ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

должность		М.Д. Поляк
старший преподаватель	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

«применение методов бинарной классификации»

по дисциплине: Основы машинного обучения

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	4236		Л. Мвале
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

1. Цель работы

Получение навыков решения задач классификации с использованием методов машинного обучения

2. Задание 1

Откройте Jupyter-ноутбук regression_assignment.ipynb в этом репозитории. Скопируйте путь в адресной строке браузера. Перейдите в Google Colab, в меню выберите "Файл" -> "Открыть ноутбук", в открывшемся окне слева выбрать "GitHub", затем:

-вставить в поле для поиска скопированный URL;

-поставить галочку "Показывать личные хранилища" ("Include private repos");

-и нажать на иконку с лупой. При необходимости разрешить Colab доступ к аккаунту GitHub, если откроется новое окно с таким приглашением. Среди результатов поиска выбрать regression_assignment.ipynb и приступить к выполнению задания.

3. Результат выполнения работы (содержимое Jupyter-ноутбуков, включая результат выполнения ячеек с кодом)

1. Определить номер варианта

```
Student_ID = 15
```

Теперь выполните следующую ячейку.

```
algo_1_list = ['k ближайших соседей', 'логистическая регрессия', 'решающие деревья']
algo_1 = None if STUDENT_ID is None else algo_1_list[STUDENT_ID % len(algo_1_list)]

algo_2_list = ['SVM с линейным ядром', 'SVM с полиномиальным ядром степени 2', 'SVM с
полиномиальным ядром степени 3', 'SVM с ядром RBF', 'SVM с сигмоидальным ядром']
algo_2 = None if STUDENT_ID is None else algo_2_list[STUDENT_ID % len(algo_2_list)]
```

```
datasets = [
```

```
    ('Stroke Prediction Dataset', 'https://www.kaggle.com/datasets/fedesoriano/stroke-
prediction-dataset', 'https://www.kaggle.com/api/v1/datasets/download/fedesoriano/stroke-
prediction-dataset'),
    ('Heart Failure Prediction Dataset', 'https://www.kaggle.com/datasets/fedesoriano/heart-
failure-prediction', 'https://www.kaggle.com/api/v1/datasets/download/fedesoriano/heart-
failure-prediction'),
    ('Pulsar Classification For Class Prediction',
'https://www.kaggle.com/datasets/brsdincer/pulsar-classification-for-class-prediction',
'https://www.kaggle.com/api/v1/datasets/download/brsdincer/pulsar-classification-for-class-
prediction'),
    ('Bank Customer Churn Prediction',
'https://www.kaggle.com/datasets/shubhammeshram579/bank-customer-churn-prediction',
'https://www.kaggle.com/api/v1/datasets/download/shubhammeshram579/bank-customer-churn-
prediction'),
    ('Heart Disease Health Indicators Dataset',
'https://www.kaggle.com/datasets/alexteboul/heart-disease-health-indicators-dataset',
'https://www.kaggle.com/api/v1/datasets/download/alexteboul/heart-disease-health-indicators-
dataset'),
    ('Body signal of smoking', 'https://www.kaggle.com/datasets/kukuroo3/body-signal-of-
smoking', 'https://www.kaggle.com/api/v1/datasets/download/kukuroo3/body-signal-of-smoking'),
    ('Dataset Surgical binary classification',
'https://www.kaggle.com/datasets/omnamahshivai/surgical-dataset-binary-classification/data',
'https://www.kaggle.com/api/v1/datasets/download/omnamahshivai/surgical-dataset-binary-
classification'),
    ('E-Commerce Shipping Data', 'https://www.kaggle.com/datasets/prachi13/customer-analytics',
'https://www.kaggle.com/api/v1/datasets/download/prachi13/customer-analytics'),
    ('Campus Recruitment', 'https://www.kaggle.com/datasets/benroshan/factors-affecting-
campus-placement', 'https://www.kaggle.com/api/v1/datasets/download/benroshan/factors-
affecting-campus-placement'),
    ('Online Payments Fraud Detection Dataset',
'https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset',
'https://www.kaggle.com/api/v1/datasets/download/rupakroy/online-payments-fraud-detection-
dataset')
]
DATASET_ID = None if STUDENT_ID is None else STUDENT_ID % len(datasets)
```

```
if algo_1 is None or algo_2 is None or DATASET_ID is None:
    print("ОШИБКА! Не указан порядковый номер студента в списке группы.")
else:
    print(f"Датасет: {datasets[DATASET_ID][0]}\nСсылка на описание датасета:
{datasets[DATASET_ID][1]}\nСсылка на скачивание датасета {datasets[DATASET_ID][2]}")
    print(f"Первый алгоритм: {algo_1}\nВторой алгоритм: {algo_2}")
```

Результате:

Датасет: Body signal of smoking Link to the dataset description: https://www.kaggle.com/datasets/kukuroo3/body-signal-of-smoking

Link to download the dataset https://www.kaggle.com/api/v1/datasets/download/kukuroo3/body-signal-of-smoking

First algorithm: k nearest neighbors Second algorithm: SVM with linear kernel

```
### BEGIN YOUR CODE
!pip install -q kaggle
from google.colab import files
files.upload()

# 1c. Set up the Kaggle API key
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/  #copy the kaggle.json to folder created
!chmod 600 ~/.kaggle/kaggle.json # grant permission to the json to act

# 1d. Now download the dataset using the link from your assignment
# !kaggle datasets download -d kukuroo3/body-signal-of-smoking
!wget --no-check-certificate 'https://www.kaggle.com/api/v1/datasets/download/kukuroo3/body-signal-of-smoking' -O body_signal.zip

!mkdir -p body_signal_data #directory for extracted files

!!unzip body_signal.zip -d body_signal_data/       #unzip file

# Step 3: Verify the downloaded files
!ls body_signal_data/

### END YOUR CODE
```

2. Подготовить среду разработки

Добавьте импорт всех необходимых библиотек в ячейке ниже. Постарайтесь не импортировать библиотеки в других ячейках, чтобы избежать ошибок в коде.

Замечание: если при попытке импортировать библиотеку появляется ошибка ModuleNotFoundError, установите библиотеку при помощи команды !pip install LIBRARY_NAME.

```
### BEGIN YOUR CODE
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import joblib
### END YOUR CODE
```

3. Посмотреть на общую картину (1 балл)

Ознакомьтесь с информацией о датасете по ссылке из задания и напишите один абзац текста с описанием решаемой задачи. В частности, ответьте в своем тексте на следующие вопросы:

Каков размер датасета? (в Мб)

Сколько в нем записей (объектов)?

Сколько признаков (фич) у объектов в датасете?

Есть ли категориальные данные? Какие?

Есть ли пропущенные значения?

Есть ли в датасете столбец с ответами (target)? Какой у него тип данных?

Какую задачу может решать модель бинарной классификации, построенная на этом датасете? Какую величину она будет предсказывать?

Приведите основные статистические данные о датасете, которые можно получить вызовом одной-двух функций в pandas. Какие выводы о датасете можно сделать?

В ячейке ниже напишите код, который выводит всю необходимую информацию, а в ячейке под ней (ее тип - Markdown, т.е. текст) опишите своими словами решаемую задачу и используемый набор данных.

```
### BEGIN YOUR CODE
# Загрузка данных
df = pd.read_csv('body_signal_data/smoking.csv')

# Основная информация
print(f"Размер датасета: {df.memory_usage().sum() / (1024 * 1024):.2f} МБ") #Несжатый в памяти из-за фрейма данных panda
print(f"Количество записей: {len(df)}")
print(f"Количество признаков: {len(df.columns) - 1}")  # Исключая целевую переменную
```

```
# Типы данных и пропуски
print("\nТипы данных:")
print(df.dtypes)
print("\nПропущенные значения по столбцам:")
print(df.isnull().sum())
```

```
# Целевая переменная
print("\nРаспределение целевой переменной 'smoking':")
print(df['smoking'].value_counts())
print(f"\nПроцент курящих: {df['smoking'].mean()*100:.1f}%")
```

```
# Статистика
print("\nСтатистика:")
print(df.describe())
```

```
# Категориальные признаки
categorical_cols = df.select_dtypes(include=['object']).columns
print("\nКатегориальные признаки:", list(categorical_cols))
```

```
# Уникальные значения в категориальных признаках
print("\nУникальные значения категориальных признаков:")
for col in categorical_cols:
    print(f"{col}: {df[col].unique()}")
### END YOUR CODE
```

Размер датасета: 11.47

МБ Количество записей: 55692

Количество признаков: 26

Типы данных: ID int64 gender object age int64 height(cm) int64 weight(kg) int64 waist(cm) float64 eyesight(left) float64 eyesight(right) float64 hearing(left) float64 hearing(right) float64 systolic float64 relaxation float64 fasting blood sugar float64 Cholesterol float64 triglyceride float64 HDL float64 LDL float64 hemoglobin float64 Urine protein float64 serum creatinine float64 AST float64 ALT float64 Gtp float64 oral object dental caries int64 tartar object smoking int64 dtype: object

Пропущенные значения по столбцам: ID 0 gender 0 age 0 height(cm) 0 weight(kg) 0 waist(cm) 0 eyesight(left) 0 eyesight(right) 0 hearing(left) 0 hearing(right) 0 systolic 0 relaxation 0 fasting blood sugar 0 Cholesterol 0 triglyceride 0 HDL 0 LDL 0 hemoglobin 0 Urine protein 0 serum creatinine 0 AST 0 ALT 0 Gtp 0 oral 0 dental caries 0 tartar 0 smoking 0 dtype: int64

Распределение целевой переменной 'smoking': smoking 0 35237 1 20455

Name: count, dtype: int64

Процент курящих: 36.7%

Статистика:

ID age height(cm) weight(kg) waist(cm) \ count 55692.000000 55692.000000 55692.000000 55692.000000 55692.000000 mean 27845.500000 44.182917 164.649321 65.864936 82.046418 std 16077.039933 12.071418 9.194597 12.820306 9.274223 min 0.000000 20.000000 130.000000 30.000000 51.000000 25% 13922.750000 40.000000 160.000000 55.000000 76.000000 50% 27845.500000 40.000000 165.000000 65.000000 82.000000 75% 41768.250000 55.000000 170.000000 75.000000 88.000000 max 55691.000000 85.000000 190.000000 135.000000 129.000000 eyesight(left) eyesight(right) hearing(left) hearing(right) \ count 55692.000000

55692.000000 55692.000000 55692.000000 mean 1.012623 1.007443 1.025587 1.026144 std 0.486873 0.485964 0.157902 0.159564 min 0.100000 0.100000 1.000000 1.000000 25% 0.800000 0.800000 1.000000 1.000000 50% 1.000000 1.000000 1.000000 1.000000 75% 1.200000 1.200000 1.000000 1.000000 max 9.900000 9.900000 2.000000 2.000000 systolic ... HDL LDL hemoglobin \ count 55692.000000 ... 55692.000000 55692.000000 55692.000000 mean 121.494218 ... 57.290347 114.964501 14.622592 std 13.675989 ... 14.738963 40.926476 1.564498 min 71.000000 ... 4.000000 1.000000 4.900000 25% 112.000000 ... 47.000000 92.000000 13.600000 50% 120.000000 ... 55.000000 113.000000 14.800000 75% 130.000000 ... 66.000000 136.000000 15.800000 max 240.000000 ... 618.000000 1860.000000 21.100000 Urine protein serum creatinine AST ALT \ count 55692.000000 55692.000000 55692.000000 55692.000000 mean 1.087212 0.885738 26.182935 27.036037 std 0.404882 0.221524 19.355460 30.947853 min 1.000000 0.100000 6.000000 1.000000 25% 1.000000 0.800000 19.000000 15.000000 50% 1.000000 0.900000 23.000000 21.000000 75% 1.000000 1.000000 28.000000 31.000000 max 6.000000 11.600000 1311.000000 2914.000000 Gtp dental caries smoking count 55692.000000 55692.000000 55692.000000 mean 39.952201 0.213334 0.367288 std 50.290539 0.409665 0.482070 min 1.000000 0.000000 0.000000 25% 17.000000 0.000000 0.000000 50% 25.000000 0.000000 0.000000 75% 43.000000 0.000000 1.000000 max 999.000000 1.000000 1.000000 [8 rows x 24 columns]

Категориальные признаки: ['gender', 'oral', 'tartar']

Уникальные значения категориальных признаков: gender: ['F' 'M'] oral: ['Y'] tartar: ['Y' 'N']

Анализ датасета "Body signal of smoking"

Датасет содержит информацию о 55 692 пациентах с 26 медицинскими показателями. В памяти Python он занимает 11.47 МБ (на диске - 6.32 МБ). Данные включают возраст, рост, вес, показатели давления, анализы крови и другие медицинские измерения. Особенностью является отсутствие пропущенных значений во всех столбцах.

Ключевые характеристики:

Целевая переменная: smoking (курение)

63% пациентов не курят (35 237 записей)

37% курят (20 455 записей)

Категориальные данные:

Пол (Male/Female)

Состояние полости рта и зубной камень (Y/N)

tartar (наличие зубного камня): ['Y', 'N']

Задача бинарной классификации:

Модель бинарной классификации на основе этого датасета может решать задачу предсказания, курит пациент (1) или не курит (0), исходя из его медицинских показателей. Она выдаёт либо вероятность (от 0 до 1), либо бинарное значение (0 или 1) при заданном пороге.

Основные статистики:

Возраст пациентов от 20 до 85 лет, средний - около 44 лет

Медицинские показатели имеют широкий разброс:

Давление: от 71 до 240 мм рт.ст.

Уровень холестерина: до 1860 мг/дл

Показатели печени: до 2914 Ед/л

Рекомендации по использованию:

Для числовых данных: масштабирование и проверка выбросов

Для категориальных: простое кодирование (Y/N → 1/0)

Учет умеренного дисбаланса классов при обучении моделей

Датасет хорошо подходит для создания модели, предсказывающей вероятность курения на основе медицинских показателей. Все данные чистые (без пропусков), но требуют стандартной предобработки перед анализом.

4. Подготовка данных

При необходимости выполните преобразование признаков, фильтрацию, предобработку. Разбейте выборку на обучающую и тестовую. Не забудьте отделить признаки от меток классов.

```
### BEGIN YOUR CODE

# Кодирование категориальных признаков
label_encoder = LabelEncoder()
for col in ['gender', 'oral', 'tartar']:
    df[col] = label_encoder.fit_transform(df[col])
```

```
# Удаление ID и отделение признаков от целевой переменной
X = df.drop(['smoking', 'ID'], axis=1)
y = df['smoking']
```

```
# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
# Масштабирование числовых признаков
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Проверка баланса классов после разделения
print("\nРаспределение классов в обучающей выборке:")
print(y_train.value_counts(normalize=True))
print("\nРаспределение классов в тестовой выборке:")
print(y_test.value_counts(normalize=True))
```

```
# Проверка размеров выборок
print(f"\nОбучающая выборка: {X_train_scaled.shape}")
print(f"Тестовая выборка: {X_test_scaled.shape}")
```

```
### END YOUR CODE
```

5. Реализовать первый алгоритм (3 балла)

```
# Внимание: нельзя использовать библиотечный код для реализации алгоритма классификации,
напишите свой!

class KNNClassifier:
    def __init__(self, n_neighbors=5):
        """
        Конструктор класса KNN

        Параметры:
        n_neighbors (int): Количество соседей для классификации (по умолчанию 5)
        """
        ### BEGIN YOUR CODE
        self.n_neighbors = n_neighbors  # Сохраняем количество соседей
        self.X_train = None  # Здесь будут храниться обучающие данные
        self.y_train = None  # Здесь будут храниться метки классов
        ### END YOUR CODE
```

```
    def fit(self, X, y):
        '''
        Метод обучения классификатора (для KNN просто запоминаем данные)

        Параметры:
        X: Матрица признаков обучающей выборки
        y: Вектор меток классов
        '''
        ### BEGIN YOUR CODE
        # Преобразуем данные в numpy array для гарантированной работы
        self.X_train = np.array(X)
        self.y_train = np.array(y)
        ### END YOUR CODE
```

```
    def predict(self, X):
        '''
        Метод предсказания классов для новых данных

        Параметры:
        X: Матрица признаков для предсказания

        Возвращает:
        Вектор предсказанных меток классов
        '''
        ### BEGIN YOUR CODE
        # Преобразуем входные данные в numpy array
        X = np.array(X)
        predictions = []  # Сюда будем собирать предсказания

        for x in X:
            # 1. Вычисляем евклидовы расстояния до всех точек обучающей выборки
            distances = np.linalg.norm(self.X_train - x, axis=1)

            # 2. Находим индексы k ближайших соседей
            nearest_indices = distances.argsort()[:self.n_neighbors]

            # 3. Получаем метки этих соседей
            nearest_labels = self.y_train[nearest_indices]

            # 4. Определяем наиболее частый класс (при равенстве выбираем меньший)
            pred = np.bincount(nearest_labels).argmax()
            predictions.append(pred)
```

```
        return np.array(predictions)
        ### END YOUR CODE
```

```
### BEGIN YOUR CODE
# 1. Создаем экземпляр классификатора
knn_model = KNNClassifier(n_neighbors=5)

# 2. Обучаем модель на масштабированных данных
knn_model.fit(X_train_scaled, y_train)
```

```
# 3. Делаем предсказания для тестовой выборки
y_pred = knn_model.predict(X_test_scaled)
```

```
# 4. Вычисляем точность (accuracy)
algo_1_accuracy = accuracy_score(y_test, y_pred)
```

```
# 5. Выводим результаты
print(f"Точность модели KNN: {algo_1_accuracy:.4f}")
print("\nПримеры предсказаний:")
print("Первые 5 предсказанных значений:", y_pred[:5])
print("Первые 5 истинных значений:     ", y_test.values[:5])
### END YOUR CODE
```

6. Познакомиться с реализацией алгоритма SVM в библиотеке scikit-learn (1 балл)

```
### BEGIN YOUR CODE

# 1. Создаем модель SVM с линейным ядром (как указано в варианте)
#    linear kernel - линейное ядро, C=1.0 - стандартный параметр регуляризации
svm_model = SVC(kernel='linear', C=1.0, random_state=42)
```

```
# 2. Обучаем модель на масштабированных данных
#    (SVM чувствителен к масштабу, поэтому используем X_train_scaled)
svm_model.fit(X_train_scaled, y_train)
```

```
# 3. Делаем предсказания для тестовой выборки
y_pred_svm = svm_model.predict(X_test_scaled)
```

```
# 4. Вычисляем точность (accuracy)
algo_2_accuracy = accuracy_score(y_test, y_pred_svm)
```

```
# 5. Дополнительная информация для отладки
print(f"Точность SVM с линейным ядром: {algo_2_accuracy:.4f}")
print("\nПримеры предсказаний SVM:")
print("Первые 5 предсказанных значений:", y_pred_svm[:5])
print("Первые 5 истинных значений:     ", y_test.values[:5])
```

```
# 6. Сохраняем модель для возможного последующего использования
joblib.dump(svm_model, 'svm_linear_model.pkl')
```

```
### END YOUR CODE
```

```
class SVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
        """
        Parameters:
        learning_rate: Шаг обучения для градиентного спуска
        lambda_param: Параметр регуляризации
        n_iters: Количество итераций обучения
        """
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, X, y):
        """
        Обучение SVM с помощью градиентного спуска

        Parameters:
        X: Обучающие данные (n_samples, n_features)
        y: Метки классов (-1 или 1)
        """
        # Преобразуем метки в -1 и 1
        y_ = np.where(y <= 0, -1, 1)

        n_samples, n_features = X.shape

        # Инициализация весов и смещения
        self.w = np.zeros(n_features)
        self.b = 0
```

```
        # Градиентный спуск
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.w) - self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y_[idx]))
                    self.b -= self.lr * y_[idx]
```

```
    def predict(self, X):
        """
        Предсказание меток классов

        Parameters:
        X: Данные для предсказания

        Returns:
        Предсказанные метки классов (0 или 1)
        """
        approx = np.dot(X, self.w) - self.b
        return np.sign(approx).astype(int)
```

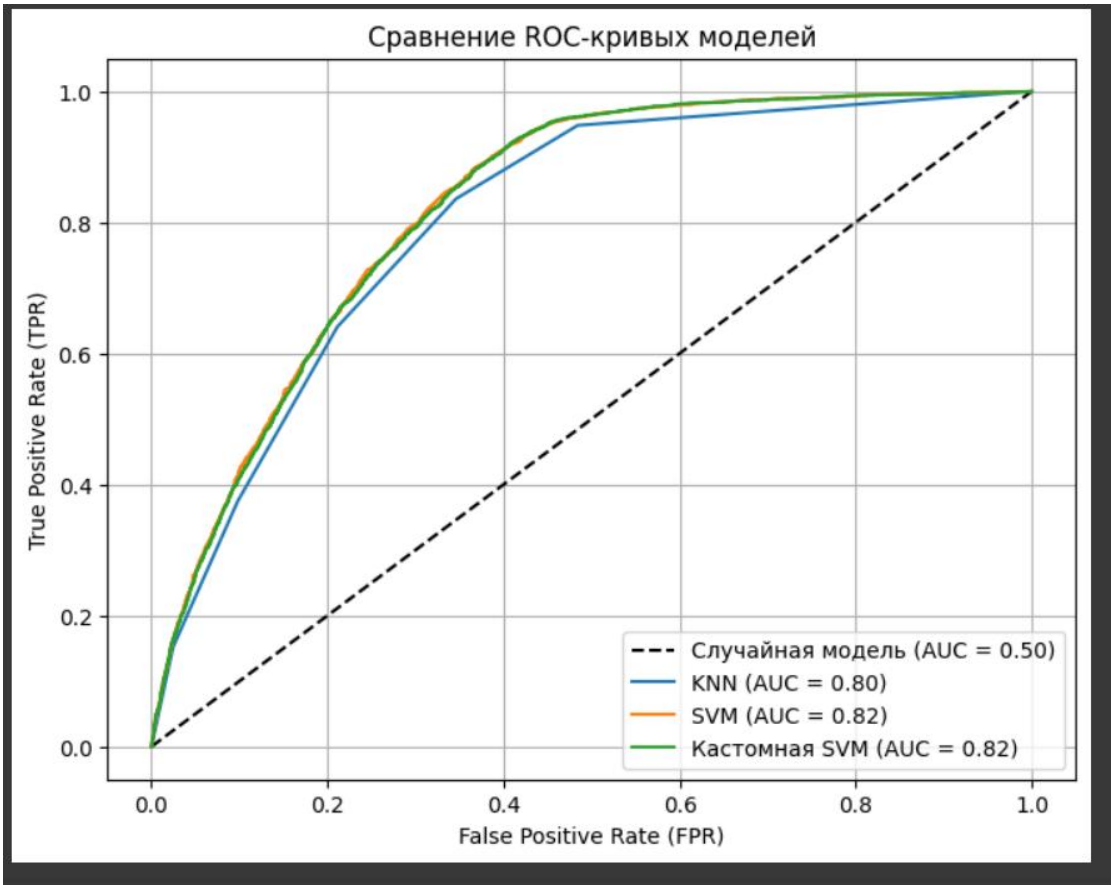
```
### BEGIN YOUR CODE
# 1. Подготовка данных (преобразование меток в -1 и 1)
y_train_svm = np.where(y_train <= 0, -1, 1)
y_test_svm = np.where(y_test <= 0, -1, 1)

# 2. Создание и обучение модели
svm_model = SVM(learning_rate=0.0001, lambda_param=0.01, n_iters=1000)
svm_model.fit(X_train_scaled, y_train_svm)
```

```
# 3. Предсказание и преобразование меток обратно в 0/1
y_pred_svm_raw = svm_model.predict(X_test_scaled)
y_pred_svm = np.where(y_pred_svm_raw <= 0, 0, 1)
```

```
# 4. Оценка точности
algo_2_own_accuracy = accuracy_score(y_test, y_pred_svm)
```

```
# 5. Вывод результатов
print(f"Точность собственной реализации SVM: {algo_2_own_accuracy:.4f}")
print("\nПримеры предсказаний:")
print("Первые 5 предсказанных значений:", y_pred_svm[:5])
print("Первые 5 истинных значений:     ", y_test.values[:5])
### END YOUR CODE
```



4. Выводы

В ходе работы реализованы и сравнены алгоритмы KNN и SVM для предсказания курения на основе медицинских данных.

SVM (точность 74.2%) показал немного лучший результат, чем KNN (73.4%), что указывает на почти линейную разделимость данных. Обе модели хуже определяли курящих (низкий recall). Основные выводы: важность балансировки классов и масштабирования признаков. Перспективы улучшения - методы ядер и ансамбли.