МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Автоматизация схемотехнического проектирования»
на тему «Классификатор на основе логистической регрессии с градиентным спуском»

Студенты гр. 1301 _____ Семейкин С.А.

_____ Гальченко М.А.

Преподаватель _____ Боброва Ю.О.

Санкт-Петербург
2025

Цель:

Разработка модели классификатора на основе логистической регрессии, изучение его свойств и принципов работы, получение навыков программирования на Python и использования модуля scikit-learn.

Ход работы:

1. Создадим переменные, распределение по нормальному закону с незначительно различными средними и дисперсиями.
2. Создадим переменные, соответствующие классам.
3. Обучим классификатор (например, логистическую регрессию) на обучающем наборе данных.
4. Визуализируем распределение вероятностей для обучающей и тестовой выборок с помощью гистограмм.
5. Оценим эффективность классификатора на тестовом наборе данных.

При:

mu0 = [0, 2] mu1 = [3, 5] sigma0 = [2, 1] sigma1 = [1, 2]

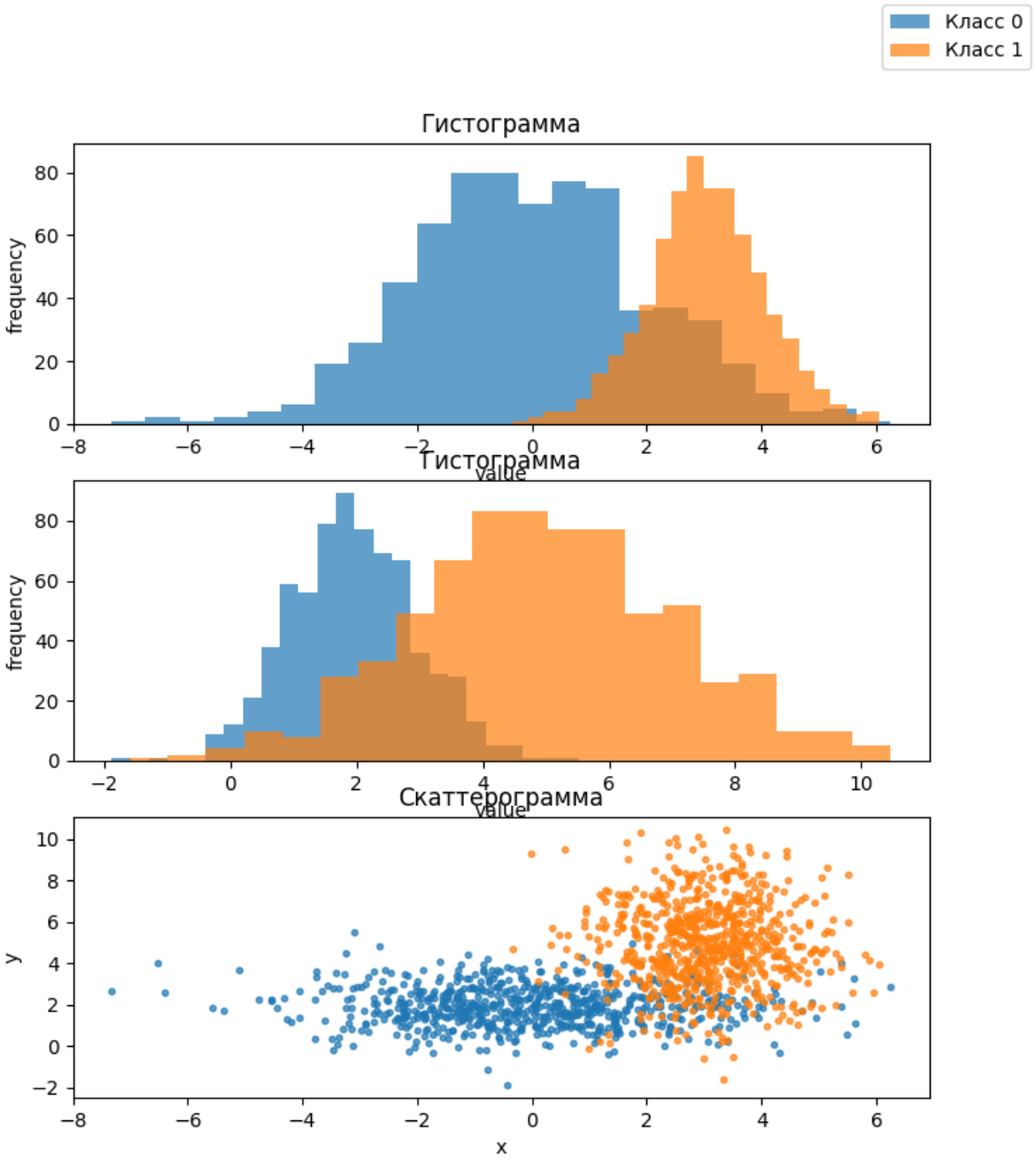

Рисунок 1 Распределение классов

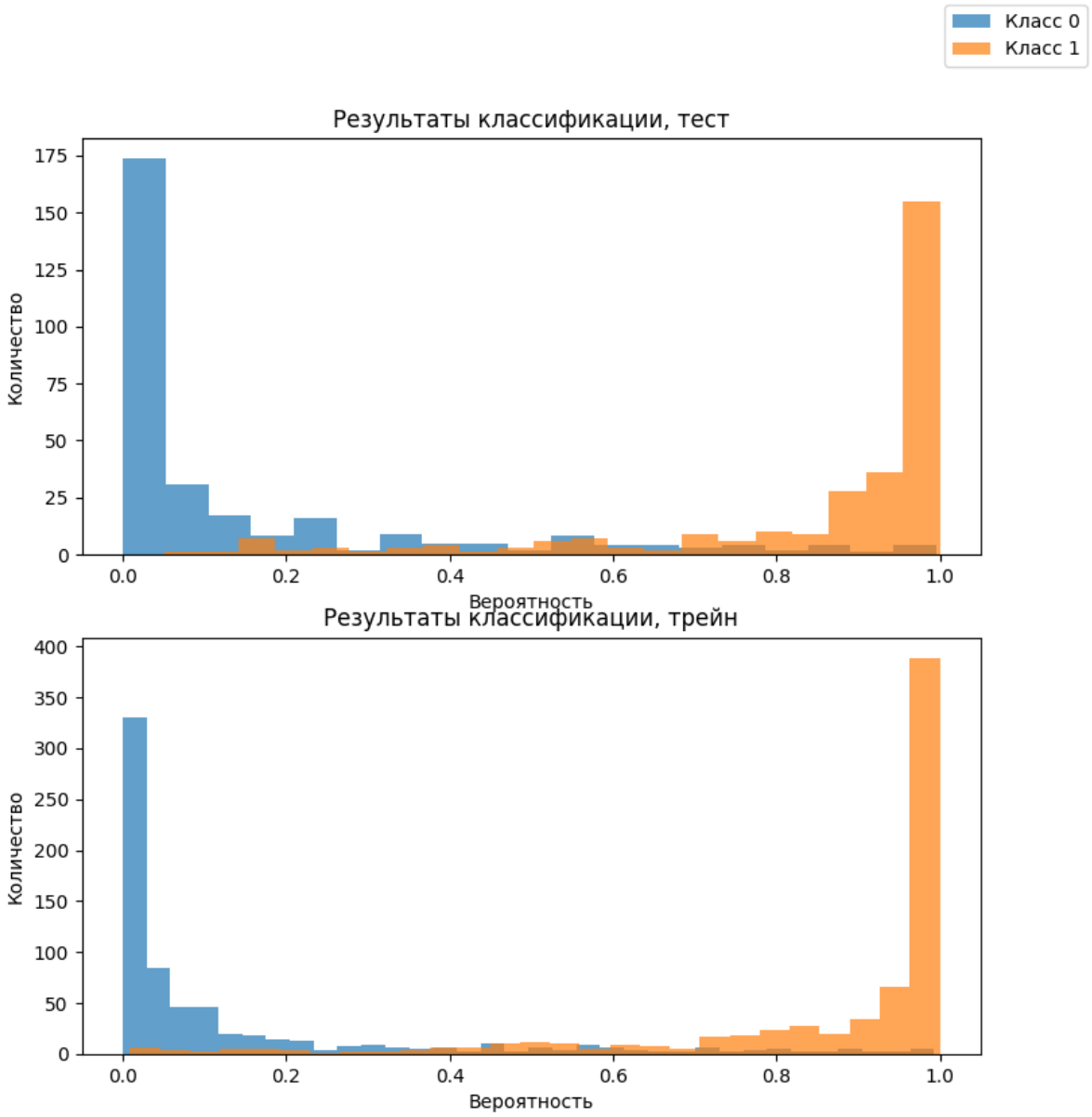

Рисунок 2 — Вероятности принадлежности объектов классам для обучающей и тестовой выборок

	Число объектов	Точность, %	Чувствительность, %	Специфичность, %
Train	1400	90.16	90.96	89.36
Test	600	91.92	91.72	92.13

При более плотном пересечении
mu0 = [0, 2] mu1 = [1, 3] sigma0 = [2, 2] sigma1 = [2, 2]

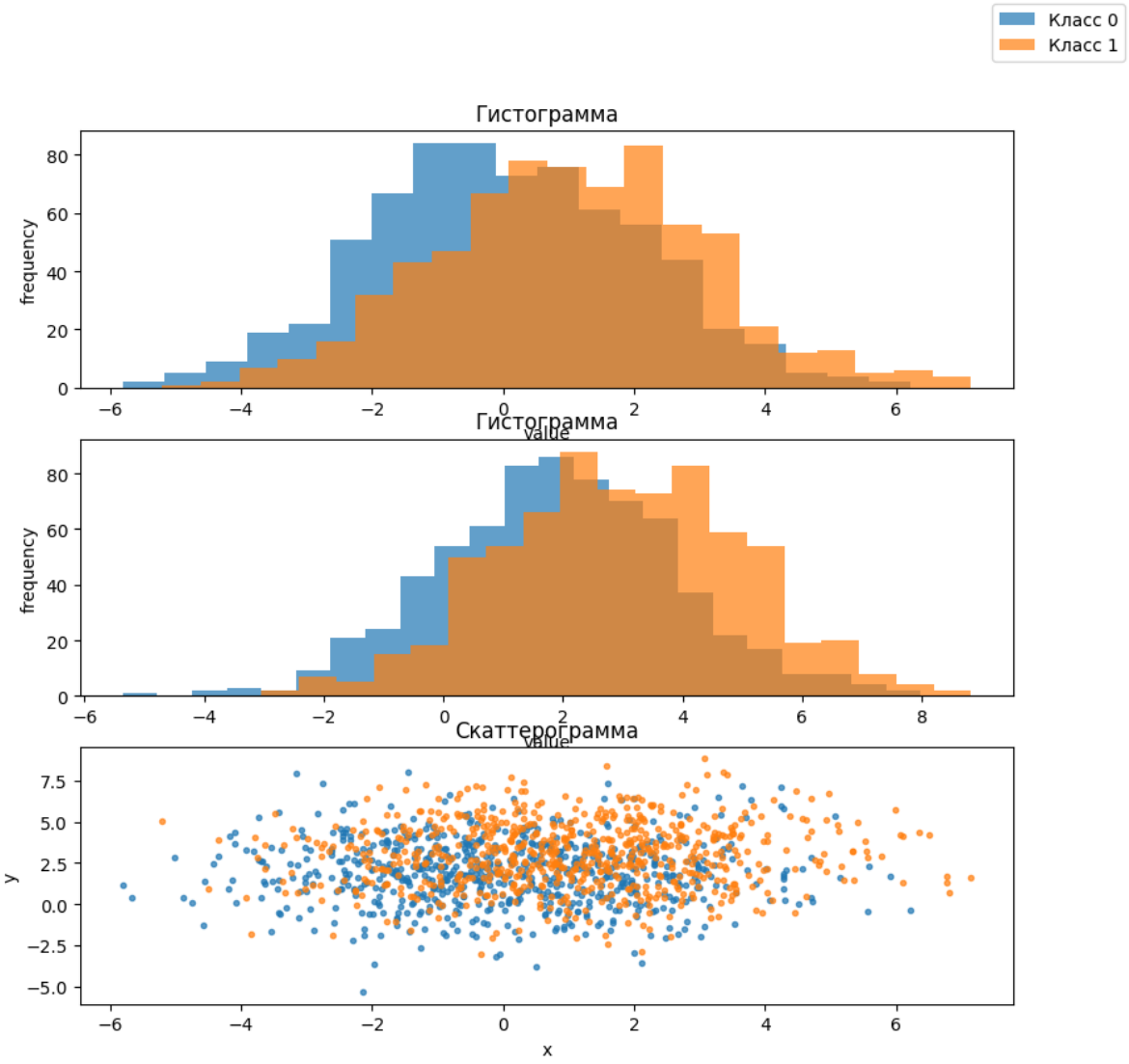

Рисунок 3 Распределение классов

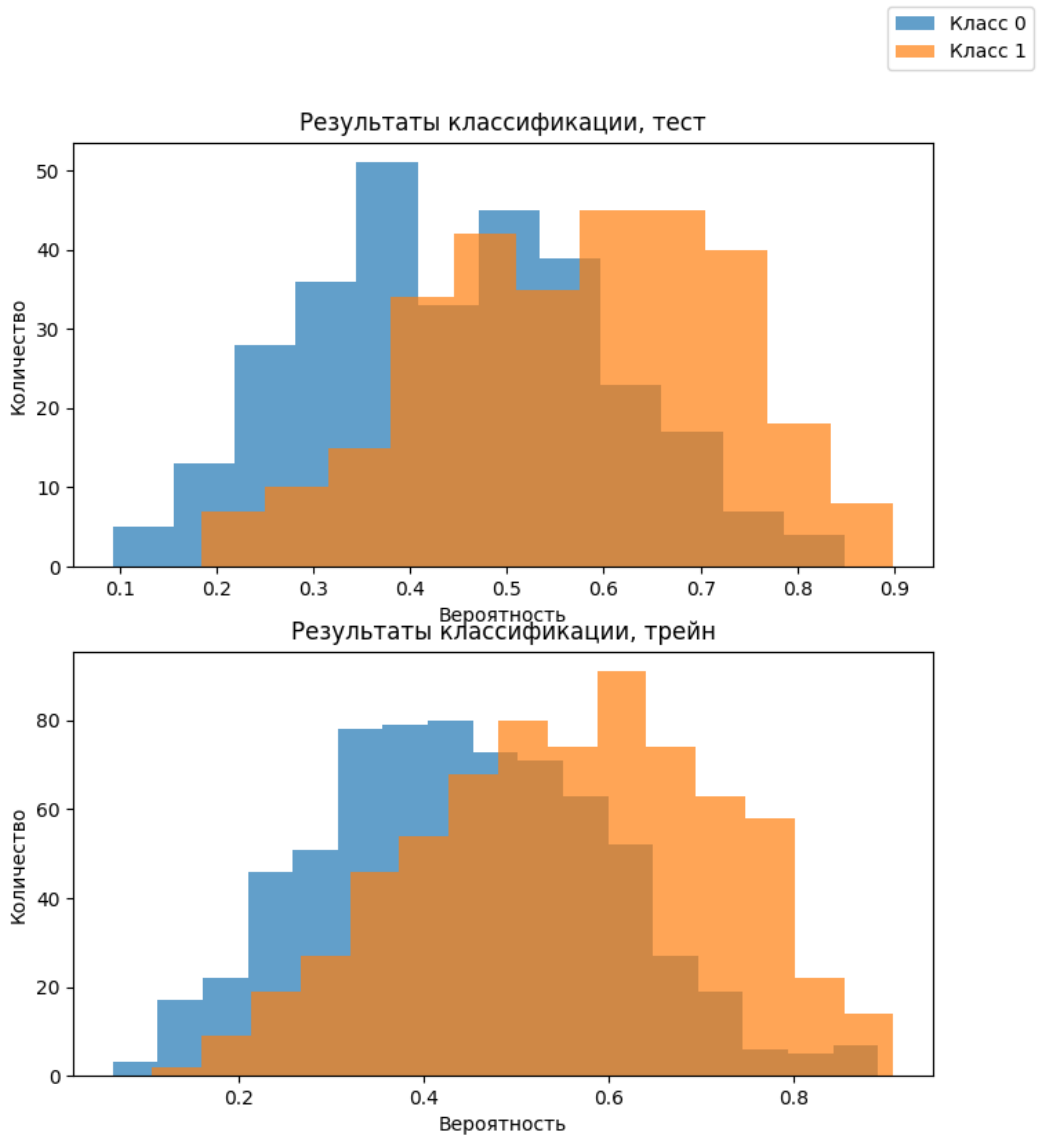

Рисунок 4 — Вероятности принадлежности объектов классам при более плотном пересечении для обучающей и тестовой выборок

	Число объектов	Точность, %	Чувствительность, %	Специфичность, %
Train	1400	63.85	64.05	63.66
Test	600	64.66	66.88	62.45

При нелинейно пересекаемой выборке:

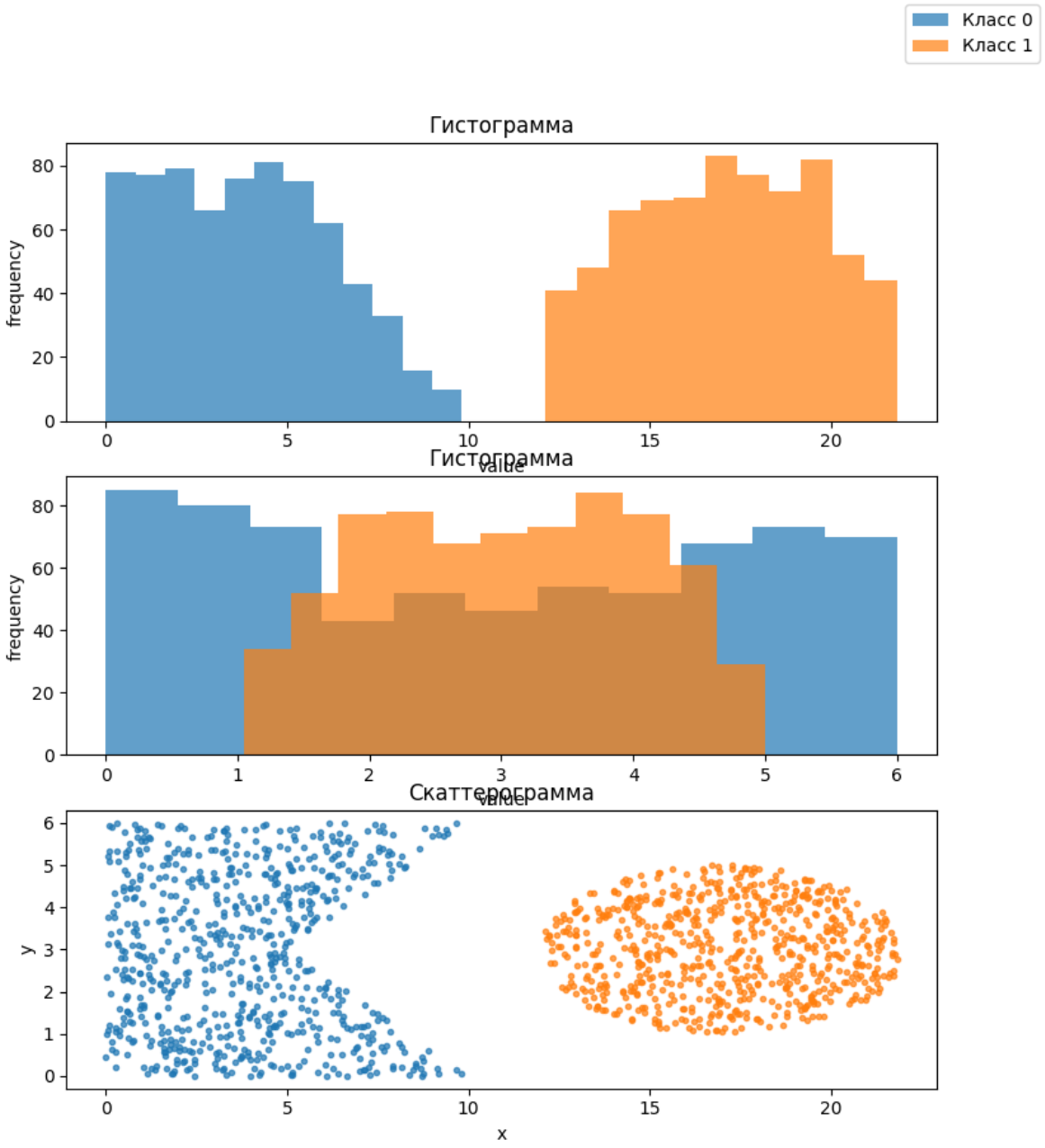

Рисунок 5 Распределение классов

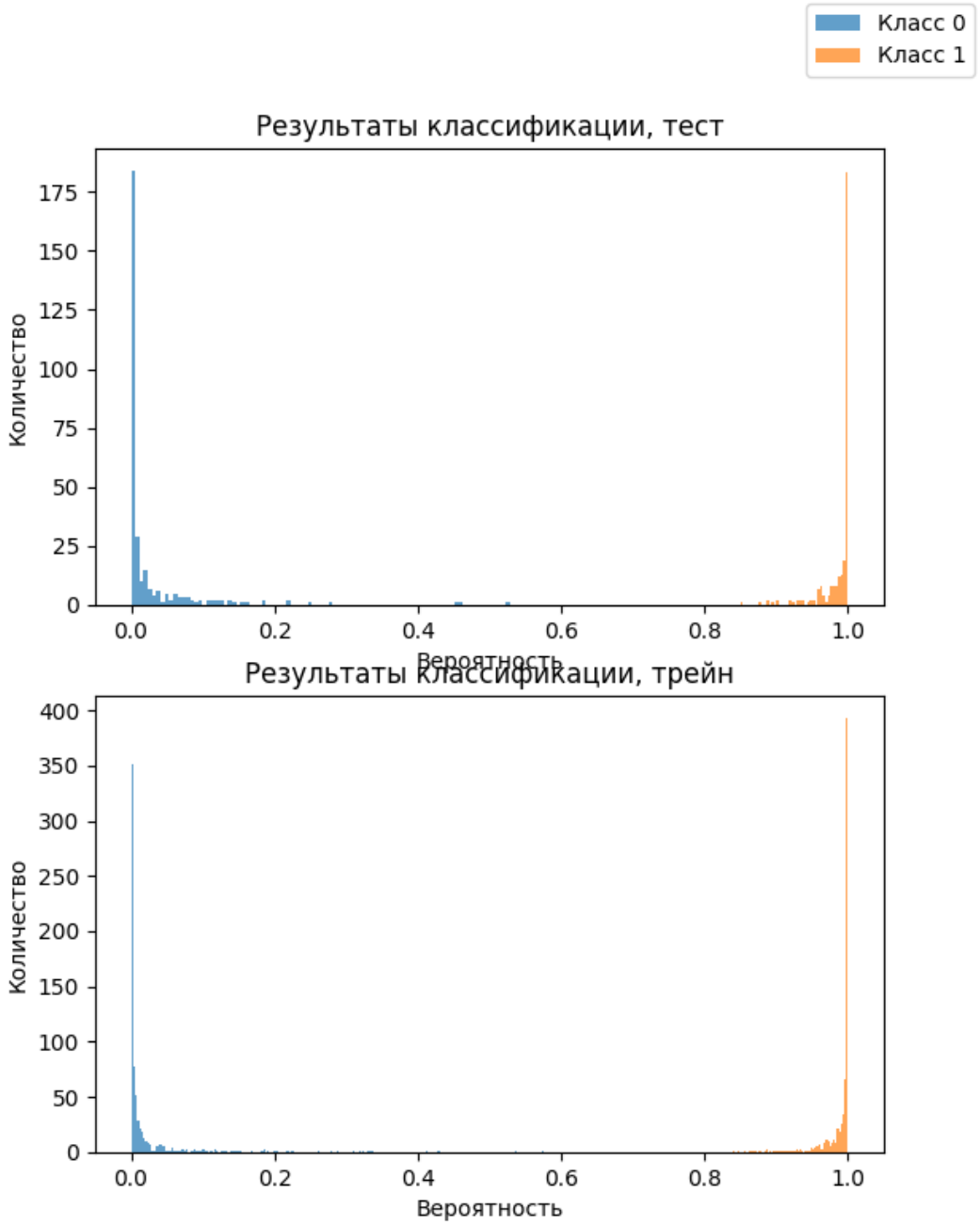

Рисунок 6 Вероятности принадлежности объектов классам при более плотном пересечении для обучающей и тестовой выборок

	Число объектов	Точность, %	Чувствительность, %	Специфичность, %
Train	1400	99.85	100	99.71
Test	600	99.83	100	99.67

Вывод:

В ходе лабораторной работы был разработана модель классификатора на основе логистической регрессии, изучены его свойства и принципы работы, получены навыки программирования на Python и использования модуля scikit-learn.

Изменяя параметры при генерации данных, мы получили более или менее пересекающиеся классы и оценили, как это влияет на эффективность классификатора — при более плотном пересечении классов точность классификатора ухудшается.

Это связано с тем, что при плотном пересечении классов граница между классами становится менее очевидной. Модель сталкивается с трудностями в поиске правильного разделения пространства, что приводит к увеличению ошибок классификации. Кроме того, может требоваться больше обучающих данных для построения более точной модели.

При нелинейно пересекаемой выборке значения метрик модели стремились к 100%, что означает ее хорошую работу, ведь в данных явно просматривается принадлежность к разным классам.

Листинг:

```
import numpy as np
from pathlib import Path

from sklearn.linear_model import LogisticRegression
from lab1.data_generator import DataGenerator
import matplotlib.pyplot as plt


def calculate_metrics(prediction, answers):
    true_positive_test = np.sum((prediction == 1) & (answers == 1))
    true_negative_test = np.sum((prediction == 0) & (answers == 0))
    false_positive_test = np.sum((prediction == 1) & (answers == 0))
    false_negative_test = np.sum((prediction == 0) & (answers == 1))
    sensivity = true_positive_test / (true_positive_test +
false_negative_test)
    specifity = true_negative_test / (true_negative_test +
false_positive_test)
    accuracy = sum(prediction == answers) / len(answers)

    return sensivity, specifity, accuracy


with_graphs = True
x_train_path = Path("x_train.npy")
x_test_path = Path("x_test.npy")
y_train_path = Path("y_train.npy")
y_test_path = Path("y_test.npy")

path_to_files = [x_train_path,
                 x_test_path,
                 y_train_path,
                 y_test_path
                 ]
mu0 = [0, 2]
mu1 = [1, 3]
sigma0 = [2, 2]
sigma1 = [2, 2]
col = len(mu0)
N = 1000

if not all([e.exists() for e in path_to_files]):

    # X, Y, *_ = DataGenerator.norm_dataset((mu0, mu1), (sigma0, sigma1), N)
    X, Y, *_ = DataGenerator.nonlinear_dataset_N(N)

    train_size = 0.7
    trainCount = round(train_size * N * 2)  # *2 потому что было 2 класса
    Xtrain = X[0:trainCount]
    Xtest = X[trainCount:N * 2 + 1]
    Ytrain = Y[0:trainCount]
    Ytest = Y[trainCount:N * 2 + 1]
    np.save(x_train_path, Xtrain)
    np.save(x_test_path, Xtest)
    np.save(y_train_path, Ytrain)
    np.save(y_test_path, Ytest)
else:
    Xtrain = np.load(x_train_path)
    Xtest = np.load(x_test_path)
    Ytrain = np.load(y_train_path)
    Ytest = np.load(y_test_path)
# print(Xtrain)
# print(Xtrain.shape)
# print(Ytrain)
```

```
# print(Ytrain.shape)
# print(Xtest)
# print(Xtest.shape)
# print(Ytest)
# print(Ytest.shape)
"""
[[ 2.89062473  5.12569201]
 [ 0.0512457   1.35763416]
 [-1.29076494  2.22371003]
 ...
 [-0.49192833  2.9665729 ]
 [-0.03987324  3.53267207]
 [ 1.10493208  3.17819484]]
(1400, 2)
[ True False False ... False  True  True]
(1400,)
[[ 1.2702289   7.55313551]
 [ 3.45246334  2.21036868]
 [ 0.56758523  5.41047317]
 ...
 [ 0.35327595  4.96346211]
 [-0.42812353  3.71269175]
 [ 2.276523    4.46411915]]
(600, 2)
(600,)
"""
if with_graphs:
    X = Xtrain
    figure, axis = plt.subplots(3)
    axis[0].set_title(f"Гистограмма")
    axis[0].hist(X[Ytrain == 0][:, 0], bins='auto', alpha=0.7)
    axis[0].hist(X[Ytrain == 1][:, 0], bins='auto', alpha=0.7)
    axis[0].set_xlabel("value")
    axis[0].set_ylabel("frequency")
    axis[1].set_title(f"Гистограмма")
    axis[1].hist(X[Ytrain == 0][:, 1], bins='auto', alpha=0.7)
    axis[1].hist(X[Ytrain == 1][:, 1], bins='auto', alpha=0.7)
    axis[1].set_xlabel("value")
    axis[1].set_ylabel("frequency")
    axis[2].set_xlabel("x")
    axis[2].set_ylabel("y")
    axis[2].set_title(f"Скаттерограмма")
    axis[2].scatter(X[Ytrain == 0][:, 0], X[Ytrain == 0][:, 1], marker=".",
alpha=0.7)
    axis[2].scatter(X[Ytrain == 1][:, 0], X[Ytrain == 1][:, 1], marker=".",
alpha=0.7)

    figure.legend(['Класс 0', 'Класс 1'])
    plt.show()

Nvar = 18
clf = LogisticRegression(random_state=Nvar, solver='saga').fit(Xtrain,
Ytrain)

pred_test = clf.predict(Xtest)
# print(pred test)
pred_test_proba = clf.predict_proba(Xtest)
print(pred_test_proba)


pred_train = clf.predict(Xtrain)
pred_train_proba = clf.predict_proba(Xtrain)
# print(pred_test_proba)
```

```
acc_train = clf.score(Xtrain, Ytrain)
# print(acc train)
acc_test = clf.score(Xtest, Ytest)
# print(acc_test)
# acc_test = sum(pred_test == Ytest) / len(Ytest)
# print(acc test)
# from sklearn.calibration import calibration curve
# print(Ytest.shape)
# y_means, proba_means = calibration_curve(Ytest, pred_test_proba, n_bins=10)
if with_graphs:
    figure, axis = plt.subplots(2)
    axis[0].hist(pred_test_proba[~Ytest, 1], bins='auto', alpha=0.7)
    axis[0].hist(pred_test_proba[Ytest, 1], bins='auto', alpha=0.7)

    axis[0].set_xlabel("Вероятность")
    axis[0].set_ylabel("Количество")
    axis[0].set_title("Результаты классификации, тест")

    axis[1].hist(pred_train_proba[~Ytrain, 1], bins='auto', alpha=0.7)
    axis[1].hist(pred_train_proba[Ytrain, 1], bins='auto', alpha=0.7)

    axis[1].set_xlabel("Вероятность")
    axis[1].set_ylabel("Количество")
    axis[1].set_title("Результаты классификации, трейн")
    figure.legend(['Класс 0', 'Класс 1'])
    plt.show()
print(Ytrain.size)
print(calculate_metrics(pred_train, Ytrain))
print(Ytest.size)
print(calculate_metrics(pred_test, Ytest))
```