**Министерство науки и высшего образования Российской Федерации**
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***
***(КФ МГТУ им. Н.Э. Баумана)***

**ФАКУЛЬТЕТ ИУК «Информатика и управление»**

**КАФЕДРА ИУК4 «Программное обеспечение ЭВМ, информационные технологии»**

# ЛАБОРАТОРНАЯ РАБОТА №3

«Машинное обучение с использованием OpenCV»

**ДИСЦИПЛИНА:** «Программные системы распознавания и обработки информации»

Выполнил: студент гр. ИУК4-31М _____ ( Сафронов Н.С, )
(подпись) (Ф.И.О.)

Проверил: _____ ( Гагарин Ю.Е. )
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2025

**Цель:**

Изучить некоторые алгоритмы обучения с учителем и без с использованием соответствующих функций библиотеки компьютерного зрения OpenCV

**Задачи:**

1. Изучить основные идеи, лежащие в основе следующих алгоритмов классификации:
   - машина опорных векторов;
   - дерево решений;
   - случайный лес;
   - градиентный бустинг деревьев решений.
2. Изучить идеи метода центров тяжести (k-means) для кластеризации.
3. Рассмотреть прототипы функций и интерфейсы классов, реализующих перечисленные алгоритмы в библиотеке OpenCV.
4. Рассмотреть простые примеры использования указанного набора функций.
5. Применить полученные навыки к решению модельных задач и проанализировать полученные результаты.

**Задание**

1. Реализуйте возможность сохранения и загрузки обученной модели в приложении для решения задач классификации.
2. Реализуйте функцию вычисления матрицы ошибок классификации $E$, где элемент $E_{i,j}$ равен количеству прецедентов выборки принадлежащих к классу $j$ и отнесенных алгоритмом классификации к классу $i$.
3. Реализуйте метод перекрестного контроля для подбора параметров алгоритмов обучения.
4. На рисунке 6 показано распределение для классов «false» и «true», а также показаны несколько потенциальных мест (a, b, c, d, e, f, g), где можно установить порог.

а) Нарисуйте точки a-g на ROC-кривой.

б) Как бы дерево решений разделило эти данные?

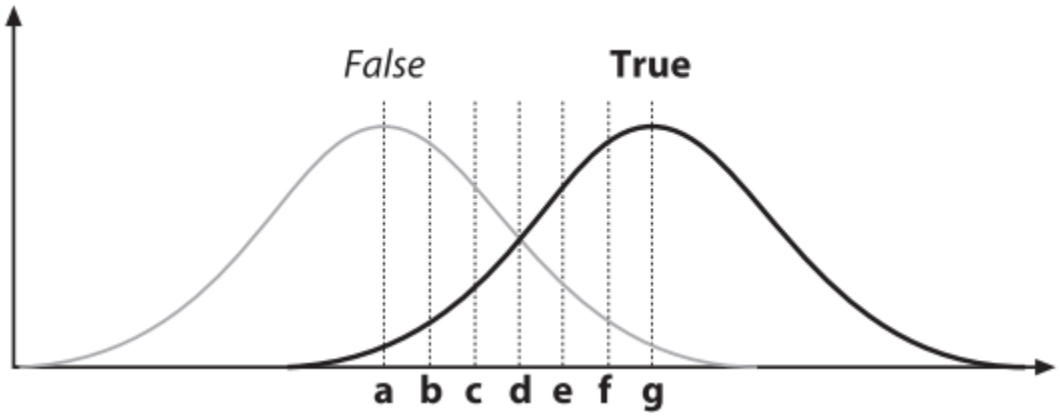


**Рис. 6.** Гауссовское распределение двух классов «false» и «true»

5. Измените исходный код для генерации данных - около верхней части внешнего для цикла for{} в секции K-means - для создания случайно сгенерированного помеченного набора данных. Используется одно нормальное распределение в 10 000 точек с центром в пикселях (63, 63) в изображении 128 на 128 со стандартным отклонением (img->width/6, img->height/6). Чтобы пометить эти данные, пространство разделяется на четыре квадранта с центром в пикселе (63, 63). Для получения вероятностей маркировки используется следующая схема. Если $x < 64$ используется 20%-ную вероятность для класса A; Если $x \geq 64$ используется коэффициент 90% для класса A. Если $y < 64$ используется 40%-ную вероятность для класса B; Если $y \geq 64$ используется коэффициент 60% для класса B. Умножение вероятностей $x$ и $y$ дает общую вероятность для класса A по квадранту со значениями, указанными в приведенной матрице 2 на 2. Если точка не помечена как A, то по умолчанию она помечается B. Например, если $x < 64$и $y < 64$ у нас будет 8%-ная вероятность того, что точка будет помечена классом A и 92%-ная - что эта точка будет помечена классом B. Матрица вероятности того, что точка, помеченная классом A (а если нет, это класс B), равна:

| | |
|---|---|
| $0.2 \times 0.6 = 0.12$ | $0.9 \times 0.6 = 0.54$ |
| $0.2 \times 0.4 = 0.08$ | $0.9 \times 0.4 = 0.36$ |

Используйте эти коэффициенты для обозначения точек данных. Для каждой точки данных определите его квадрант. Затем создайте случайное число от 0 до 1. Если значение меньше или равно вероятности квадранта, отметьте, что данные относятся к классу A; иначе - классом B. Получится список помеченных точек, x и y выступают в качестве функций.

6. Используя данные задания 5, примените AdaBoost для изучения двух моделей: с набором weak_count в 20 деревьев и другой с 500 деревьями. Произвольно выберите обучающий и тестовый наборы данных из 10 000 точек. Обучите алгоритм и сообщите результаты теста, когда обучающий набор содержит:

а) 150 точек;

б) 500 точек;

в) 1200 точек;

г) 5000 точек;

е) Проанализируйте результаты и сделайте выводы.

7. Повторите упражнение 6, но используйте классификатор случайного леса с 50 и 500 деревьями.

8. Повторите упражнение 6, но на этот раз используйте 60 деревьев и сравните случайный лес с против SVM (машина опорных векторов).

**Результаты выполнения работы**


**Рисунок 1** – Распределение двух классов

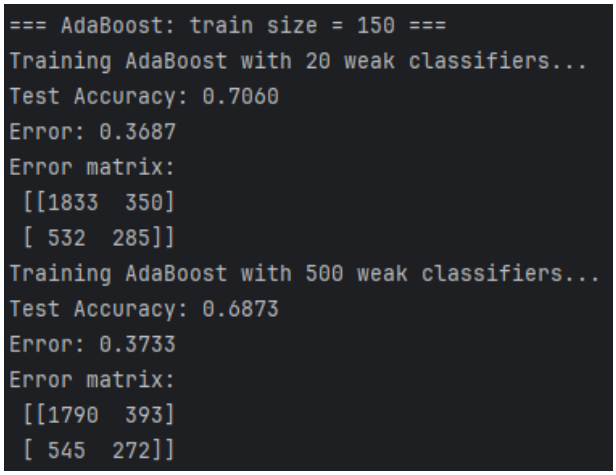


**Рисунок 2 –** Результаты для AdaBoost с размером выборки теста 150

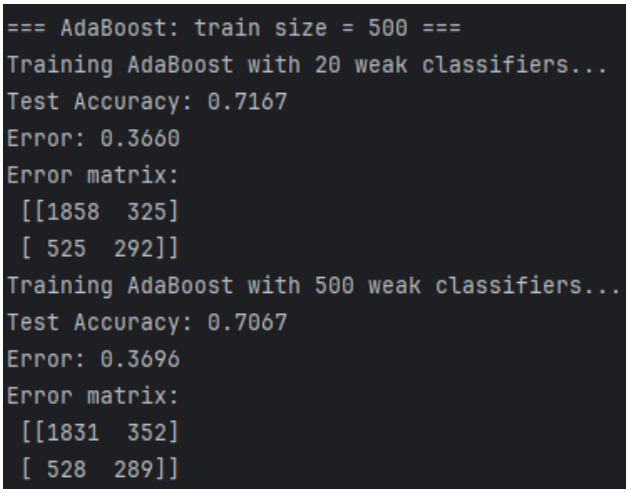


**Рисунок 3 –** Результаты для AdaBoost с размером выборки теста 500

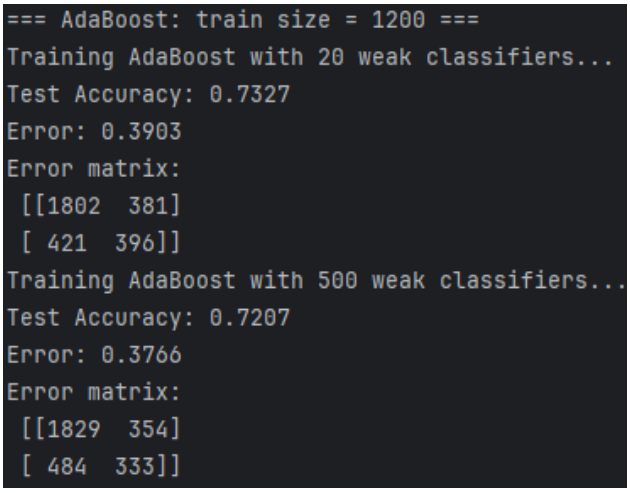


**Рисунок 4 –** Результаты для AdaBoost с размером выборки теста 1200

```
=== AdaBoost: train size = 5000 ===
Training AdaBoost with 20 weak classifiers...
Test Accuracy: 0.7380
Error: 0.3632
Error matrix:
 [[1899  284]
 [ 502  315]]
Training AdaBoost with 500 weak classifiers...
Test Accuracy: 0.7417
Error: 0.3549
Error matrix:
 [[1932  251]
 [ 524  293]]
```

**Рисунок 5 –** Результаты для AdaBoost с размером выборки теста 5000

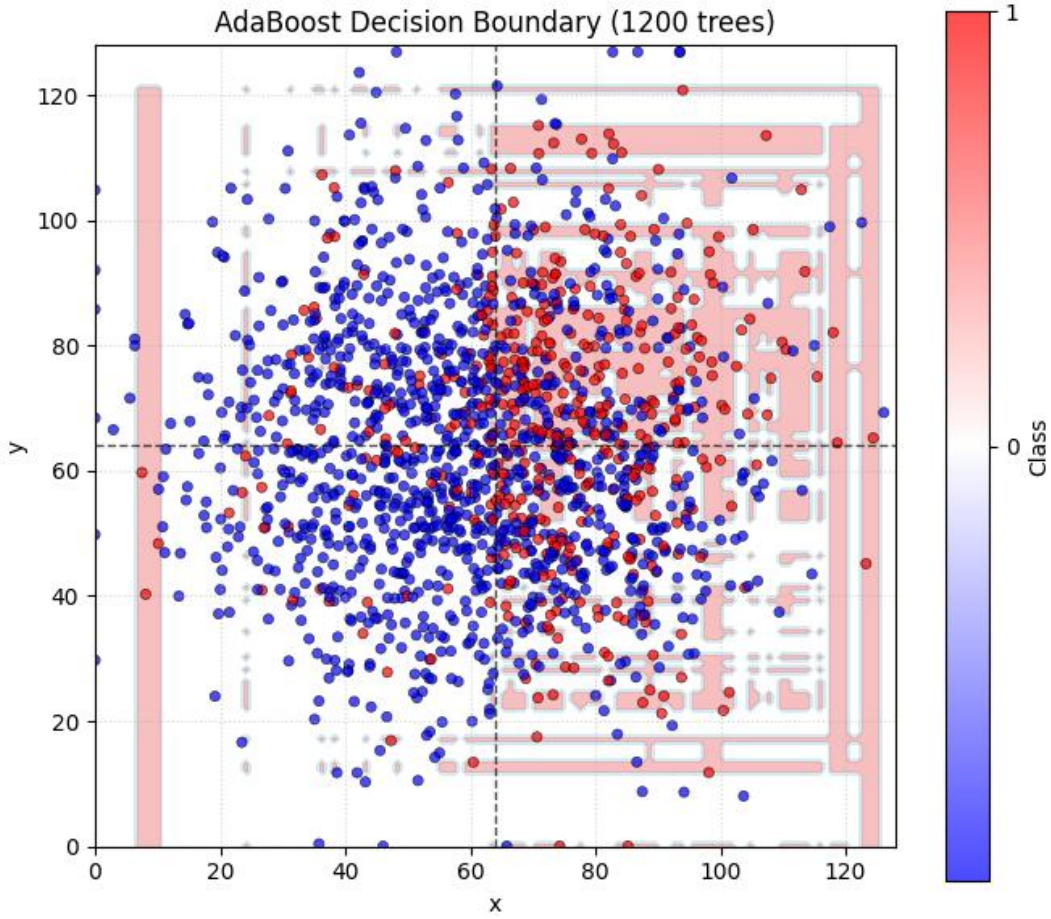


**Рисунок 6 –** Предсказанные границы классов и реальные точки классов на плоскости

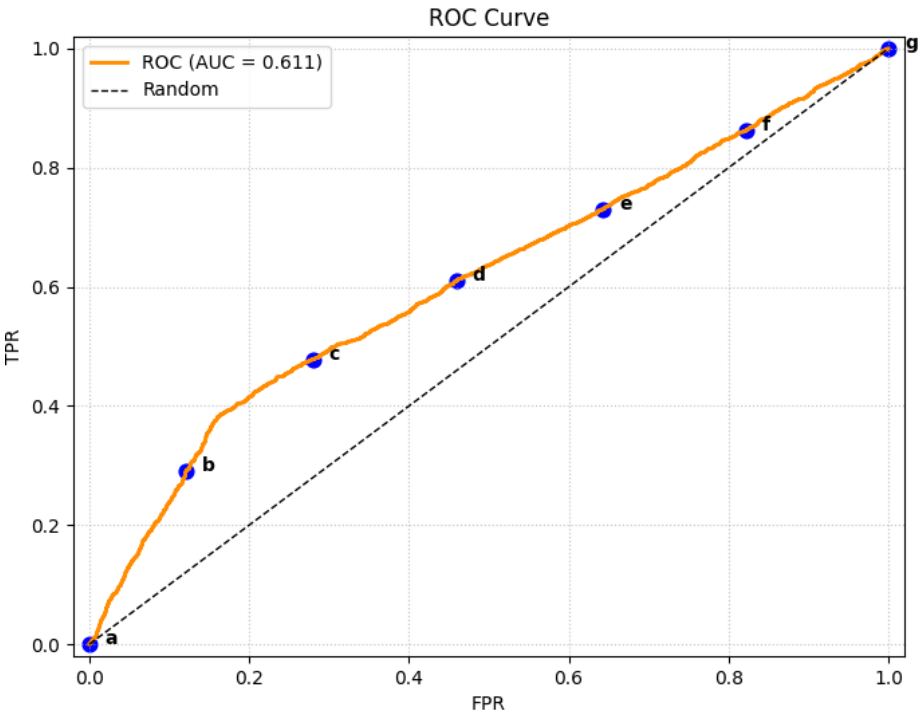


**Рисунок 7** – Кривая ROC для AdaBoost

```
=== Random Trees: train size = 150 ===
Training Random Trees with ~50 trees...
Test Accuracy: 0.7243
Error: 0.3598
Error matrix:
 [[1890  293]
 [ 534  283]]
Training Random Trees with ~500 trees...
Test Accuracy: 0.7193
Error: 0.3636
Error matrix:
 [[1870  313]
 [ 529  288]]

=== Random Trees: train size = 500 ===
Training Random Trees with ~50 trees...
Test Accuracy: 0.7283
Error: 0.3464
Error matrix:
 [[1940  243]
 [ 572  245]]
Training Random Trees with ~500 trees...
Test Accuracy: 0.7263
Error: 0.3582
Error matrix:
 [[1898  285]
 [ 536  281]]
```

**Рисунок 8** – Результаты для RandomTrees с размером выборки теста 150 и 500

```
=== Random Trees: train size = 1200 ===
Training Random Trees with ~50 trees...
Test Accuracy: 0.7350
Error: 0.3607
Error matrix:
 [[1903  280]
 [ 515  302]]
Training Random Trees with ~500 trees...
Test Accuracy: 0.7387
Error: 0.3614
Error matrix:
 [[1906  277]
 [ 507  310]]

=== Random Trees: train size = 5000 ===
Training Random Trees with ~50 trees...
Test Accuracy: 0.7467
Error: 0.3724
Error matrix:
 [[1882  301]
 [ 459  358]]
Training Random Trees with ~500 trees...
Test Accuracy: 0.7440
Error: 0.3781
Error matrix:
 [[1859  324]
 [ 444  373]]
```

**Рисунок 9 –** Результаты для RandomTrees с размером выборки теста 1200 и 5000

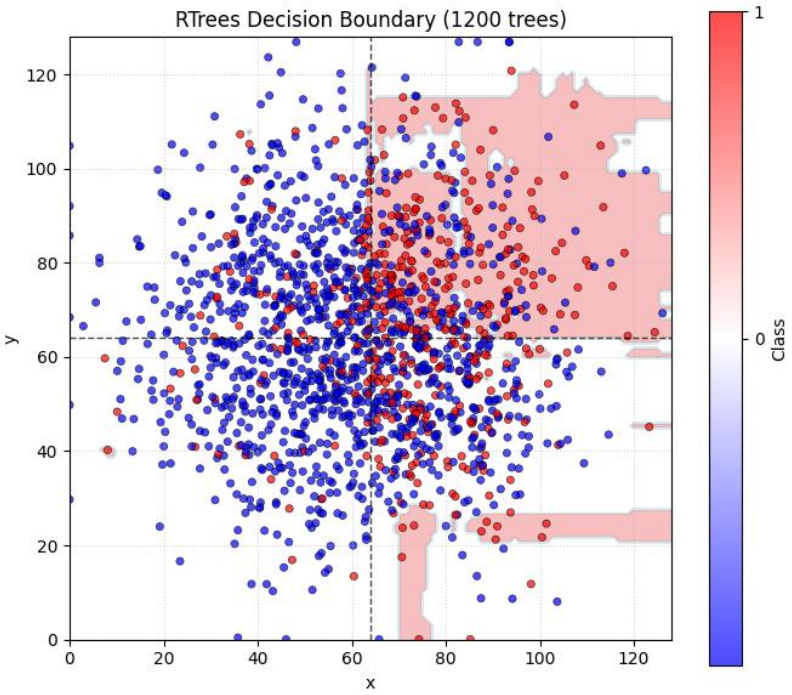


**Рисунок 10 –** Предсказанные границы классов и реальные точки классов на плоскости

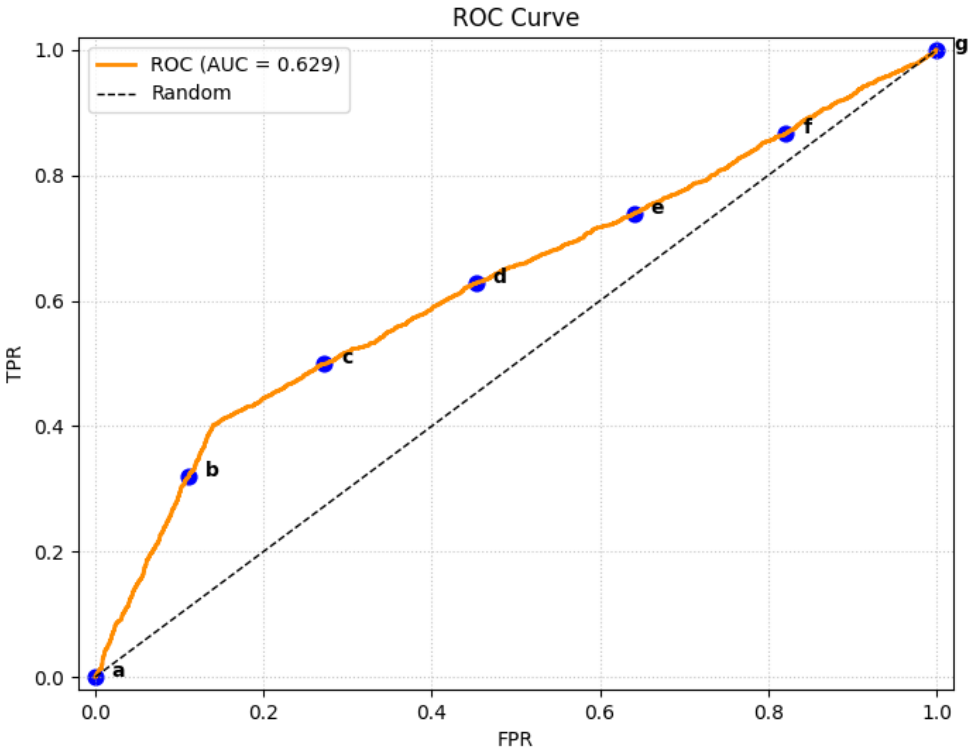


**Рисунок 11 –** Кривая ROC для RandomTrees

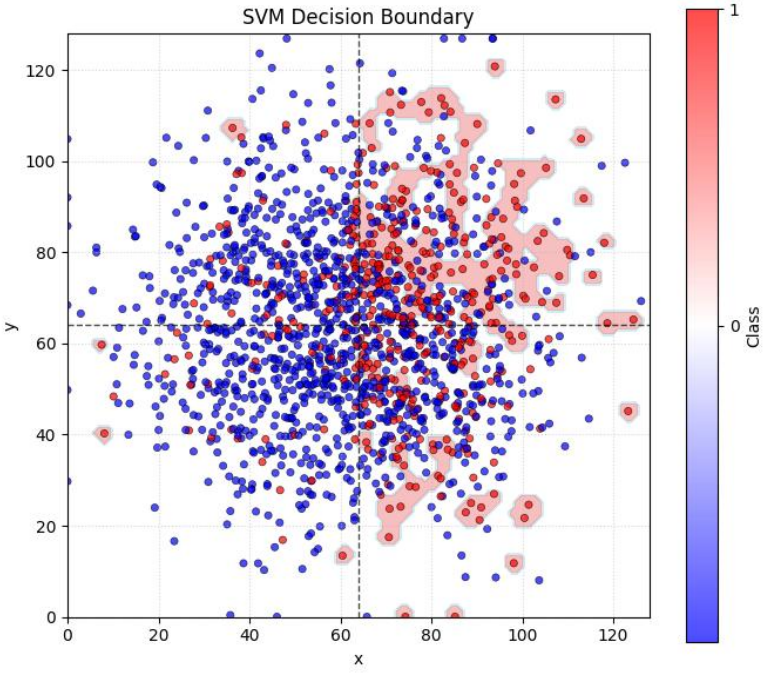


**Рисунок 12 –** Предсказанные границы классов и реальные точки классов на плоскости SVM

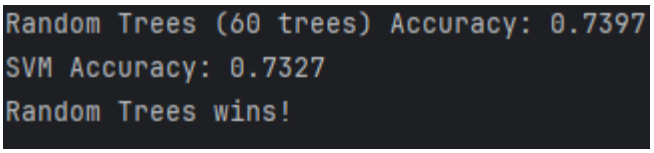


**Рисунок 13 –** Результат сравнения случайного леса против SVM

**Вывод:** в процессе выполнения лабораторной работы были изучены некоторые алгоритмы обучения с учителем и без с использованием соответствующих функций библиотеки компьютерного зрения OpenCV.

## Листинг программы

```
import numpy as np
import cv2
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import AdaBoostClassifier,
RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
import pickle
import os
import typing as t

Features = np.ndarray
Labels = np.ndarray

# == Задание 5 ==
def generate_labeled_data(
        n_points: int = 10000,
        img_size: int = 128,
        center: float = 63.0,
        seed: t.Optional[int] = None,
) -> tuple[Features, Labels]:
    """Генерация размеченных данных."""
    if seed is not None:
        np.random.seed(seed)

    x = np.random.normal(center, img_size / 6.0, n_points)
    y = np.random.normal(center, img_size / 6.0, n_points)
    x = np.clip(x, 0, img_size - 1)
    y = np.clip(y, 0, img_size - 1)
    samples = np.column_stack((x, y))
    samples = samples.astype(np.float32)

    quadrant_probs = [0.54, 0.12, 0.08, 0.36]

    labels = np.empty(n_points, dtype=np.int32)
    for i, (xi, yi) in enumerate(samples):
        if xi >= center and yi >= center:
            q = 0
        elif xi < center and yi >= center:
            q = 1
        elif xi < center and yi < center:
            q = 2
        else:
            q = 3
        labels[i] = -1 if np.random.rand() > quadrant_probs[q] else 1

    return samples, labels
```

```
def plot_decision_boundary(
        model: cv2.ml.StatModel,
        X: np.ndarray,
        y: np.ndarray,
        title: str = "Decision Boundary",
        resolution: int = 128
):
    """Строит границу принятия решений модели в 2D-пространстве."""
    # Границы графика
    x_min, x_max = 0, 128
    y_min, y_max = 0, 128

    # Создаём сетку
    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, resolution),
        np.linspace(y_min, y_max, resolution)
    )
    grid_points             =             np.column_stack([xx.ravel(),
yy.ravel()]).astype(np.float32)

    # Предсказания на сетке
    _, predictions = model.predict(grid_points)
    predictions = predictions.reshape(xx.shape)

    # Визуализация
    plt.figure(figsize=(8, 7))

    # Фон — предсказания модели
    plt.contourf(xx,   yy,   predictions,   levels=[-0.5,   0.5,   1.5],
colors=['lightblue', 'lightcoral'], alpha=0.5)

    # Точки данных
    scatter = plt.scatter(X[:, 0], X[:, 1], c=y.flatten(), cmap='bwr',
edgecolors='k', s=20, linewidth=0.5, alpha=0.7)
    plt.colorbar(scatter, ticks=[0, 1], label='Class')

    # Линии x=64 и y=64 (границы квадрантов)
    plt.axvline(x=64,    color='k',    linestyle='--',    linewidth=1,
alpha=0.7)
    plt.axhline(y=64,    color='k',    linestyle='--',    linewidth=1,
alpha=0.7)

    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(title)
    plt.gca().set_aspect('equal')
    plt.grid(True, linestyle=':', alpha=0.5)
    plt.show()
```

```
# == Задание 1 ==
def save_model(model: cv2.ml.StatModel, filename: str):
    """Сохранить модель."""
    model.save(filename)

def load_model(filename: str, model_type: t.Any) -> cv2.ml.StatModel |
None:
    """Загрузить модель."""
    if not os.path.exists(filename):
        return None
    return model_type.load(filename)

# == Задание 2 ==
def  compute_confusion_matrix(y_test:  Labels,  y_pred:  Labels)  ->
np.ndarray:
    """Вычислить матрицу ошибок классификации."""
    error = np.mean(y_pred != y_test)
    cm = confusion_matrix(y_test, y_pred)
    print(f"Error: {error:.4f}")
    print("Error matrix:\n", cm)
    return error

# == Задание 3 ==
def cross_validate_model(
    model_factory: t.Callable[[], cv2.ml.StatModel],
    X: Features,
    y: Labels,
    k: int = 5
) -> np.ndarray:
    """Кросс-валидация модели."""
    indices = np.arange(len(X))
    np.random.shuffle(indices)
    fold_size = len(X) // k
    scores = []

    for i in range(k):
        start = i * fold_size
        end = start + fold_size if i < k - 1 else len(X)
        val_idx = indices[start:end]
        train_idx = np.concatenate([indices[:start], indices[end:]])

        X_train, y_train = X[train_idx], y[train_idx]
        X_val, y_val = X[val_idx], y[val_idx]

        model = model_factory()
        model.train(cv2.ml.TrainData.create(X_train,
cv2.ml.ROW_SAMPLE, y_train))
        _, y_pred = model.predict(X_val)
        y_pred = y_pred.astype(int)
        acc = np.mean(y_pred.flatten() == y_val.flatten())
        scores.append(acc)
```

```
    scores = np.array(scores)
    print(f"Accuracy: {scores.mean():.4f} ± {scores.std() * 2:.4f}")
    return scores

# == Задание 4 ==
def plot_class_distributions(
    x: np.ndarray,
    false_pdf: np.ndarray,
    true_pdf: np.ndarray,
    thresholds,
    labels,
    save_path: t.Optional[str] = None
) -> None:
    plt.figure(figsize=(8, 5))
    plt.fill_between(x,     false_pdf,     color='red',     alpha=0.7,
label='False')
    plt.fill_between(x,     true_pdf,     color='green',     alpha=0.7,
label='True')
    for th, label in zip(thresholds, labels):
        plt.axvline(x=th,         color='gray',         linestyle='--',
label=f'{label}')
    plt.title('Гауссовское распределение двух классов')
    plt.xlabel('Значение')
    plt.ylabel('Плотность вероятности')
    plt.legend()
    if save_path:
        plt.savefig(save_path, dpi=150)
    plt.show()


def plot_roc(
    y_true: np.ndarray,
    y_scores: np.ndarray,
    point_labels: t.Optional[t.List[str]] = None,
    title: str = "ROC Curve"
) -> None:
    y_true = np.asarray(y_true).flatten()
    y_scores = np.asarray(y_scores).flatten()

    if set(np.unique(y_true)) == {-1, 1}:
        y_true = (y_true == 1).astype(int)
    elif set(np.unique(y_true)) == {0, 1}:
        pass
    else:
        raise ValueError("Метки должны быть 0/1 или -1/1")

    from sklearn.metrics import roc_auc_score
    try:
        auc_original = roc_auc_score(y_true, y_scores)
    except:
        auc_original = 0.0
```

```
    if auc_original < 0.5:
        y_scores = -y_scores

    # Сортировка по убыванию оценок
    sorted_idx = np.argsort(y_scores)[::-1]
    y_true_sorted = y_true[sorted_idx]

    tp = np.cumsum(y_true_sorted)
    fp = np.cumsum(1 - y_true_sorted)

    total_pos = tp[-1] if len(tp) > 0 else np.sum(y_true)
    total_neg = fp[-1] if len(fp) > 0 else len(y_true) - total_pos

    tpr = tp / total_pos if total_pos > 0 else np.zeros_like(tp)
    fpr = fp / total_neg if total_neg > 0 else np.zeros_like(fp)

    # Добавляем (0,0) и (1,1)
    fpr = np.concatenate([[0.0], fpr, [1.0]])
    tpr = np.concatenate([[0.0], tpr, [1.0]])

    auc_val = np.trapz(tpr, fpr)

    # Визуализация
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC (AUC =
{auc_val:.3f})')
    plt.plot([0, 1], [0, 1], 'k--', lw=1, label='Random')

    # Точки a–g
    if point_labels is None:
        point_labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
    n_pts = len(point_labels)
    indices = np.linspace(0, len(fpr) - 1, min(n_pts, len(fpr)),
dtype=int)
    if len(indices) < n_pts:
        indices = np.pad(indices, (0, n_pts - len(indices)),
constant_values=indices[-1])

    for i, idx in enumerate(indices[:n_pts]):
        plt.scatter(fpr[idx], tpr[idx], c='blue', s=60)
        plt.text(fpr[idx] + 0.02, tpr[idx], point_labels[i],
fontweight='bold')

    plt.xlim([-0.02, 1.02])
    plt.ylim([-0.02, 1.02])
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.title(title)
    plt.legend()
    plt.grid(True, linestyle=':', alpha=0.7)
    plt.show()
```

```
def get_scores(model: cv2.ml.StatModel, X: np.ndarray) -> np.ndarray:
    """Возвращает непрерывные оценки для построения ROC."""
    if isinstance(model, (cv2.ml.SVM, cv2.ml.Boost)):
        _,            scores            =            model.predict(X,
flags=cv2.ml.STAT_MODEL_RAW_OUTPUT)
        return scores.flatten()
    elif isinstance(model, cv2.ml.RTrees):
        _, preds = model.predict(X)
        return preds.flatten().astype(np.float32)
    else:
        _, preds = model.predict(X)
        return preds.flatten().astype(np.float32)

def create_adaboost_model(weak_count: int = 20) -> cv2.ml.Boost:
    """AdaBoost через OpenCV."""
    model = cv2.ml.Boost.create()
    model.setBoostType(cv2.ml.BOOST_REAL)
    model.setWeakCount(weak_count)
    model.setWeightTrimRate(0.0)
    return model

def create_random_trees_model(n_trees: int = 50) -> cv2.ml.RTrees:
    """Random Trees."""
    model = cv2.ml.RTrees.create()
    model.setMaxDepth(10)
    model.setMinSampleCount(10)
    model.setRegressionAccuracy(0)
    model.setUseSurrogates(False)
    model.setMaxCategories(15)
    model.setPriors(np.array([]))
    model.setCalculateVarImportance(False)
    model.setActiveVarCount(1)
    model.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, n_trees, 1))
    return model

def create_svm_model() -> cv2.ml.SVM:
    """SVM через OpenCV."""
    model = cv2.ml.SVM.create()
    model.setType(cv2.ml.SVM_C_SVC)
    model.setKernel(cv2.ml.SVM_RBF)
    model.setC(1.0)
    model.setGamma(0.1)
    return model

# == Задание 6 ==
def run_adaboost_experiment(
        X: Features,
        y: Labels,
        train_sizes: t.List[int] = [150, 500, 1200, 5000],  # noqa
        weak_counts: t.List[int] = [20, 500],  # noqa
):
    split_idx = int(0.7 * len(X))
```

```
    X_train_full, X_test = X[:split_idx], X[split_idx:]
    y_train_full, y_test = y[:split_idx], y[split_idx:]

    for n_train in train_sizes:
        print(f"\n=== AdaBoost: train size = {n_train} ===")
        X_train = X_train_full[:n_train]
        y_train = y_train_full[:n_train]

        for wc in weak_counts:
            print(f"Training AdaBoost with {wc} weak classifiers...")
            model = create_adaboost_model(weak_count=wc)
            train_data       =       cv2.ml.TrainData.create(X_train,
cv2.ml.ROW_SAMPLE, y_train)
            model.train(train_data)

            model_path = f"adaboost_wc{wc}_n{n_train}.xml"
            save_model(model, model_path)

            _, y_pred = model.predict(X_test)
            y_pred = y_pred.astype(int)
            acc = np.mean(y_pred.flatten() == y_test.flatten())
            print(f"Test Accuracy: {acc:.4f}")

            compute_confusion_matrix(y_test, y_pred)

            y_true_flat = y_test.flatten()
            y_scores = get_scores(model, X_test)

# == Задание 7 ==
def run_rtrees_experiment(
        X: Features,
        y: Labels,
        train_sizes: t.List[int] = [150, 500, 1200, 5000],  # noqa
        n_trees_list: t.List[int] = [50, 500],  # noqa
):
    split_idx = int(0.7 * len(X))
    X_train_full, X_test = X[:split_idx], X[split_idx:]
    y_train_full, y_test = y[:split_idx], y[split_idx:]

    for n_train in train_sizes:
        print(f"\n=== Random Trees: train size = {n_train} ===")
        X_train = X_train_full[:n_train]
        y_train = y_train_full[:n_train]

        for n_trees in n_trees_list:
            print(f"Training Random Trees with ~{n_trees} trees...")
            model = create_random_trees_model(n_trees=n_trees)
            train_data       =       cv2.ml.TrainData.create(X_train,
cv2.ml.ROW_SAMPLE, y_train)
            model.train(train_data)

            _, y_pred = model.predict(X_test)
```

```
            y_pred = y_pred.astype(int)
            acc = np.mean(y_pred.flatten() == y_test.flatten())
            print(f"Test Accuracy: {acc:.4f}")

            compute_confusion_matrix(y_test, y_pred)

# == Задание 8 ==
def compare_rtrees_vs_svm(X: Features, y: Labels, train_size: int =
1200):
    split_idx = int(0.7 * len(X))
    X_train_full, X_test = X[:split_idx], X[split_idx:]
    y_train_full, y_test = y[:split_idx], y[split_idx:]

    X_train = X_train_full[:train_size]
    y_train = y_train_full[:train_size]

    rt_model = create_random_trees_model(n_trees=60)
    rt_model.train(cv2.ml.TrainData.create(X_train, cv2.ml.ROW_SAMPLE,
y_train))
    _, y_pred_rt = rt_model.predict(X_test)
    acc_rt = np.mean(y_pred_rt.flatten() == y_test.flatten())
    print(f"Random Trees (60 trees) Accuracy: {acc_rt:.4f}")

    svm_model = create_svm_model()
    svm_model.train(cv2.ml.TrainData.create(X_train,
cv2.ml.ROW_SAMPLE, y_train))
    _, y_pred_svm = svm_model.predict(X_test)
    acc_svm = np.mean(y_pred_svm.flatten() == y_test.flatten())
    print(f"SVM Accuracy: {acc_svm:.4f}")

    if acc_rt > acc_svm:
        print("Random Trees wins!")
    elif acc_svm > acc_rt:
        print("SVM wins!")
    else:
        print("Draw!")

if __name__ == "__main__":
    X, y = generate_labeled_data(n_points=10000)

    plot_split_idx = int(0.2 * len(X))
    X_plot, y_plot = X[:plot_split_idx], y[:plot_split_idx]

    std = 1.0
    mean_false, mean_true = 0.0, 2.0
    x = np.linspace(mean_false - 3.5 * std, mean_true + 3.5 * std,
10_000)
    false_pdf = (1 / np.sqrt(2 * np.pi * std ** 2)) * np.exp(-(x -
mean_false) ** 2 / (2 * std ** 2))
    true_pdf = (1 / np.sqrt(2 * np.pi * std ** 2)) * np.exp(-(x -
mean_true) ** 2 / (2 * std ** 2))
```

```
    thresholds = np.linspace(mean_false, mean_true, 7)
    labels = [chr(ord('a') + i) for i in range(7)]
    plot_class_distributions(x,   false_pdf,   true_pdf,   thresholds,
labels, "distribution.png")

    split_idx = int(0.8 * len(X_plot))
    X_train, X_test = X[:split_idx], X[split_idx:]
    y_train, y_test = y[:split_idx], y[split_idx:]
    ds = cv2.ml.TrainData.create(X_train, cv2.ml.ROW_SAMPLE, y_train)

    run_adaboost_experiment(X, y)
    adaboost = create_adaboost_model(1200)
    adaboost.train(ds)
    plot_decision_boundary(adaboost, X_train, y_train, title="AdaBoost
Decision Boundary (1200 trees)")
    _, predictions = adaboost.predict(X_test)
    plot_roc(y_test, predictions)

    run_rtrees_experiment(X, y)
    rtrees = create_random_trees_model(1200)
    rtrees.train(ds)
    plot_decision_boundary(rtrees,  X_train,  y_train,  title="RTrees
Decision Boundary (1200 trees)")
    _, predictions = rtrees.predict(X_test)
    plot_roc(y_test, predictions)

    svm = create_svm_model()
    svm.train(ds)
    plot_decision_boundary(svm, X_train, y_train, title="SVM Decision
Boundary")
    _, predictions = svm.predict(X_test)
    plot_roc(y_test, predictions)

    compare_rtrees_vs_svm(X, y)
```