# **Классификация классификаций**

Теперь создадим набор данных для работы с ними:

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=210,

n\_features=2,

n\_redundant=0,

n\_informative=2,

n\_clusters\_per\_class=1,

n\_classes=3,

random\_state=RANDOM\_STATE

)

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

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

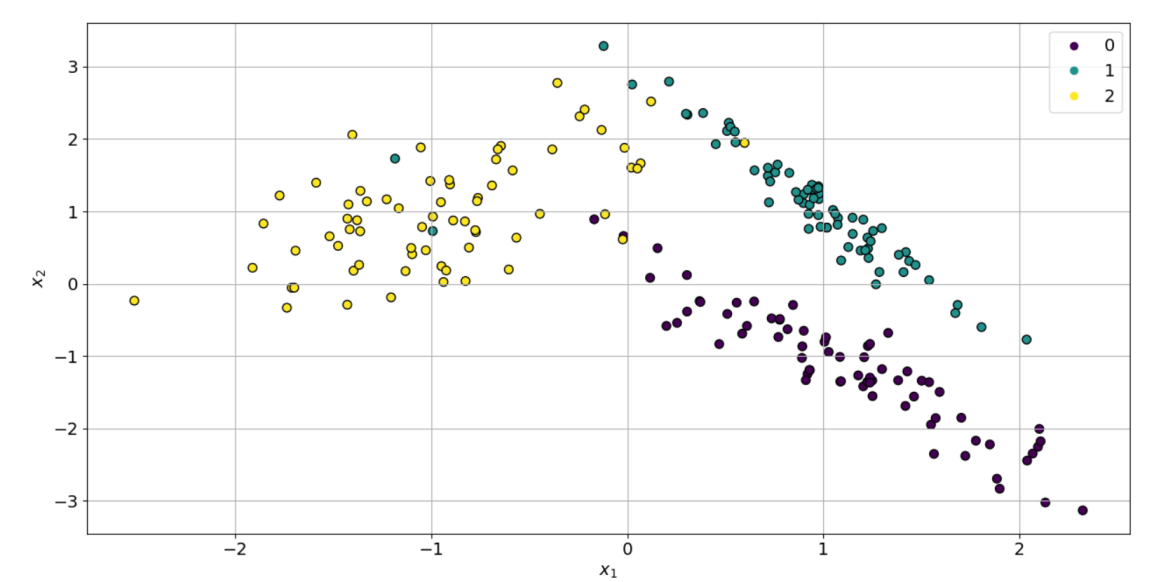
plt.ylabel('$x\_2$')

plt.grid(True)

plt.legend(handles=pnts\_scatter.legend\_elements()[0], labels=['0', '1', '2'])

plt.show()

Результат



**Функция предказания**  
Настало время реализации данного функционала. Напишем необходимые функции для выполнения предсказания:

# TODO - напишите функции вычисления softmax и предсказания

def softmax(z):

z\_norm=(np.exp(z)/np.sum(np.exp(z)))

return z\_norm

def predict\_proba(X, W):

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

z=X@W

y\_proba=np.zeros(z.shape)

for i in range(z.shape[0]):

y\_proba[i,:]=softmax(z[i,:])

return y\_proba

def predict(X, W):

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

y\_pred=np.argmax(y\_proba,axis=1)

return y\_pred

# Проверка

X = np.array([

[1, 2],

[-3, 4],

[1, 2],

[-1, 1],

[0, 5],

[0, -2],

])

W = np.array([

[1, 2, 0],

[0, 2, 1],

[2, 2, 0]

])

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

y\_pred = predict(X, W)

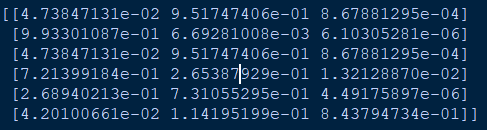
y\_pred\_true = np.array([1, 0, 1, 0, 1, 2])

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

assert np.all(np.abs(np.sum(y\_prob, axis=1)-np.ones(X.shape[0])) < 1e-8)

print(y\_prob)

Результат



**Метрики**

## **Задание**

С учетом ранее изученных методов визуальной и численной оценки классификатора произведите разделение данных на выборку обучения/тестирования и проведите оценку работы классификатора на данных со случайно заданными весами:

Для разделения воспользуйте функцией sklearn.model\_selection.train\_test\_split из пакета sklearn и обратите внимание на аргумент stratify

rand\_W = np.array([

[-1, -2, -2],

[-7, 2, -3],

[-1, 1, 4],

])

# TODO - произведите разделение данных

# NOTE - не забудьте зафиксировать RANDOM\_STATE

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_data, y\_data, test\_size=0.3, random\_state=RANDOM\_STATE, stratify=y\_data)

# Проверка

assert np.all([

X\_train.shape[0] == 147,

X\_test.shape[1] == X\_data.shape[1],

y\_test.shape[0] == 63,

y\_test[y\_test == 2].shape[0] == 20])

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(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()

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

# TODO - функция генерации матрицы ошибок по векторам предсказания/разметки

class\_count=len(np.unique(y\_true))

conf\_mtrx=np.ndarray((class\_count,class\_count))

for i in range(conf\_mtrx.shape[0]):

for j in range(conf\_mtrx.shape[1]):

conf\_mtrx[i,j]=np.sum((y\_true==i)&(y\_pred==j))

return conf\_mtrx

def draw\_confusion\_matrix(conf\_mtrx):

# TODO - функция отображения матрицы ошибок

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

plt.xlabel('Предсказанное')

plt.ylabel('Истинное')

plt.show()

# TODO - функции вычисления метрик

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

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

acc=np.sum(np.diag(conf\_mtrx))/np.sum(conf\_mtrx)

return acc

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

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

precision=(conf\_mtrx[class\_index,class\_index]/np.sum(conf\_mtrx[:,class\_index]))

recall=(conf\_mtrx[class\_index,class\_index]/np.sum(conf\_mtrx[class\_index,:]))

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

return recall, precision, f1

X = X\_test

y\_true = y\_test

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

print(f'Accuracy: {calculate\_accuracy(y\_true, y\_pred)}')

print(f'Class 0 metrics: {calculate\_recall\_precision\_f1(y\_true, y\_pred, 0)}')

print(f'Class 1 metrics: {calculate\_recall\_precision\_f1(y\_true, y\_pred, 1)}')

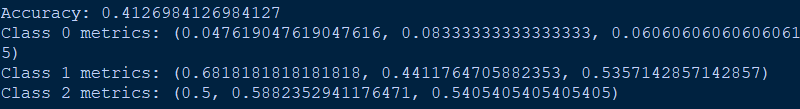
print(f'Class 2 metrics: {calculate\_recall\_precision\_f1(y\_true, y\_pred, 2)}')

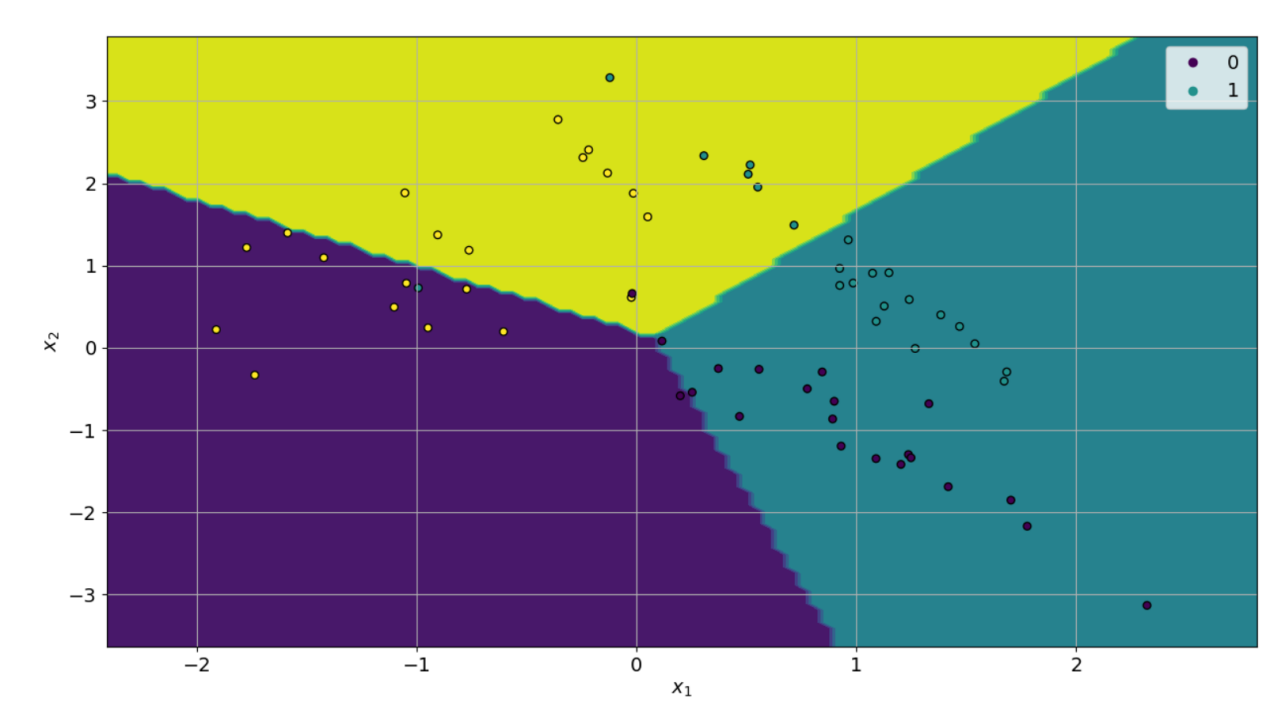
plot\_2d\_decision\_boundary(X, rand\_W, y\_true)

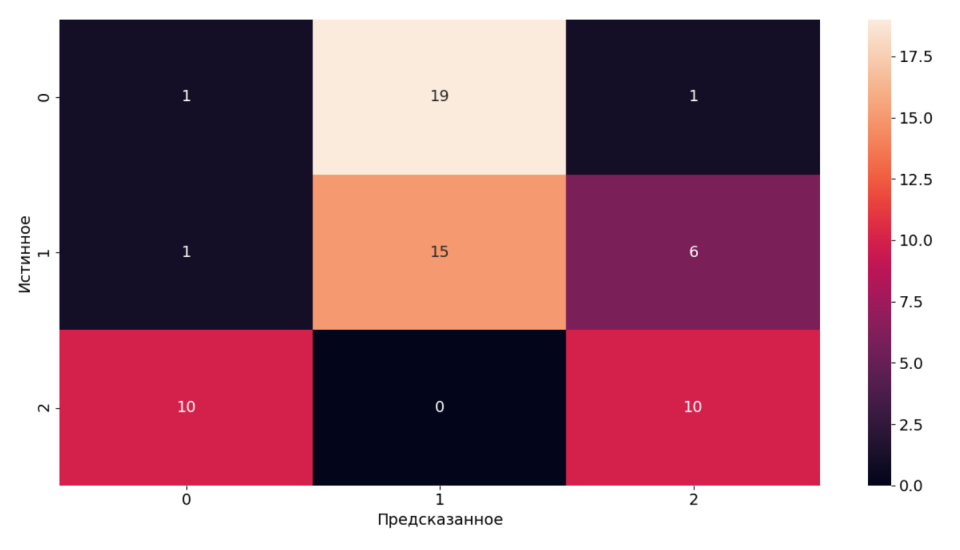
conf\_mtrx = get\_confusion\_matrix(y\_true, y\_pred)

draw\_confusion\_matrix(conf\_mtrx)

Результат







# **Функция потерь**

Суть OHE метода заключается в том, что каждый индекс класс в векторе мы заменяем на вектор, состоящий из нулей и единиц. Делается это путем размещения единицы в векторе на место по индексу номера класса. Количество возможных значений (а в нашем случае - это количество предсказываемых классов) определяет длину вектора:

# TODO - функция кодирования вектора индексов классов в представление OHE

def onehot\_encode(y):

# NOTE - для получения размера вектора воспользуйтесь максимальным значением

# из вектора y

vec\_length=np.max(y)+1

y\_ohe=np.zeros((len(y),vec\_length))

for i in range(len(y)):

y\_ohe[i,y[i]]=1

return y\_ohe

# Проверка

y\_ohe = onehot\_encode([1, 2, 1, 2, 1, 0])

y\_ohe\_true = np.array([

[0, 1, 0],

[0, 0, 1],

[0, 1, 0],

[0, 0, 1],

[0, 1, 0],

[1, 0, 0]

])

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

## **Категориальная кросс-энтропия**

# TODO - функция потерь и функция получения матрицы производных

def cce\_loss(y\_true, y\_pred\_proba):

y\_ohe=onehot\_encode(y\_true)

loss=-np.sum(y\_ohe\*np.log(y\_pred\_proba),axis=1)

loss=np.mean(loss)

return loss

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

y\_ohe=onehot\_encode(y\_true)

dJ=X.T@(y\_pred\_proba-y\_ohe)

return dJ

y\_true = np.array([0, 1, 2, 3])

y\_pred\_proba = np.array([

[0.98, 0.01, 0.01, 1e-8],

[0.6, 0.1, 1e-8, 0.3],

[0.1, 0.2, 0.4, 0.3],

[0.2, 0.1, 0.1, 0.6],

])

X = np.array([

[1, 2, 3],

[2, 3, 4],

[8, 9, 0],

[7, 6, 5],

])

loss = cce\_loss(y\_true, y\_pred\_proba)

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

dJ\_true = np.array([

[ 3.38 , 0.51 , -4.08999998, 0.20000001],

[ 3.86 , -0.28 , -4.77999997, 1.20000002],

[ 3.34 , -3.07 , 0.53000004, -0.79999997]

])

assert loss == 0.9374760389879278

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

# **Обучение**

## **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 сce\_loss\_deriv() более общая и не добавляет колонку единиц.

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

y\_ohe = onehot\_encode(y)

K = y\_ohe.shape[1]

M = X.shape[1]+1

W = np.zeros((M, K))

X\_linreg = np.c\_[np.ones(X.shape[0]), X]

loss\_history = []

for i\_iter in range(n\_iter):

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

loss = cce\_loss(y, y\_pred)

dJ = cce\_loss\_deriv(X\_linreg, y, y\_pred)

W -= lr\*dJ

loss\_history.append(loss)

return W, loss\_history

def show\_loss(loss\_history):

plt.plot(loss\_history)

plt.grid()

plt.title('Loss history')

plt.xlabel('Iteration')

plt.ylabel('$J(X)$')

plt.show()

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

lr=0.01,

n\_iter=100,

X=X\_train,

y=y\_train

)

show\_loss(loss\_history)

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

X = X\_test

y\_true = y\_test

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

print(f'Accuracy: {calculate\_accuracy(y\_true, y\_pred)}')

print(f'Class 0 metrics: {calculate\_recall\_precision\_f1(y\_true, y\_pred, 0)}')

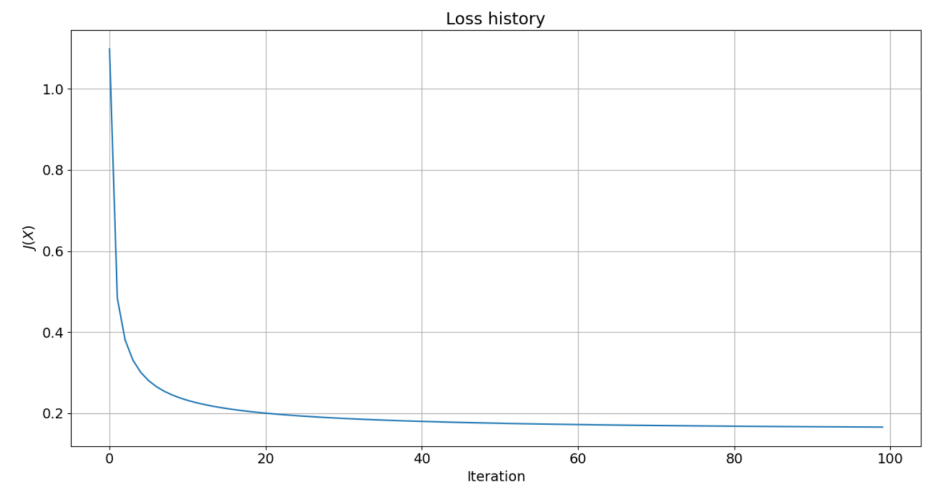
print(f'Class 1 metrics: {calculate\_recall\_precision\_f1(y\_true, y\_pred, 1)}')

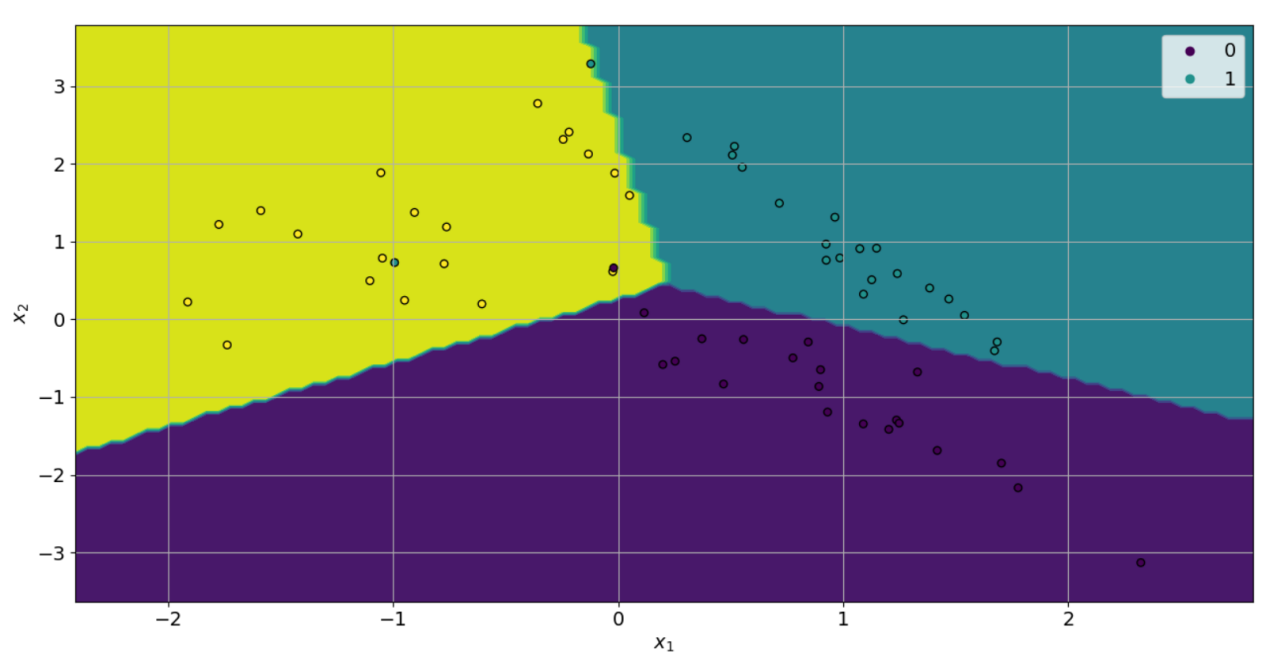
print(f'Class 2 metrics: {calculate\_recall\_precision\_f1(y\_true, y\_pred, 2)}')

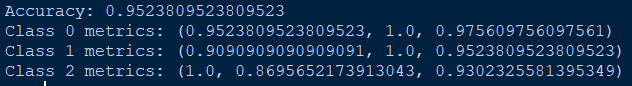
conf\_mtrx = get\_confusion\_matrix(y\_true, y\_pred)

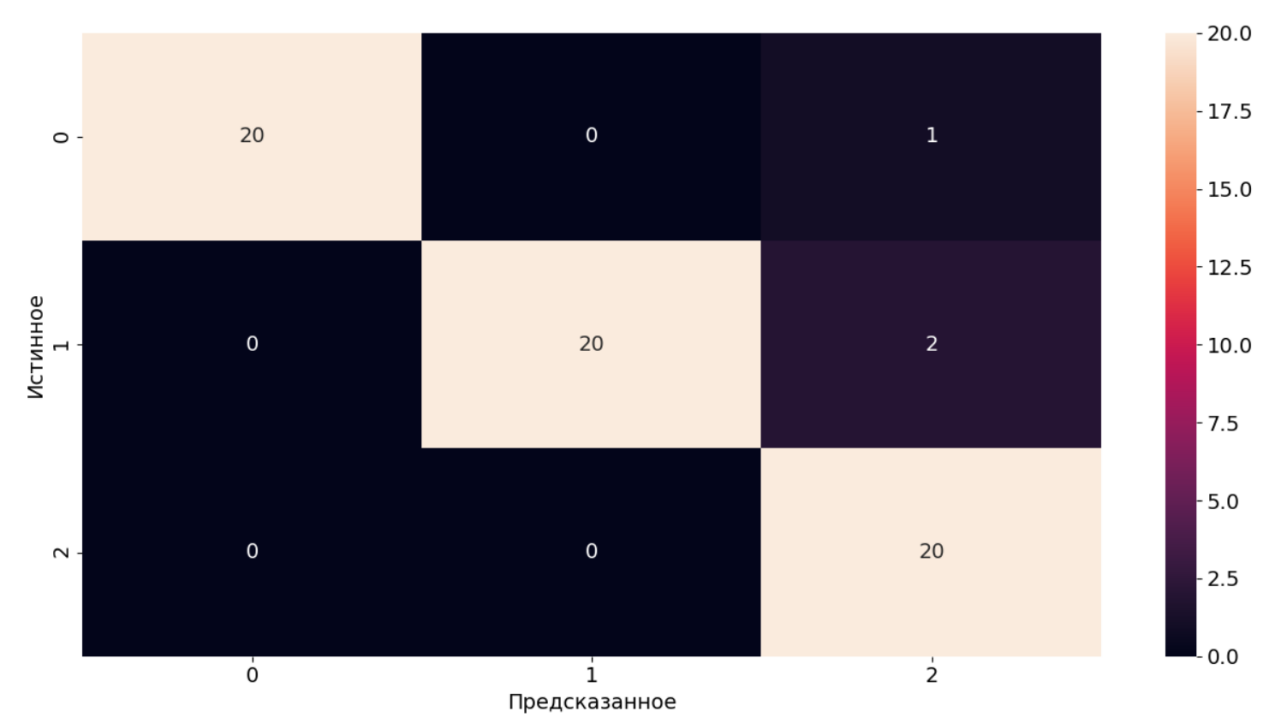
draw\_confusion\_matrix(conf\_mtrx)

Результат









# **Разработка модели с помощью класса**

# TODO - реализуйте класс логистической регрессии

class LogisticRegression:

def \_\_init\_\_(self):

# В начале инициализируем веса как None, что означает "модель не обучена"

self.W = None

def softmax(self,z):

z\_norm=(np.exp(z)/np.sum(np.exp(z)))

return z\_norm

def predict\_proba(self, X):

# Проверяем веса, если они None (через is - так проверяется тип),

# тогда сообщаем об ошибке, так как надо сначала обучить модель!

if self.W is None:

print('Model is not trained!')

return None

# TODO - напишите продолжение предсказания

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

z=X@self.W

y\_proba=np.zeros(z.shape)

for i in range(z.shape[0]):

y\_proba[i,:]=self.softmax(z[i,:])

return y\_proba

def predict(self, X):

# Проверяем веса, если они None (через is - так проверяется тип),

# тогда сообщаем об ошибке, так как надо сначала обучить модель!

if self.W is None:

print('Model is not trained!')

return None

# TODO - напишите продолжение предсказания

y\_proba=self.predict\_proba(X)

y\_pred=np.argmax(y\_proba,axis=1)

return y\_pred

def onehot\_encode(self,y):

# NOTE - для получения размера вектора воспользуйтесь максимальным значением

# из вектора y

vec\_length=np.max(y)+1

y\_ohe=np.zeros((len(y),vec\_length))

for i in range(len(y)):

y\_ohe[i,y[i]]=1

return y\_ohe

def cce\_loss(self,y\_true, y\_pred\_proba):

y\_ohe=self.onehot\_encode(y\_true)

loss=-np.sum(y\_ohe\*np.log(y\_pred\_proba),axis=1)

loss=np.mean(loss)

return loss

def cce\_loss\_deriv(self,X, y\_true, y\_pred\_proba):

y\_ohe=self.onehot\_encode(y\_true)

dJ=X.T@(y\_pred\_proba-y\_ohe)

return dJ

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

# Для начала нужно инициализировать массив весов с размером (M, K)

y\_ohe =self.onehot\_encode(y)

K = y\_ohe.shape[1]

M = X.shape[1]+1

self.W = np.zeros((M, K))

X\_linreg = np.c\_[np.ones(X.shape[0]), X]

loss\_history = []

for i\_iter in range(n\_iter):

y\_pred = self.predict\_proba(X)

loss = self.cce\_loss(y, y\_pred)

dJ = self.cce\_loss\_deriv(X\_linreg, y, y\_pred)

self.W=self.W-lr\*dJ

loss\_history.append(loss)

# TODO - затем произвести обучения

# NOTE - возвращать веса не нужно, они являются атрибутом класса

# и должны храниться внутри модели

logreg\_model = LogisticRegression()

logreg\_model.fit(

X\_train, y\_train,

lr=0.01,

n\_iter=100

)

y\_pred = logreg\_model.predict(X\_test)

print(f'Accuracy: {calculate\_accuracy(y\_test, y\_pred)}')

print(f'Class 0 metrics: {calculate\_recall\_precision\_f1(y\_test, y\_pred, 0)}')

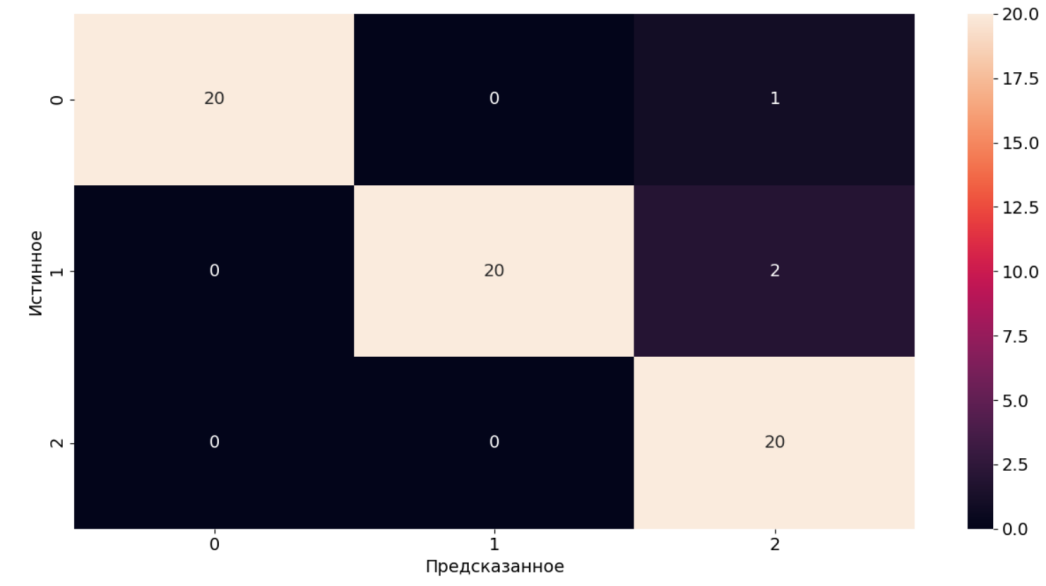
print(f'Class 1 metrics: {calculate\_recall\_precision\_f1(y\_test, y\_pred, 1)}')

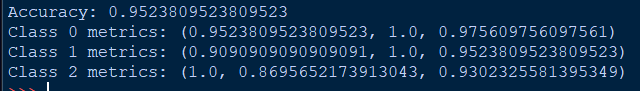
print(f'Class 2 metrics: {calculate\_recall\_precision\_f1(y\_test, y\_pred, 2)}')

conf\_mtrx = get\_confusion\_matrix(y\_test, y\_pred)

draw\_confusion\_matrix(conf\_mtrx)

Результат





## **Задание**

Произведите оценку работы модели с использованием кросс-валидации и класса LogisticRegression. В качестве метрики для оценки используйте среднюю F1 по всем классам.

# TODO - функция оценки работы модели кросс-валидацией с пятью фолдами

# NOTE - не забудьте зафиксировать RANDOM\_STATE

from sklearn.model\_selection import StratifiedKFold

# NOTE - обратите внимание, в качестве агрумента передаем модель,

# которую будем проверять

def cross\_val\_score\_f1(X, y, k\_folds, model):

f1\_values = []

skf = StratifiedKFold(n\_splits=k\_folds,random\_state=RANDOM\_STATE)

for train\_index, test\_index in skf.split(X,y):

X\_train, X\_test = X[train\_index], X[test\_index]

y\_train, y\_test = y[train\_index], y[test\_index]

loss\_history = model.fit(

lr=0.01,

n\_iter=100,

X=X\_train,

y=y\_train)

y\_pred = model.predict(X\_test)

for i in range(np.max(y\_test)+1):

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

f1\_values.append(f1)

return np.mean(f1\_values)

# Используем!

logreg\_model = LogisticRegression()

f1\_result = cross\_val\_score\_f1(X\_data, y\_data, k\_folds=5, model=logreg\_model)

print(f1\_result)

Результат



# ****Выводы - Задание****

1. ****В каких случаях необходимо использовать мультиноминальную классификацию?**** Мультиномиальная классификация используется, когда необходимо разделить данные на несколько доступных классов, при этом каждому элементу нужно присвоить лишь 1 класс.
2. ****Что позволяет оценить функция потерь? И зачем нужна эта оценка?**** Функция потерь позволяет оценить как далеко от истины находится вектор предсказаний. Необходимо свести функцию потерь к минимуму.
3. ****Как это "обучение с учителем"?**** Обучение с учителем это способ обучения модели, когда создается специальная обучающая выборка, представляющая собой желаемый выход модели. Цель обучения свести ошибку обучения (разность между желаемым и действительным выходом) к минимуму.

# ****Вопросики!****

1. ****Что используется в мультиноминальной классификации вместо сигмоиды?**** Используется функция Softmax.
2. ****Почему MSE для случае мутиноминальной классификации - не лучший выбор?**** У MSE расстояние между индексами 0 и 2 будут больше, чем расстояние между 0 и 1, что вносит искажение в смысл, так как классы по факту должны быть равнозначны.
3. ****Что такое ОНЕ и где это использовать?**** OHE это метод, согласно которому индекс каждого класса заменяется на вектор из 0 и 1, длина вектора определяется количеством классов, 1 ставится на место по индексу номера класса. Используется для кодирования классов, когда нет порядковых отношений между классами и необходимо все классы сделать равнозначными.
4. ****В чём категориальность категориальной кросс-энтропии?**** Категориальность состоит в том, что она применяется как функция потерь для множества классов.
5. ****Зачем оборачивать модель логистической регрессии в класс? Какие преимущества это даёт?**** Класс создается для удобства использования необходимых функций без необходимости каждый раз передавать в эти фунции параметры, которые используются только внутри этих функций.