МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САПР

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Автоматизация схемотехнического проектирования»

Тема: ДЕРЕВЬЯ И ЛЕСА РЕШЕНИЙ

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

Реализация классификатора на основе дерева принятия решений и исследование его свойств.

Основные теоретические положения

Дерево решений (распознающее дерево, recognition или decision tree) – распознаватели вида, при котором для распознаваемого объекта проводится конечная последовательность сравнений значений его признаков с константами на равенство или неравенство, причем от результатов каждого сравнения зависят дальнейшие действия – продолжать сравнивать с чем-то еще или давать ответ распознавания. То есть распознавание можно представить как двоичное дерево вложенных операторов вида:

if x[j] ?? d[k] then …

else…

завершающееся листьями:

return res[h]

где x-массив признаков, d-массив пороговых значений, res-массив возможных ответов, знаком ?? обозначена операция сравнения.

Деревья строятся при помощи обучения с учителем. В качестве обучающего набора данных используется множество наблюдений, для которых предварительно задана метка класса.

Структурно дерево решений состоит из объектов двух типов — узлов (node) и листьев (leaf). В узлах расположены решающие правила и подмножества наблюдений, которые им удовлетворяют. В листьях содержатся классифицированные деревом наблюдения: каждый лист ассоциируется с одним из классов, и объекту, который распределяется в лист, присваивается соответствующая метка класса.

Под обучением дерева понимается определение его структуры, операций сравнения, пороговых величин и сравниваемых на каждом узле признаков, а также ответов в каждом листе. По своей сути, дерево может быть описано как 54 набор операций «разрезания» признакового пространства гиперплоскостями, проходящими через пороговые значения признаков (рис. 3.1). Именно поэтому дерево решений является линейным классификатором, несмотря на его достаточно сложную организацию. Стоит отметить, что в общем случае деревья применимы и для решения задач регрессии.

Алгоритмы построения деревьев решений относят к категории жадных алгоритмов. Алгоритм считается жадным, если допускает, что локальнооптимальные решения на каждом шаге (разбиения в узлах), приводят к оптимальному итоговому решению. В случае деревьев решений это означает, что если один раз был выбран атрибут, и по нему было произведено разбиение на подмножества, то алгоритм не может вернуться назад и выбрать другой атрибут, который дал бы лучшее итоговое разбиение. Поэтому на этапе построения нельзя сказать обеспечит ли выбранный атрибут, в конечном итоге, оптимальное разбиение. Более подробно про алгоритмы обучения – в материалах курса, а также в дополнительных материалах.

Одно дерево не всегда может эффективно справиться с задачей классификации или регрессии. В таком случае возможным выходом является использование ансамблей. Ансамбль – это некоторая совокупность алгоритмов, объединенных в единое целое. Каждый алгоритм имеет свою вероятность ошибки, и объединяя выходы тысячи среднеточных моделей можно добиться более точного сведенного результата «голосования», усреднив результаты.

Бэггинг (от Bagging - Bootstrap aggregation) — это один из первых и самых простых видов ансамблей. Он был придуман Лéо Брéйманом в 1994 году. Бэггинг основан на статистическом методе бутстрэпа, который

позволяет оценивать многие статистики сложных распределений, когда выборка дробиться на множество подвыборок и на них оценивается бутстрэп статистика. Бэггинг позволяет снизить дисперсию обучаемого классификатора, уменьшая величину, на сколько ошибка будет отличаться, если обучать модель на разных наборах данных, или другими словами, предотвращает переобучение. Эффективность бэггинга достигается благодаря тому, что базовые алгоритмы, обученные по различным подвыборкам, получаются достаточно различными, и их ошибки взаимно компенсируются при голосовании, а также за счёт того, что объекты-выбросы могут не попадать в некоторые обучающие подвыборки.

Случайный лес (random forest) — бэггинг над решающими деревьями, при обучении которых для каждого разбиения признаки выбираются из некоторого случайного подмножества признаков. Для задачи классификации итоговое решение выбирается по большинству результатов, выданных классификаторами, а в задаче регрессии — по их среднему значению.

Одной из главных проблем случайного леса является его склонность к переобучению и «рваным краям» разделяющих поверхностей. Алгоритмы построения и обучения лесов постоянно совершенствуются и дополняются.

Используемые библиотеки: numpy, scikit-learn, scikit-plot, matplotlib

Ход работы

1. *Полный код программы:*

lab3.py:

```
import numpy as np
import scikitplot as skplt
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import roc_auc_score

from data_generator import generate_dataset_A, generate_dataset_B, generate_dataset_C
from test_classification import test_classification, build_scatters, plot_clf_results



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[0:trainCount]
Xtest = X[trainCount:N*2+1]
Ytrain = Y[0:trainCount]
Ytest = Y[trainCount:N*2+1]

# build_scatters(class0, class1)

# clf = DecisionTreeClassifier(random_state=0).fit(Xtrain, Ytrain)
# clf = DecisionTreeClassifier(random_state=0, max_depth=10).fit(Xtrain, Ytrain)
clf = RandomForestClassifier(random_state=0).fit(Xtrain, Ytrain)

Pred_test, Pred_test_proba, acc_test, sensitivity_test, specificity_test = test_classification(
    clf, Xtest, Ytest)

Pred_train, Pred_train_proba, acc_train, sensitivity_train, specificity_train = test_classification(
    clf, Xtrain, Ytrain)

# plot_clf_results(Pred_train_proba, Ytrain, "RFC Classification results (train) (dataset A)")
```

```
# plot_clf_results(Pred_test_proba, Ytest, "RFC Classification results (test) (dataset A)")

# skplt.metrics.plot_roc_curve(Ytest, Pred_test_proba, figsize = (8, 8))
# plt.savefig('RFC_roc_curve_test_dataset_A' + '.png')
# plt.show()

# Расчет площади под кривой
AUC = roc_auc_score(Ytest, Pred_test_proba[:, 1])

# Расчет максимальной площади под кривой
max_AUC = 0
max_AUC_depth = 0
for d in range(1, 301, 10):
    clf_auc = DecisionTreeClassifier(random_state=0, max_depth=d).fit(Xtrain, Ytrain)
    Pred_test, Pred_test_proba, acc_test, sensitivity, specificity = test_classification(
        clf_auc, Xtest, Ytest)
    a = roc_auc_score(Ytest, Pred_test_proba[:, 1])
    if a > max_AUC:
        max_AUC = a
        max_AUC_depth = d

print('Accuracy (train):', acc_train)
print('Sensitivity (train):', sensitivity_train)
print('Specificity (train):', specificity_train)

print('Accuracy (test):', acc_test)
print('Sensitivity (test):', sensitivity_test)
print('Specificity (test):', specificity_test)

print("AUC: " + str(AUC))
print("Max AUC: " + str(max_AUC) + " (depth: " + str(max_AUC_depth) + ")")
```

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)
```

test_classification.py:

```
import matplotlib.pyplot as plt


def test_classification(clf, X, Y):
    Pred = clf.predict(X)
    Pred_proba = clf.predict_proba(X)

    acc = clf.score(X, Y)

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

        if Pred[i]==False and Y[i]==False:
            specificity += 1

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

    return Pred, Pred_proba, acc, sensitivity, specificity
```

```
def build_scatters(class0, class1):
    col = len(class0[0])
    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()


def plot_clf_results(pred_proba, Y, title=""):
    plt.hist(pred_proba[Y, 1], bins=8, alpha=0.7)
    plt.hist(pred_proba[~Y, 1], bins=8, alpha=0.7)
    plt.title(title)
    plt.savefig(title.replace(' ', '_') + '.png')
    plt.show()
```

2. *Пояснения к коду:*

   Файл lab3.py

   Импорт библиотек:

```
import numpy as np
import scikitplot as skplt
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score

from data_generator import generate_dataset_A, generate_dataset_B, generate_dataset_C
from test_classification import test_classification, build_scatters, plot_clf_results
```

Генерация данных:

```
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
```

Разделение данных на обучающую и тестовую выборки

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

Обучение классификатора:

```
# clf = DecisionTreeClassifier(random_state=0).fit(Xtrain, Ytrain)
# clf = DecisionTreeClassifier(random_state=0, max_depth=10).fit(Xtrain, Ytrain)
clf = RandomForestClassifier(random_state=0).fit(Xtrain, Ytrain)
```

Тестирование модели:

```
Pred_test, Pred_test_proba, acc_test, sensitivity_test, specificity_test = test_classification(
   clf, Xtest, Ytest)

Pred_train, Pred_train_proba, acc_train, sensitivity_train, specificity_train = test_classification(
   clf, Xtrain, Ytrain)
```

Визуализация результатов:

```
plot_clf_results(Pred_train_proba, Ytrain, "RFC Classification results (train) (dataset A)")
plot_clf_results(Pred_test_proba, Ytest, "RFC Classification results (test) (dataset A)")

skplt.metrics.plot_roc_curve(Ytest, Pred_test_proba, figsize = (8, 8))
plt.savefig('RFC_roc_curve_test_dataset_A' + '.png')
plt.show()
```

Расчет AUC:

```
# Расчет площади под кривой
AUC = roc_auc_score(Ytest, Pred_test_proba[:, 1])
```

Расчет максимальной площади под кривой:

```
max_AUC = 0
max_AUC_n_est = 0
for d in range(1, 301, 10):
    clf_auc = RandomForestClassifier(random_state=0, n_estimators=d).fit(Xtrain, Ytrain)
    Pred_test, Pred_test_proba, acc_test, sensitivity, specificity = test_classification(
        clf_auc, Xtest, Ytest)
    a = roc_auc_score(Ytest, Pred_test_proba[:, 1])
    if a > max_AUC:
        max_AUC = a
        max_AUC_n_est = d
```

Вывод метрик:

```
print('Accuracy (train):', acc_train)
print('Sensitivity (train):', sensitivity_train)
print('Specificity (train):', specificity_train)

print('Accuracy (test):', acc_test)
print('Sensitivity (test):', sensitivity_test)
print('Specificity (test):', specificity_test)

print("AUC: " + str(AUC))
print("Max AUC: " + str(max_AUC) + " (number of trees: " + str(max_AUC_n_est) + ")")
```

Файл data_generator.py:

Функция norm_dataset:

Генерирует данные для двух классов на основе нормального распределения.

```
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
```

Функция nonlinear_dataset_13:

Генерирует нелинейные данные для двух классов на основе полярных координат.

```
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
```

Функции generate_dataset_A, generate_dataset_B, generate_dataset_C:

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.

```
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)
```

Файл test_classification.py:

Функции test_classification, build_scatters, plot_clf_results используются для вывода результатов.

```
def test_classification(clf, X, Y):
    Pred = clf.predict(X)
```

```
    Pred_proba = clf.predict_proba(X)

    acc = clf.score(X, Y)

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

        if Pred[i]==False and Y[i]==False:
            specificity += 1

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

    return Pred, Pred_proba, acc, sensitivity, specificity


def build_scatters(class0, class1):
    col = len(class0[0])
    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()


def plot_clf_results(pred_proba, Y, title=""):
    plt.hist(pred_proba[Y, 1], bins=8, alpha=0.7)
    plt.hist(pred_proba[~Y, 1], bins=8, alpha=0.7)
    plt.title(title)
    plt.savefig(title.replace(' ', '_') + '.png')
    plt.show()
```

Полученные графики

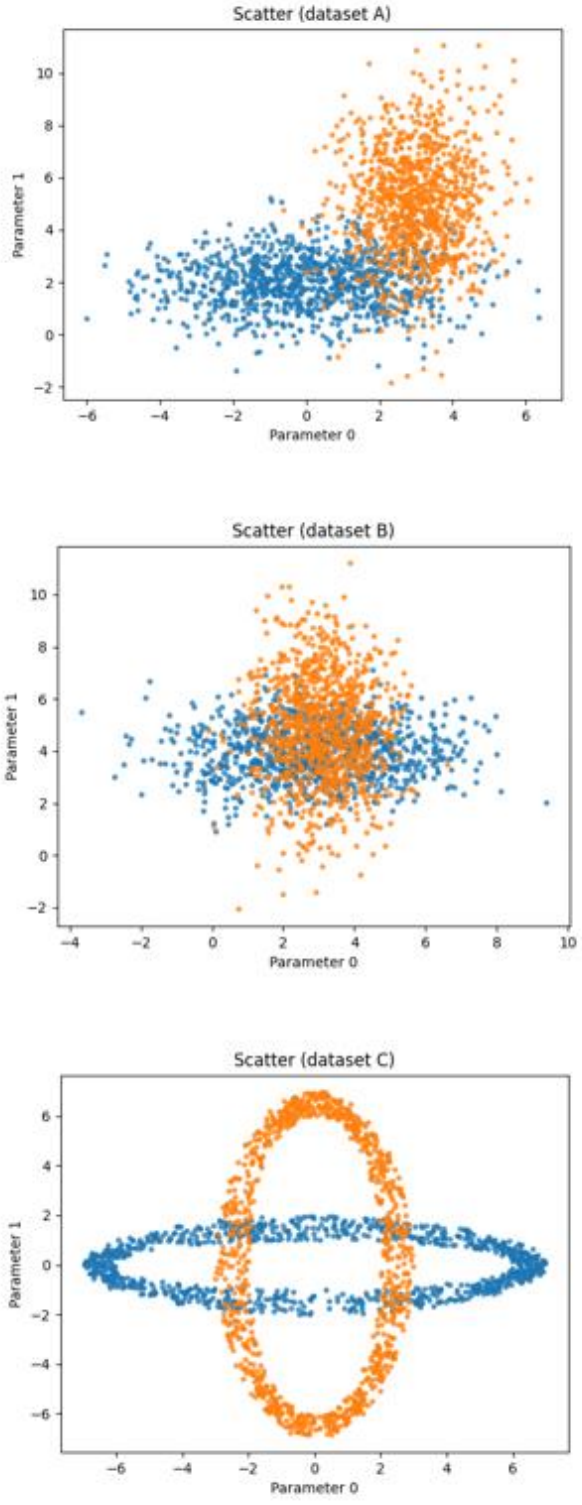

Рис. 1. Данные (наборы A, B, C)

DecisionTreeClassifier:

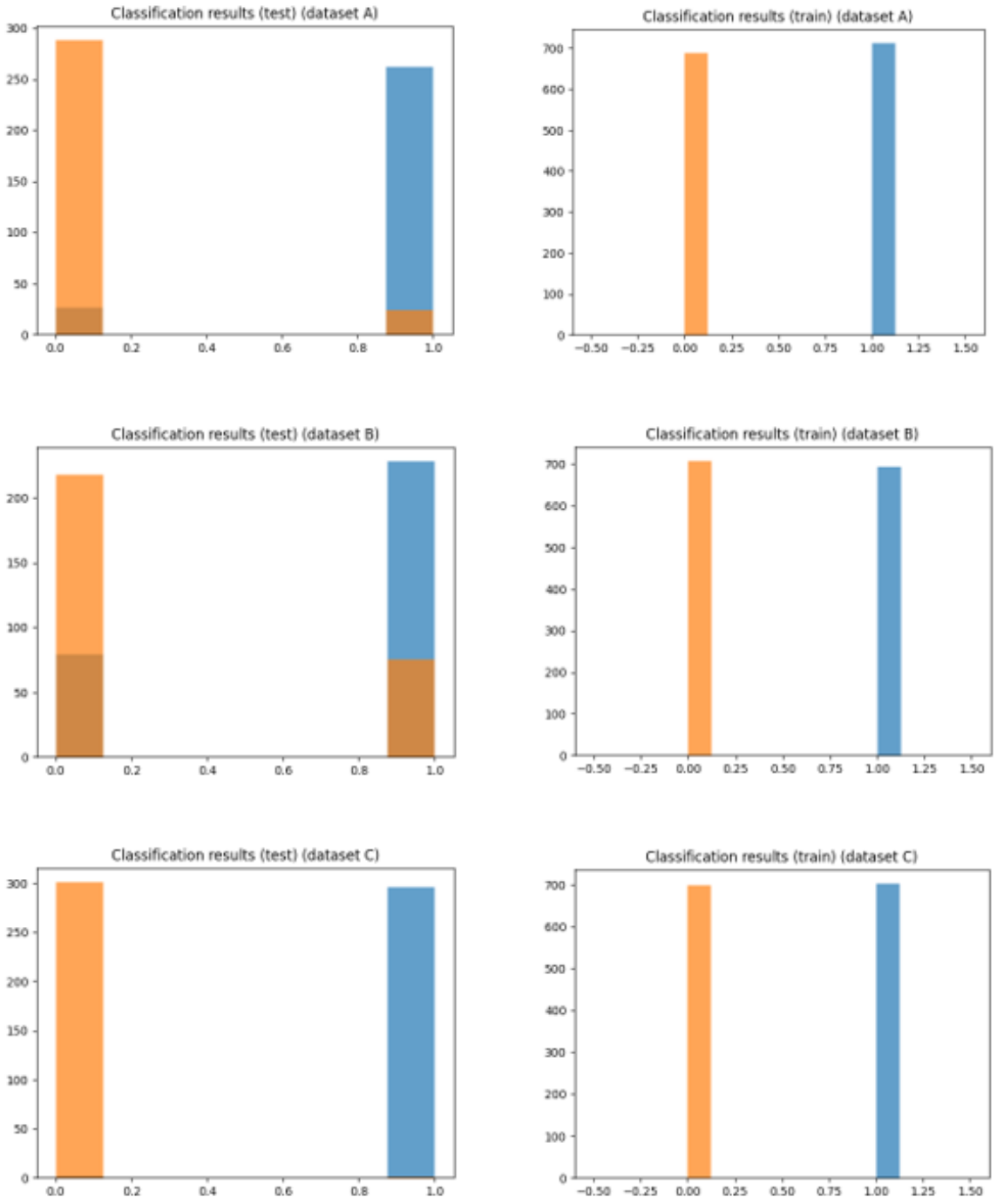

Рис. 2. Результаты классификации (наборы A, B, C). Справа – выборки, используемые при тренировке модели, слева – тестовые выборки.

	Число объектов	Точность, %	Чувствительность, %	Специфичность, %
Выборка A (Train)	700	1.0	1.0	1.0
Выборка A (Test)	300	0.902	0.895	0.907
Выборка B (Train)	700	1.0	1.0	1.0
Выборка B (Test)	300	0.743	0.75	0.73
Выборка C (Train)	700	1.0	1.0	1.0
Выборка C (Test)	300	0.995	0.997	0.993

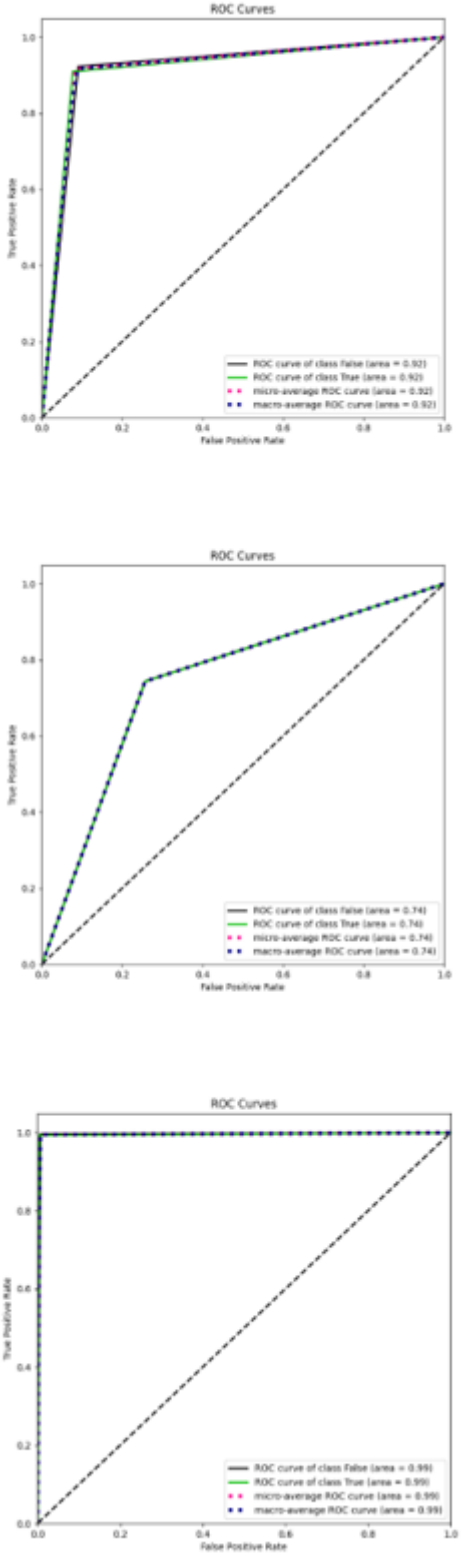

Рис. 3. ROC-кривые для наборов A, B, C

Площади ROC-кривых:

A	0.89
B	0.738
C	0.998

RandomForestClassifier:

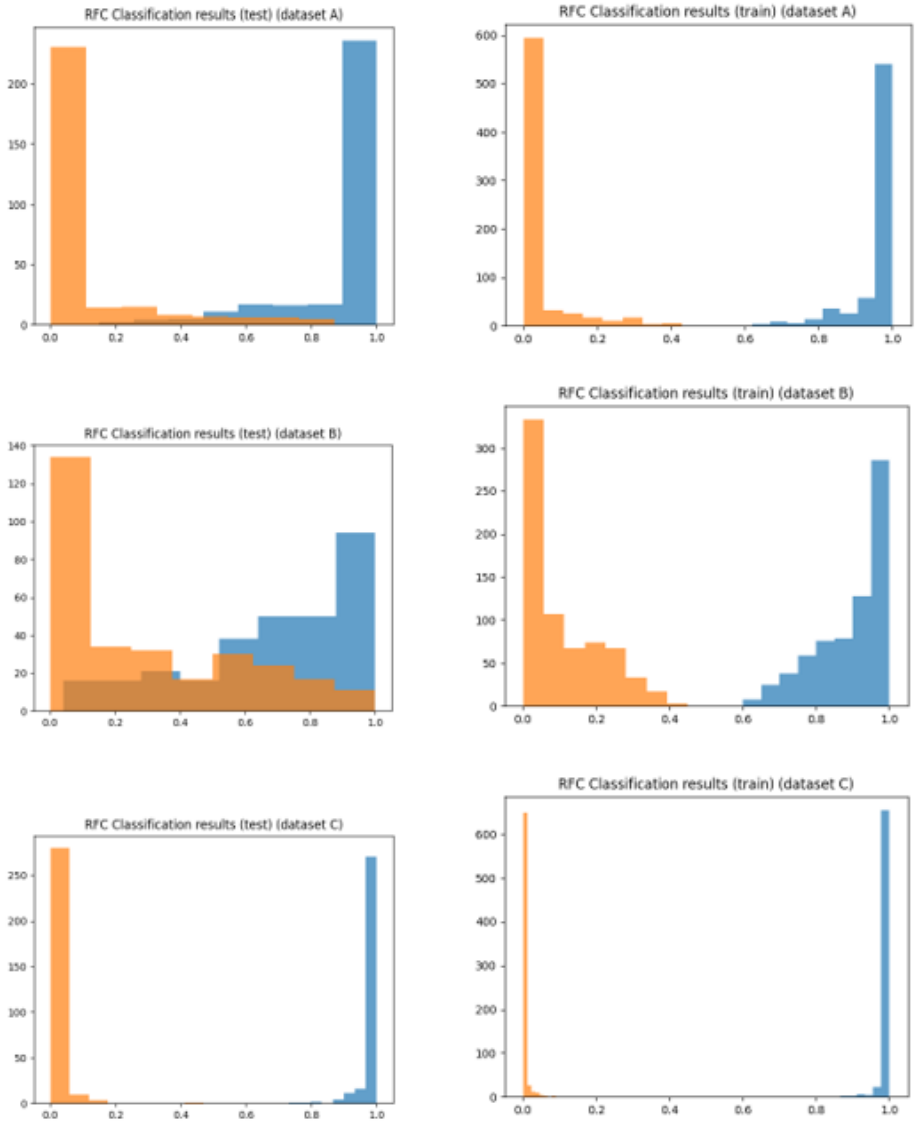

Рис. 4. Результаты классификации (наборы A, B, C). Справа – выборки, используемые при тренировке модели, слева – тестовые выборки.

	Число объектов	Точность, %	Чувствительность, %	Специфичность, %
Выборка A (Train)	700	1.0	1.0	1.0
Выборка A (Test)	300	0.925	0.933	0.947
Выборка B (Train)	700	1.0	1.0	1.0
Выборка B (Test)	300	0.715	0.741	0.759
Выборка C (Train)	700	1.0	1.0	1.0
Выборка C (Test)	300	0.996	1.0	1.0

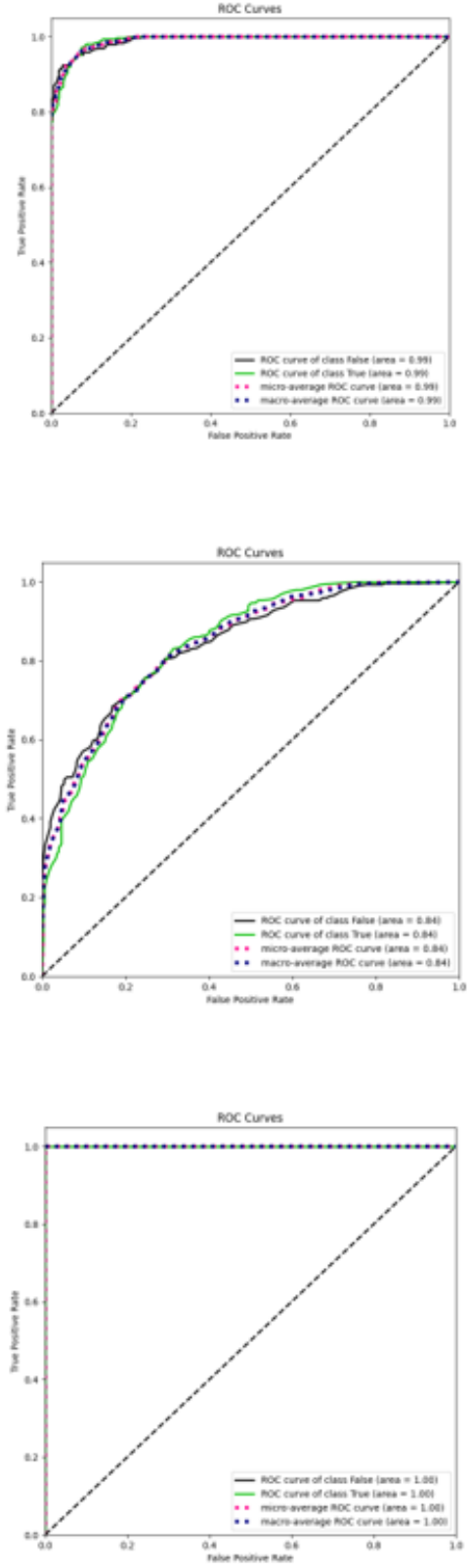

Рис. 5. ROC-кривые для наборов A, B, C

Площади ROC-кривых:

A	0.989
B	0.876
C	1.0

Результаты подбора наилучших гиперпараметров моделей

Для классификатора DecisionTreeClassifier была подобрана максимальная глубина дерева, снижающая переобучение модели.

Без подбора:

	Число объектов	Точность, %	Чувствительность, %	Специфичность, %
Выборка B (Train)	700	1.0	1.0	1.0
Выборка B (Test)	300	0.743	0.75	0.73

При глубине дерева 9:

	Число объектов	Точность, %	Чувствительность, %	Специфичность, %
Выборка B (Train)	700	0.847	0.797	0.915
Выборка B (Test)	300	0.788	0.766	0.817

С помощью цикла for было подобрано такое значение количества деревьев в лесе, которое дает наибольшее значение площади под кривой на тестовой выборке:

Выборка (test)	Наибольшая площадь	Количество деревьев
A	0.980	171
B	0.863	151
C	1.0	121

Выводы

В ходе выполнения лабораторной работы были реализованы и исследованы классификаторы на основе дерева решений и случайного леса. Результаты показали, что дерево решений склонно к переобучению, что выражается в высокой точности на обучающей выборке и более низкой на тестовой. Случайный лес демонстрирует более устойчивые результаты за счет усреднения множества деревьев, что снижает дисперсию и улучшает обобщающую способность модели. Оптимизация гиперпараметров, таких как глубина дерева и количество деревьев в лесе, позволила улучшить качество классификации.