|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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.

1. Изучить идеи метода центров тяжести (k-means) для кластеризации.
2. Рассмотреть прототипы функций и интерфейсы классов, реализующих перечисленные алгоритмы в библиотеке OpenCV.
3. Рассмотреть простые примеры использования указанного набора функций.
4. 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 , где элемент  равен количеству прецедентов выборки принадлежащих к классу  и отнесенных алгоритмом классификации к классу .
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?

Изображение выглядит как линия, График, диаграмма, скат

Содержимое, созданное искусственным интеллектом, может быть неверным.  
**Рис. 6.**Гауссовское распределение двух классов «false» и «true»

1. 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 используется 20%-ную вероятность для класса А; Если используется коэффициент 90% для класса А. Если используется 40%-ную вероятность для класса В; Если используется коэффициент 60% для класса В. Умножение вероятностей и 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 и у нас будет 8%-ная вероятность того, что точка будет помечена классом А и 92%-ная - что эта точка будет помечена классом В. Матрица вероятности того, что точка, помеченная классом A (а если нет, это класс В), равна:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

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.

1. 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.

1. Повторите упражнение 6, но используйте классификатор случайного леса с 50 и 500 деревьями.
2. Повторите упражнение 6, но на этот раз используйте 60 деревьев и сравните случайный лес с против SVM (машина опорных векторов).

**Результаты выполнения работы**

**Изображение выглядит как текст, снимок экрана, диаграмма, График

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**Рисунок 1** – Распределение двух классов

**Изображение выглядит как текст, снимок экрана, Шрифт

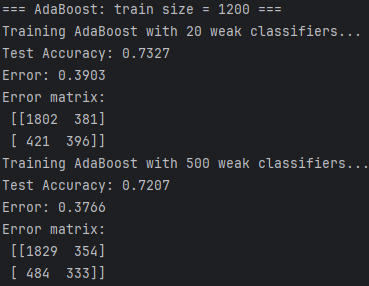
Содержимое, созданное искусственным интеллектом, может быть неверным.**

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

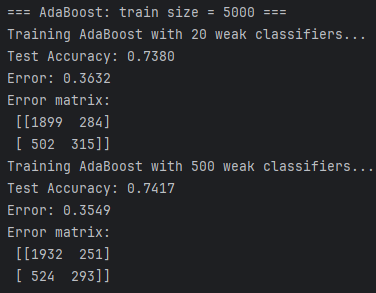
**Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.**

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

****

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

****

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

Изображение выглядит как снимок экрана, Красочность, текст, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

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

Изображение выглядит как линия, График, текст, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 7 –** Кривая ROC для AdaBoost

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

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

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

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

Изображение выглядит как текст, снимок экрана, Красочность, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

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

Изображение выглядит как линия, диаграмма, График, текст

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 11 –** Кривая ROC для RandomTrees

Изображение выглядит как снимок экрана, Красочность

Содержимое, созданное искусственным интеллектом, может быть неверным.

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

Изображение выглядит как текст, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

**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)