\_value in threshold\_values:

y\_pred = predict(X, W, threshold=threshold\_value)

recall, precision, f1 = calculate\_recall\_precision\_f1(y, y\_pred, class\_index)

recall\_values.append(recall)

precision\_values.append(precision)

plt.plot(recall\_values, precision\_values)

plt.xlabel('recall')

plt.ylabel('precision')

plt.grid(True)

plt.figure(figsize=[17,10])

plt.subplot(121)

show\_pr\_curve(X\_train, y\_train, trained\_W, class\_index=0)

plt.title('Обученная модель')

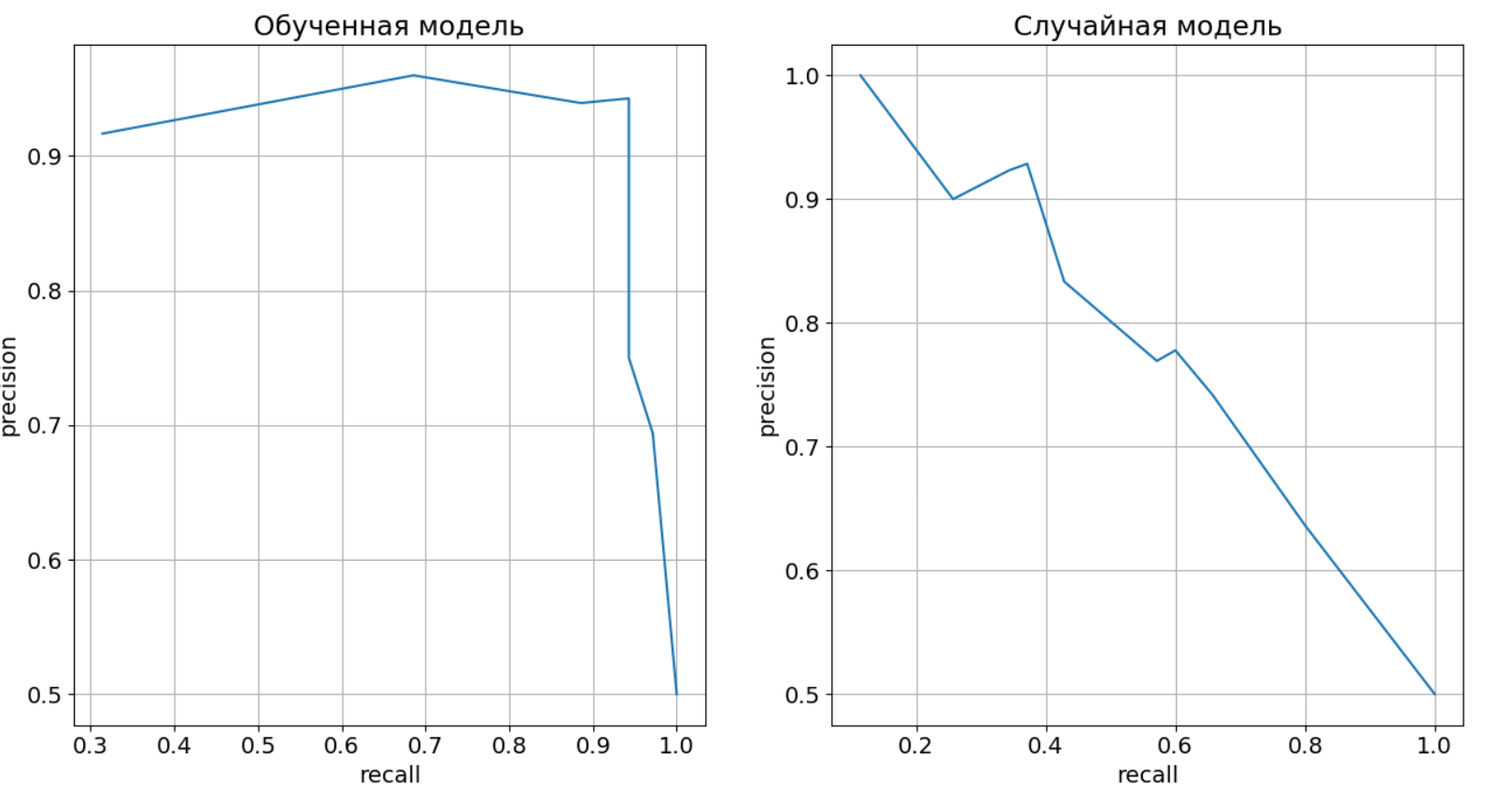
plt.subplot(122)

show\_pr\_curve(X\_train, y\_train, rand\_W, class\_index=0)

plt.title('Случайная модель')

plt.show()

Результат



# **Кросс-валидация**

Реализуйте K-fold обучение и оценку по методу кросс-валидации:

def get\_kfold\_splits(n\_samples, kfolds):

''' Функция получения индексов для разделения на K фолдов

Параметры

---------

n\_samples: int

Количество записей в данных

k: int

Количество фолдов

Возвращает

----------

Список, в котором находится k пар (кортежей)

Каждая пара представляет собой два массива:

- массив индексов обучающей выборки

- массив индексов тестовой выборки

'''

folds\_indexes = []

# Создайте список перемешанных индексов записей

# numpy.random.permutation(range(n\_samples))

mixed\_ind=np.random.permutation(range(n\_samples))

# Получите размер фолда

# n\_samples/kfolds

step=n\_samples/kfolds

for i in range(kfolds):

# Получите индекс начала тестового фолда

start\_idx = int(i\*step)

# Получите индекс конца тестового фолда

end\_idx= int (i\*step+step)

# Создайте массив нулей с типом bool

# Это будет маска для выбора индексов обучения/теста

mask=np.zeros(X\_data.shape[0],dtype=bool)

# По диапазону индексов тестового фолда запишите True в маску

# [start\_idx:end\_idx] = True

mask[start\_idx:end\_idx] = True

# По маске из перемешанных индексов получите индексы тестового фолда

# test\_indexes

test\_indexes=mixed\_ind[mask]

# По инвертированной маске из перемешанных индексов

# получите индексы обучающего фолда

# train\_indexes

train\_indexes=mixed\_ind[~mask]

# Запишем пару массивов индексов в результирующий список

folds\_indexes.append(

(train\_indexes, test\_indexes)

)

return folds\_indexes

# TODO - напишите процесс обучения и оценки с K-fold CV

K\_FOLDS = 5

f1\_scores = []

for train\_indexes, test\_indexes in get\_kfold\_splits(X\_data.shape[0], K\_FOLDS):

# По индексам фолдов получите

# {X,y}\_{train,test} из данных X\_data

X\_train=X\_data[train\_indexes,:]

X\_test=X\_data[test\_indexes,:]

y\_train=y\_data[train\_indexes]

y\_test=y\_data[test\_indexes]

# Произведите обучение модели, оцените на тестовом фолде

# и получите среднее значение f1 по всем классам,

# добавьте значение в массив f1\_scores

trained\_W, loss\_history = fit\_model(

lr=0.1,

n\_iter=5000,

X=X\_train,

y=y\_train)

y\_pred=predict(X\_test,trained\_W,0.5)

recall\_1, precision\_1, f1\_1 = calculate\_recall\_precision\_f1(y\_test, y\_pred, 1)

f1\_scores.append(f1\_1)

recall\_0, precision\_0, f1\_0 = calculate\_recall\_precision\_f1(y\_test, y\_pred, 0)

f1\_scores.append(f1\_0)

f1\_scores.append(f1\_1)

# Получите среднее значение по массиву f1\_scores

print(np.mean(f1\_scores))

Результат



# **Дисбаланс классов в данных**

В нашем рассмотренном случае мы работали с данными, в которых количество записей для одного и другого классов были приблизительно одинаковым:

y\_data[y\_data == 0].shape, y\_data[y\_data == 1].shape

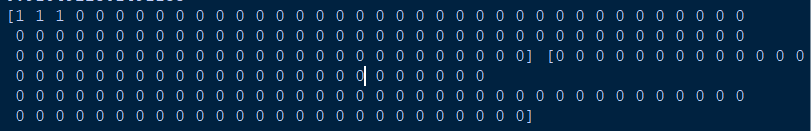
y\_true = np.zeros(100, dtype=int)

y\_true[:3] = 1

y\_pred = np.zeros(100, dtype=int)

print(y\_true, y\_pred)

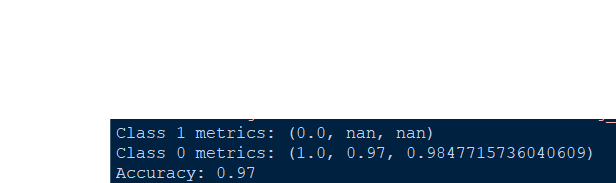
Результат



Видите, в истинных значениях всего 3 случая мошеничества, а модель просто всегда говорит, что мошеничества нет, так давайте оценим метрики:

show\_classification\_metrics(y\_true, y\_pred)

Результат



Для практики работы с данными, которые имеют дисбаланс создадим искуственный датасет:

show\_classification\_metrics(y\_true, y\_pred)

X\_data, y\_data = make\_classification(

n\_samples=200,

n\_features=2,

n\_redundant=0,

n\_informative=2,

n\_clusters\_per\_class=2,

random\_state=RANDOM\_STATE,

weights=[0.9, 0.1]

)

pnts\_scatter = plt.scatter(X\_data[:, 0], X\_data[:, 1], marker='o', c=y\_data, s=50, edgecolor='k', )

plt.scatter(X\_new[:, 0], X\_new[:, 1], marker='x', s=100, edgecolor='k', c='green')

plt.xlabel('$x\_1$')

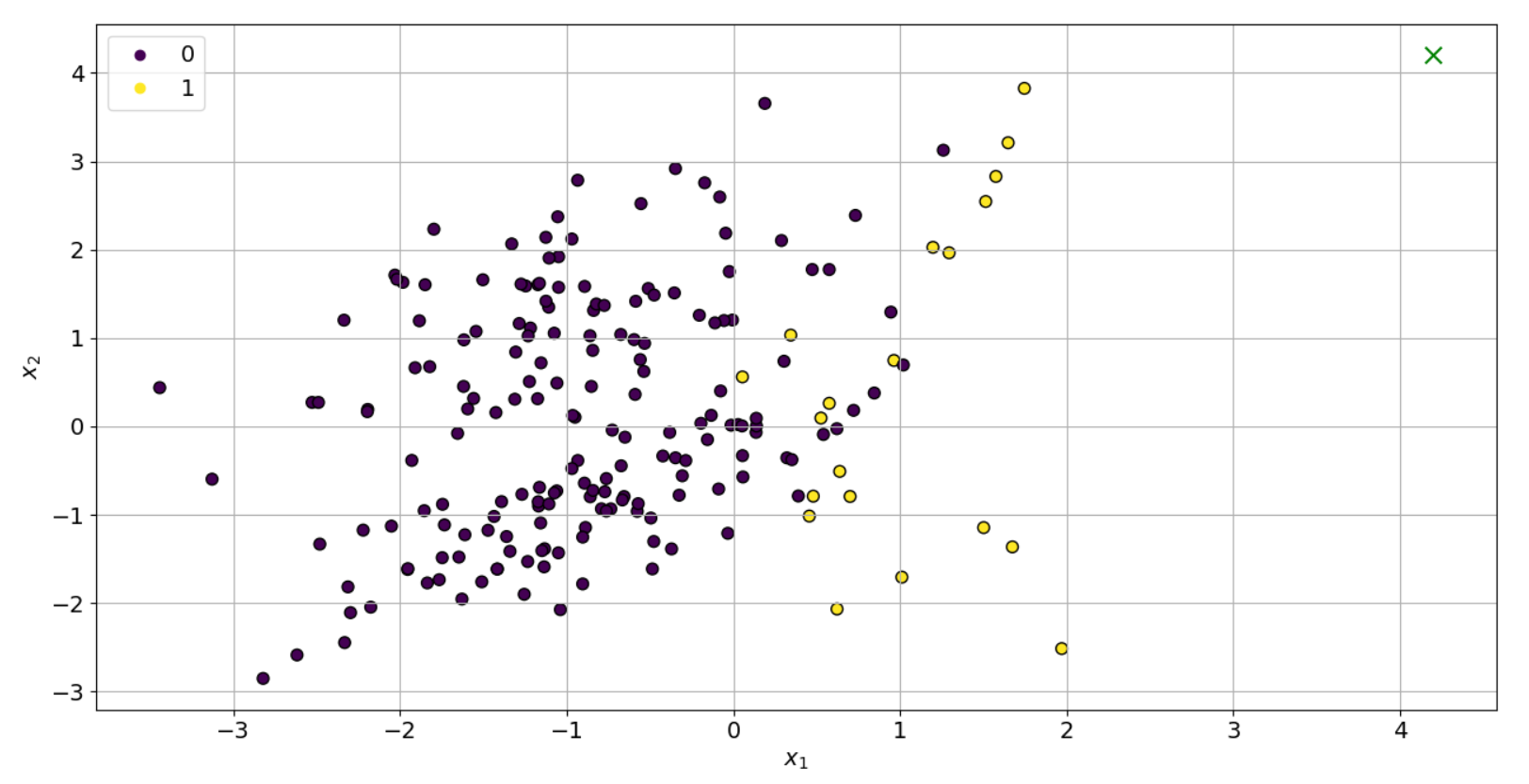
plt.ylabel('$x\_2$')

plt.grid(True)

plt.legend(handles=pnts\_scatter.legend\_elements()[0], labels=['0', '1'])

plt.show()

Результат





## **Задание**

Поразмышляйте, какие негативные стороны имеет метод разделения на обучение/тест случайной выборкой при наличии дисбаланса классов.

Теперь, давайте обучим нашу модель и оценим, как она работает при наличии такого дисбаланса данных. Оценку проведем методом кросс-валидации для всех данных для усреднения оценки.

Дополнительно выведите стандартное отклонение значений оценок (f1\_scores), чтобы понять, как это скажется на данных.

Для этого необходимо в первую очередь реализовать функцию генерации K-фолдов с учетом стратификации:

# TODO - разберите вместе с преподавателем

def get\_kfold\_splits\_stratified(y, kfolds):

''' Функция получения индексов для разделения на K фолдов

со стратификацией по y

Параметры

---------

y: [n\_samples]

Вектор целевых классов, по которому будет производиться стратификация

k: int

Количество фолдов

Возвращает

----------

Список, в котором находится k пар (кортежей)

Каждая пара представляет собой два массива:

- массив индексов обучающей выборки

- массив индексов тестовой выборки

'''

folds\_indexes = []

n\_samples = len(y)

step = n\_samples/kfolds

classes\_indices = {}

unique\_classes = np.unique(y)

for k in unique\_classes:

class\_indices = np.where(y==k)[0]

class\_indices = np.random.permutation(class\_indices)

classes\_indices[k] = class\_indices

for i in range(kfolds):

test\_indexes = []

train\_indexes = []

for k in unique\_classes:

class\_indices = classes\_indices[k]

step = len(class\_indices)/kfolds

start\_idx = int(i\*step)

end\_idx = int((i+1)\*step)

mask = np.zeros\_like(class\_indices, dtype=bool)

mask[start\_idx:end\_idx] = True

test\_indexes.extend(class\_indices[mask])

train\_indexes.extend(class\_indices[~mask])

train\_indexes = np.random.permutation(train\_indexes)

test\_indexes = np.random.permutation(test\_indexes)

folds\_indexes.append(

(train\_indexes, test\_indexes)

)

return folds\_indexes

# Проверка

folds = get\_kfold\_splits\_stratified(y\_data, K\_FOLDS)

fold = folds[2]

test\_indexes = fold[1]

assert (y\_data[test\_indexes] == 1).sum() == 4

# Реализация обучения со стратификацией CV

K\_FOLDS = 5

f1\_scores = []

for train\_indexes, test\_indexes in get\_kfold\_splits\_stratified(y\_data, K\_FOLDS):

# По индексам фолдов получите

# {X,y}\_{train,test} из данных X\_data

X\_train=X\_data[train\_indexes,:]

X\_test=X\_data[test\_indexes,:]

y\_train=y\_data[train\_indexes]

y\_test=y\_data[test\_indexes]

# Произведите обучение модели, оцените на тестовом фолде

# и получите среднее значение f1 по всем классам,

# добавьте значение в массив f1\_scores

trained\_W, loss\_history = fit\_model(

lr=0.1,

n\_iter=5000,

X=X\_train,

y=y\_train)

y\_pred=predict(X\_test,trained\_W,0.5)

recall\_1, precision\_1, f1\_1 = calculate\_recall\_precision\_f1(y\_test, y\_pred, 1)

f1\_scores.append(f1\_1)

recall\_0, precision\_0, f1\_0 = calculate\_recall\_precision\_f1(y\_test, y\_pred, 0)

f1\_scores.append(f1\_0)

f1\_scores.append(f1\_1)

# Получите среднее значение по массиву f1\_scores

print(np.mean(f1\_scores))

# TODO - напишите реализацию метода undersample

def undersample\_by\_class(X, y, undersample\_class, target\_count):

index1=np.arange(len(X))[~(y==undersample\_class)]

index0=np.arange(len(X))[y==undersample\_class]

ind0=np.random.choice(index0,target\_count)

ind=np.concatenate((index1,ind0))

X=X[ind,:]

y=y[ind]

return X, y

# Проверка

X\_check, y\_check = undersample\_by\_class(X\_data, y\_data, 0, 200)

assert (y\_check==0).sum() == 200

class1\_count = (y\_data==1).sum()

X\_new\_data, y\_new\_data = undersample\_by\_class(X\_data, y\_data, 0, class1\_count)

K\_FOLDS = 5

f1\_scores = []

for train\_indexes, test\_indexes in get\_kfold\_splits\_stratified(y\_new\_data, K\_FOLDS):

# По индексам фолдов получите

# {X,y}\_{train,test} из данных X\_data

X\_train=X\_new\_data[train\_indexes,:]

X\_test=X\_new\_data[test\_indexes,:]

y\_train=y\_new\_data[train\_indexes]

y\_test=y\_new\_data[test\_indexes]

# Произведите обучение модели, оцените на тестовом фолде

# и получите среднее значение f1 по всем классам,

# добавьте значение в массив f1\_scores

trained\_W, loss\_history = fit\_model(

lr=0.1,

n\_iter=5000,

X=X\_train,

y=y\_train)

y\_pred=predict(X\_test,trained\_W,0.5)

recall\_1, precision\_1, f1\_1 = calculate\_recall\_precision\_f1(y\_test, y\_pred, 1)

f1\_scores.append(f1\_1)

recall\_0, precision\_0, f1\_0 = calculate\_recall\_precision\_f1(y\_test, y\_pred, 0)

f1\_scores.append(f1\_0)

f1\_scores.append(f1\_1)

# Получите среднее значение по массиву f1\_scores

print(np.mean(f1\_scores))

Результат

