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»

Тема: КЛАССИФИКАТОР НА ОСНОВЕ ЛОГИСТИЧЕСКОЙ РЕГРЕССИИ С ГРАДИЕНТНЫМ СПУСКОМ

Студент гр. 1302	_____	Новиков Г.В.
Студентка гр. 1302	_____	Романова О.В.
Студентка гр. 1302	_____	Марзаева В.И.
Преподаватель	_____	Боброва Ю.О.

Санкт-Петербург

2025

Цель работы

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

Основные теоретические положения

Результат работы классификатора $\hat{y}$ – вероятность наступления события для объекта X, где X – вектор-строка {x1, x2… xn} в n-мерном пространстве, рассчитывается по формуле сигмоиды:

$$\hat{y} = \frac{1}{1 + e^{-z}}, \quad Z = b_0 + b_1 x_1 + b_2 x_2 + \cdots b_n x_n \tag{2.1}$$

Метод логистической регрессии относится к методам обучения с учителем. Под обучением понимается оптимизационный алгоритм, минимизирующий ошибку предсказания. В результате обучения модели формируется набор коэффициентов, который используется для предсказаний на новых данных. Ошибка предсказания описывается с помощью функции потерь (loss function). Она может определяться разными способами, наиболее часто используемой является следующая функция:

$$loss = -\frac{1}{m}\sum_{i=1}^{m} \log(\hat{y}_i) * y_i - (1 - y_i) * \log(1 - \hat{y}_i), \tag{2.2}$$

где m – число объектов в выборке, yi – действительное значение класса i-го объекта (0 или 1), $\hat{y}i$ – выход сигмоидальной функции (результат предсказания, от 0 до 1). Метод градиентного спуска – один из методов обучения модели. Более подробно о нем можно узнать в материалах курса.

Чувствительность (истинно положительная пропорция) отражает долю положительных результатов, которые правильно идентифицированы как таковые. Иными словами, чувствительность диагностического теста

показывает вероятность того, что больной субъект будет классифицирован именно как больной. Специфичность (истинно отрицательная пропорция) отражает долю отрицательных результатов, которые правильно идентифицированы как таковые, то есть вероятность того, что не больные субъекты будут классифицированы именно как не больные.

Библиотека scikit-learn – одна из самых популярных библиотек для анализа данных. Она включает сотни функций, покрывающие почти все задачи базового анализа. В частности, в библиотеке реализован метод логрегрессии с возможностью настройки гиперпараметров модели, а также выбора оптимизационного алгоритма.

Для установки scikit-learn надо выполнить в командной строке:

pip install -U scikit-learn

Данная библиотека ориентирована на моделирование данных. Она не ориентирована на их загрузку, манипулирование и суммирование. Для решения этих задач, как правило, применяются библиотеки NumPy и Pandas.

Ход работы

1. *Полный код программы:*

lab2.py:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from data_generator import generate_dataset_A, generate_dataset_B, generate_dataset_C


N = 1000                                # число объектов класса
col = 3

X, Y, class0, class1 = generate_dataset_A(N)          # linear good
```

```
# X, Y, class0, class1 = generate_dataset_B(N)            # linear bad
# X, Y, class0, class1 = generate_dataset_C(N)             # non-linear

# разделяем данные на 2 подвыборки
trainCount = round(0.7*N*2)
Xtrain = X[1:trainCount]
Xtest = X[trainCount:N*2+1]
Ytrain = Y[0:trainCount]
Ytest = Y[trainCount:N*2+1]

for i in range(0, col):
    # построение одной скатеррограммы по выбранным признакам
    plt.scatter(class0[:, i], class0[:, (i + 1) % col], marker=".", alpha=0.7)
    plt.scatter(class1[:, i], class1[:, (i + 1) % col], marker=".", alpha=0.7)
    plt.title('Scatter')
    plt.xlabel('Parameter ' + str(i))
    plt.ylabel('Parameter ' + str((i + 1) % col))
    plt.savefig('scatter_' + str(i + 1) + '.png')
    plt.show()

clf = LogisticRegression(random_state=13, solver='saga').fit(Xtrain, Ytrain)

Pred_test = clf.predict(Xtest)                    # Predict class labels for samples in X.
Pred_test_proba = clf.predict_proba(Xtest)              # Probability estimates

acc_train = clf.score(Xtrain, Ytrain)
acc_test = clf.score(Xtest, Ytest)
# acc_test = sum(Pred_test==Ytest)/len(Ytest)

print('Accuracy train:', acc_train)
print('Accuracy test:', acc_test)

plt.hist(Pred_test_proba[Ytest,1], bins='auto', alpha=0.7)
```

```
plt.hist(Pred_test_proba[~Ytest,1], bins='auto', alpha=0.7) # т.к массив свероятностями имеет два столбца, мы берем один - первый
plt.title("Результаты классификации, тест")
plt.savefig('classification_res' + '.png')
plt.show()

sensitivity = 0
specificity = 0
for i in range(len(Pred_test)):
    if Pred_test[i]==True and Ytest[i]==True:
        sensitivity += 1

    if Pred_test[i]==False and Ytest[i]==False:
        specificity += 1

sensitivity /= len(Pred_test[Pred_test==True])
specificity /= len(Pred_test[Pred_test==False])

print('Sensitivity:', sensitivity)
print('Specificity:', specificity)
```

data_generator.py:

```
import numpy as np



def norm_dataset(mu,sigma,N):
    mu0 = mu[0]
    mu1 = mu[1]
    sigma0 = sigma[0]
    sigma1 = sigma[1]

    col = len(mu0)                          # количество столбцов-признаков – длина массива средних
```

```
    class0 = np.random.normal(mu0[0], sigma0[0], [N, 1])      # инициализируем первый столбец (в Python нумерация от 0)
    class1 = np.random.normal(mu1[0], sigma1[0], [N, 1])
    for i in range(1, col):
        v0 = np.random.normal(mu0[i], sigma0[i], [N, 1])
        class0 = np.hstack((class0, v0))

        v1 = np.random.normal(mu1[i], sigma1[i], [N, 1])
        class1 = np.hstack((class1, v1))

    Y1 = np.ones((N, 1), dtype=bool)
    Y0 = np.zeros((N, 1), dtype=bool)

    X = np.vstack((class0, class1))
    Y = np.vstack((Y0, Y1)).ravel()                    # ravel позволяет сделать массив плоским – одномерным, размера (N,)

    # перемешиваем данные
    rng = np.random.default_rng()
    arr = np.arange(2*N)                               # индексы для перемешивания
    rng.shuffle(arr)
    X = X[arr]
    Y = Y[arr]

    return X, Y, class0, class1


def nonlinear_dataset_13(cen0, cen1, radii0, radii1, N):
    col = len(cen0)
    theta = 2 * np.pi * np.random.rand(N)
    theta = theta[:, np.newaxis]

    class0 = np.empty((N, col))
    class1 = np.empty((N, col))
```

```
    r = radii0[0] + np.random.rand(N)
    r = r[:, np.newaxis]
    class0[:, 0] = (r * np.sin(theta) + cen0[0]).flatten()

    r = radii1[0] + np.random.rand(N)
    r = r[:, np.newaxis]
    class1[:, 0] = (r * np.sin(theta) + cen1[0]).flatten()

    for i in range(1, col):
        r = radii0[i] + np.random.rand(N)
        r = r[:, np.newaxis]
        class0[:, i] = (r * np.cos(theta) + cen0[i]).flatten()

        r = radii1[i] + np.random.rand(N)
        r = r[:, np.newaxis]
        class1[:, i] = (r * np.cos(theta) + cen1[i]).flatten()

    Y1 = np.ones((N, 1), dtype=bool)
    Y0 = np.zeros((N, 1), dtype=bool)

    X = np.vstack((class0, class1))
    Y = np.vstack((Y0, Y1)).ravel()                    # ravel позволяет сделать массив плоским – одномерным,
размера (N,)

    # перемешиваем данные
    rng = np.random.default_rng()
    arr = np.arange(2*N)                               # индексы для перемешивания
    rng.shuffle(arr)
    X = X[arr]
    Y = Y[arr]

    return X, Y, class0, class1
```

```
def generate_dataset_A(N: int):
    mu0 = [0, 2, 3]
    mu1 = [3, 5, 1]
    sigma0 = [2, 1, 2]
    sigma1 = [1, 2, 1]

    mu = [mu0, mu1]
    sigma = [sigma0, sigma1]
    return norm_dataset(mu, sigma, N)


def generate_dataset_B(N: int):
    mu0 = [3, 4, 3]
    mu1 = [3, 5, 2]
    sigma0 = [2, 1, 2]
    sigma1 = [1, 2, 1]

    mu = [mu0, mu1]
    sigma = [sigma0, sigma1]
    return norm_dataset(mu, sigma, N)


def generate_dataset_C(N: int):
    cen0 = [0, 0, 0]
    cen1 = [0, 0, 0]
    radii0 = [6, 1, 2]
    radii1 = [2, 6, 1]

    return nonlinear_dataset_13(cen0, cen1, radii0, radii1, N)
```

2. Пояснения к коду:

Файл lab2.py:

Импорт библиотек:

numpy — библиотека для работы с массивами и математическими операциями.

matplotlib.pyplot — библиотека для визуализации данных (построение графиков).

LogisticRegression из sklearn.linear_model — модель логистической регрессии для классификации.

Функции generate_dataset_A, generate_dataset_B, generate_dataset_C из модуля data_generator — генераторы данных.

Генерация данных:

N = 1000 — количество объектов в каждом классе.

col = 3 — количество признаков (столбцов) в данных.

X, Y, class0, class1 = generate_dataset_A(N) — генерация данных с помощью функции generate_dataset_A. Данные разделяются на два класса (class0 и class1), а X и Y — это объединенные данные и метки классов соответственно.

Закомментированные строки generate_dataset_B и generate_dataset_C позволяют использовать другие наборы данных.

Разделение данных на обучающую и тестовую выборки:

trainCount = round(0.7*N*2) — 70% данных используются для обучения, остальные 30% — для тестирования.

Xtrain, Xtest, Ytrain, Ytest — разделение данных на обучающую и тестовую выборки.

Визуализация данных:

Цикл for i in range(0, col) строит scatter-графики (диаграммы рассеяния) для каждой пары признаков. Графики сохраняются в файлы scatter_1.png, scatter_2.png, scatter_3.png.

Обучение модели логистической регрессии:

clf = LogisticRegression(random_state=13, solver='saga').fit(Xtrain, Ytrain) — создание и обучение модели логистической регрессии на обучающих данных.

Pred_test = clf.predict(Xtest) — предсказание меток классов для тестовых данных.

Pred_test_proba = clf.predict_proba(Xtest) — предсказание вероятностей принадлежности к классам.

Оценка точности модели:

acc_train = clf.score(Xtrain, Ytrain) — точность модели на обучающей выборке.

acc_test = clf.score(Xtest, Ytest) — точность модели на тестовой выборке.

Результаты выводятся в консоль.

Визуализация результатов классификации:

Строится гистограмма вероятностей для каждого класса на тестовых данных. Гистограмма сохраняется в файл classification_res.png.

Расчет чувствительности (sensitivity) и специфичности (specificity):

```
sensitivity = 0
specificity = 0
for i in range(len(Pred_test)):
    if Pred_test[i]==True and Ytest[i]==True:
        sensitivity += 1

    if Pred_test[i]==False and Ytest[i]==False:
        specificity += 1

sensitivity /= len(Pred_test[Pred_test==True])
```

specificity /= len(Pred_test[Pred_test==False])

Чувствительность — это доля правильно предсказанных положительных примеров.

Специфичность — это доля правильно предсказанных отрицательных примеров.

Результаты выводятся в консоль.

Файл data_generator.py:

Функция norm_dataset:

Генерирует данные для двух классов на основе нормального распределения.

mu0, mu1 — средние значения для каждого класса.

sigma0, sigma1 — стандартные отклонения для каждого класса.

class0, class1 — данные для каждого класса.

Y1, Y0 — метки классов (1 для первого класса, 0 для второго).

Данные объединяются, перемешиваются и возвращаются в виде X, Y, class0, class1.

Функция nonlinear_dataset_13:

Генерирует нелинейные данные для двух классов на основе полярных координат.

cen0, cen1 — центры для каждого класса.

radii0, radii1 — радиусы для каждого класса.

Данные генерируются с использованием тригонометрических функций (sin, cos).

Данные объединяются, перемешиваются и возвращаются в виде X, Y, class0, class1.

Функции generate_dataset_A, generate_dataset_B, generate_dataset_C:

Эти функции вызывают norm_dataset или nonlinear_dataset_13 с конкретными параметрами для генерации данных.

generate_dataset_A и generate_dataset_B генерируют линейно разделимые данные.

generate_dataset_C генерирует нелинейно разделимые данные.

Результаты

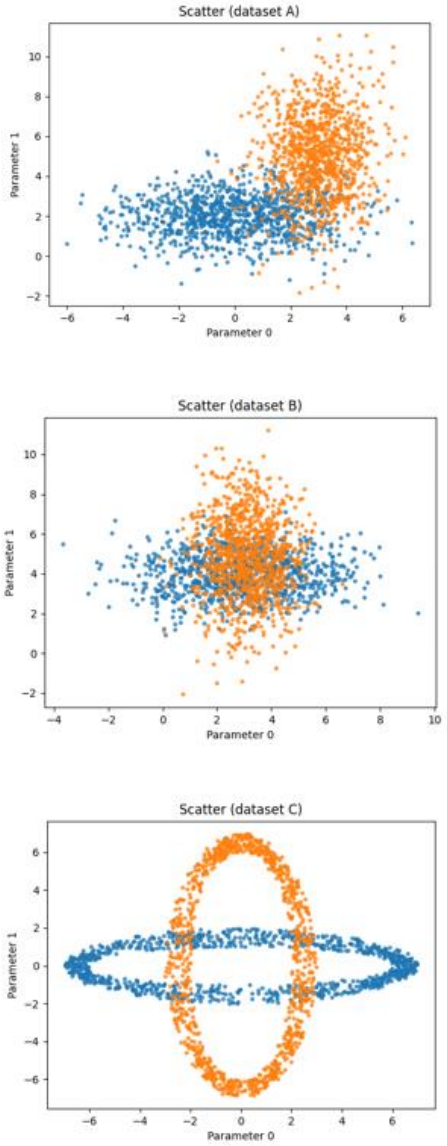

Рис. 1. Данные (наборы A, B, C)

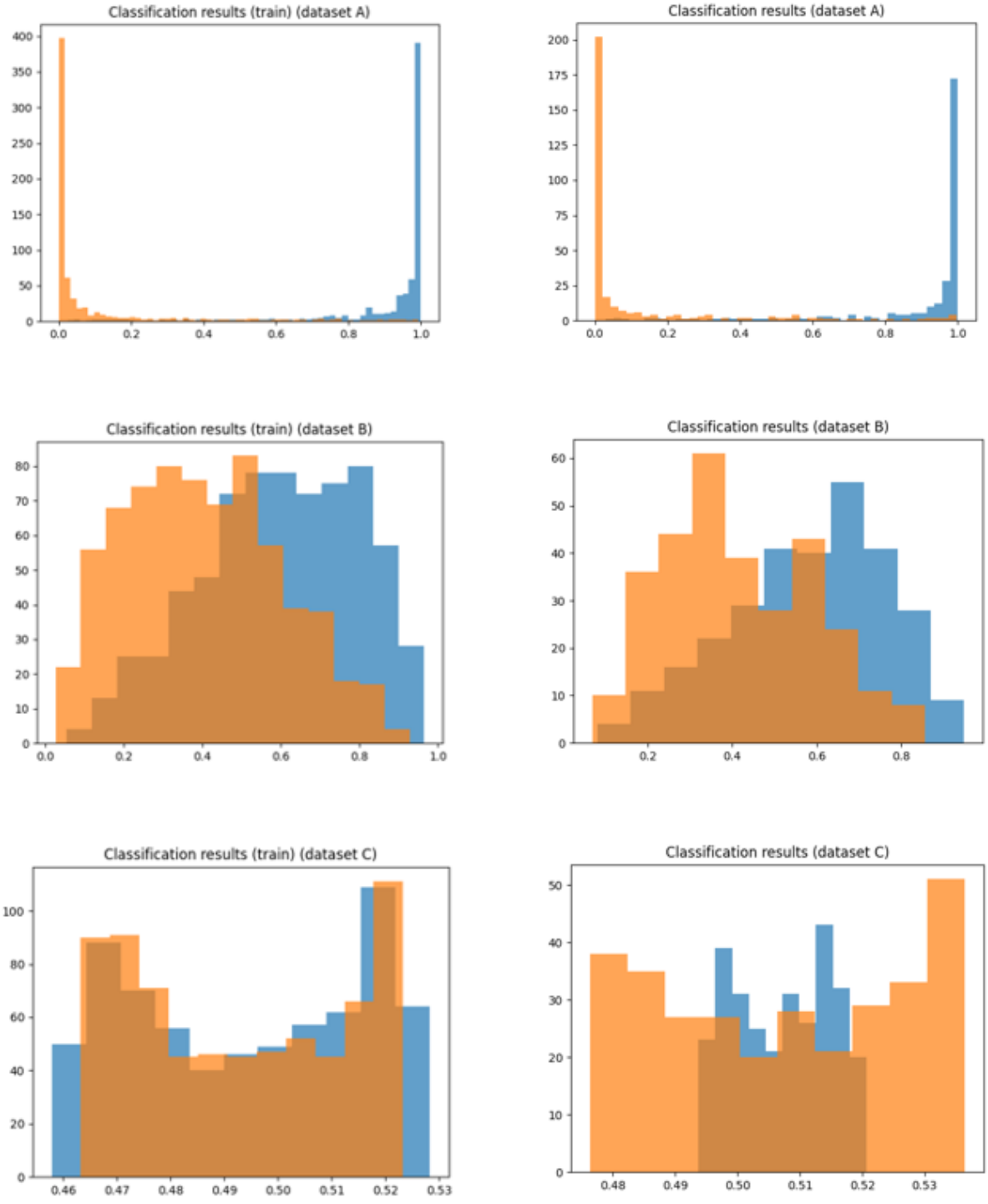

Рис. 2. Результаты классификации (наборы A, B, C). Слева – выборки, используемые при тренировке модели, справа – тестовые выборки.

	Число объектов	Точность, %	Чувствительность, %	Специфичность, %
Выборка A (Train)	700	0.935	0.930	0.940
Выборка A (Test)	300	0.91	0.91	0.91
Выборка B (Train)	700	0.6807	0.685	0.676
Выборка B (Test)	300	0.692	0.696	0.687
Выборка C (Train)	700	0.535	0.536	0.533
Выборка C (Test)	300	0.538	0.524	0.556

Выводы

В ходе работы были изучены принципы работы логистической регрессии, реализована модель классификатора и проведена её оценка на различных наборах данных. Полученные результаты подтвердили, что логистическая регрессия эффективна для линейно разделимых данных, но её применение ограничено в случае нелинейной разделимости классов.