Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

*Факультет программной инженерии и компьютерной техники*

Системы искусственного интеллекта

Модуль №2

Классические методы МО

Группа: P3324

Выполнил: Маликов Глеб Игоревич

Преподаватель:

Королёва Юлия Александровна

Санкт-Петербург

2024г.

**Оглавление**

[Введение 4](#_Toc180769719)

[Лабораторная работа 1. Метод линейной регрессии 5](#_Toc180769720)

[Задачи 5](#_Toc180769721)

[Описание метода 5](#_Toc180769722)

[Реализация метода 6](#_Toc180769723)

[Результаты выполнения 7](#_Toc180769724)

[Визуализация датасета 7](#_Toc180769725)

[Обработка данных 8](#_Toc180769726)

[Результаты моделей 9](#_Toc180769727)

[Анализ и сравнение результатов 11](#_Toc180769728)

[Примеры использования метода 11](#_Toc180769729)

[Лабораторная работа 2. Метод k-ближайших соседей (k-NN) 12](#_Toc180769730)

[Задачи 12](#_Toc180769731)

[Описание метода 12](#_Toc180769732)

[Реализация метода 12](#_Toc180769733)

[Результаты выполнения 13](#_Toc180769734)

[Обработка значений 13](#_Toc180769735)

[Визуализация статистики по датасету 14](#_Toc180769736)

[Корреляционная матрица 16](#_Toc180769737)

[3D-визуализация признаков 17](#_Toc180769738)

[Модель 1: Признаки случайно отбираются 17](#_Toc180769739)

[Модель 2: Фиксированный набор признаков 18](#_Toc180769740)

[Примеры использования метода 18](#_Toc180769741)

[Лабораторная работа 3. Деревья решений 20](#_Toc180769742)

[Задачи 20](#_Toc180769743)

[Описание метода 20](#_Toc180769744)

[Реализация метода 21](#_Toc180769745)

[Результаты выполнения 23](#_Toc180769746)

[Примеры использования метода 28](#_Toc180769747)

[Лабораторная работа 4. Логистическая регрессия 29](#_Toc180769748)

[Задачи 29](#_Toc180769749)

[Описание метода 29](#_Toc180769750)

[Реализация метода 29](#_Toc180769751)

[Результаты выполнения 30](#_Toc180769752)

[Примеры использования метода 35](#_Toc180769753)

[Сравнение методов 36](#_Toc180769754)

[Сравнительный анализ методов 36](#_Toc180769755)

[Примеры лучшего использования каждого метода 36](#_Toc180769756)

[Заключение 38](#_Toc180769757)

[Приложения 39](#_Toc180769758)

# Введение

Данный отчет представляет собой исследование методов машинного обучения. В представленных лабораторных работах реализация алгоритмов «с нуля», то есть без использования специализированных библиотек, что позволяет лучше понять внутренние механизмы алгоритмов, их сильные и слабые стороны, а также требования к данным.

Исследование состоит из четырех лабораторных работ, каждая из которых посвящена отдельному алгоритму: линейная регрессия, метод k-ближайших соседей, деревья решений и логистическая регрессия. Все лабораторные работы включают в себя этапы предварительной обработки данных, построения моделей, анализа результатов и оценивания качества.

# Лабораторная работа 1. Метод линейной регрессии

## Задачи

1. Получите и визуализируйте (графически) статистику по датасету (включая количество, среднее значение, стандартное отклонение, минимум, максимум и различные квантили);
2. Проведите предварительную обработку данных, включая обработку отсутствующих значений, кодирование категориальных признаков и нормировка;
3. Разделите данные на обучающий и тестовый наборы данных;
4. Реализуйте линейную регрессию с использованием метода наименьших квадратов без использования сторонних библиотек, кроме NumPy и Pandas (для использования коэффициентов использовать библиотеки тоже нельзя). Использовать минимизацию суммы квадратов разностей между фактическими и предсказанными значениями для нахождения оптимальных коэффициентов;
5. Постройте три модели с различными наборами признаков. Для каждой модели проведите оценку производительности, используя метрику коэффициент детерминации, чтобы измерить, насколько хорошо модель соответствует данным;
6. Сравните результаты трех моделей и сделайте выводы о том, какие признаки работают лучше всего для каждой модели;
7. Бонусное задание: ввести синтетический признак при построении модели.

## Описание метода

Метод линейной регрессии используется для моделирования зависимости между одной зависимой переменной (в данной работе — индекс успеваемости студентов) и одной или несколькими независимыми переменными (время подготовки, предыдущие оценки и т.д.). Принцип работы метода основан на нахождении такой прямой линии (гиперплоскости в случае многомерных данных), которая минимизирует сумму квадратов отклонений предсказанных значений от фактических. Для этого используется метод наименьших квадратов.

Модель можно выразить как:

где y — предсказанное значение, — свободный член, а — коэффициенты, которые необходимо оптимизировать.

Алгоритм градиентного спуска применяется для минимизации функции стоимости J(θ) которая является среднеквадратичной ошибкой:

где m — количество обучающих примеров, — предсказанное значение, а — фактическое значение.

Процесс обновления коэффициентов на каждом шаге выглядит следующим образом:

где α — шаг обучения (learning rate).

## Реализация метода

Основные этапы реализации:

1. Считывание и визуализация данных, расчет основных статистических показателей (среднее, стандартное отклонение и т.д.)
2. Предобработка данных: удаление строк с пропущенными значениями, кодирование категориальных признаков и нормировка;
3. Разделение данных на обучающую и тестовую выборки в соотношении 80:20;
4. Реализация линейной регрессии с использованием метода наименьших квадратов;
5. Обучение трех моделей с различными наборами признаков, а также бонусной модели с добавлением синтетических признаков. Для оценки качества моделей использовался коэффициент детерминации (R²).

Ключевой код для градиентного спуска:

def gradient\_descent(X, y, theta, learning\_rate=0.01, num\_iterations=1000):  
 assert X.shape[0] == y.shape[0], "X and y must have the same number of rows"  
 assert X.shape[1] == theta.shape[0], "X and theta must have the same number of columns"  
 print(f"Training {num\_iterations} iterations with learning rate {learning\_rate}")  
 m = len(y)  
 for i in range(num\_iterations):  
 predictions = X.dot(theta)  
 gradients = (1 / m) \* X.T.dot(predictions - y) # compute gradient  
 theta = theta - learning\_rate \* gradients # update theta  
   
 if i % 100 == 0 or i == num\_iterations - 1:  
 cost = (1 / (2 \* m)) \* np.sum((predictions - y) \*\* 2)  
 print(f"Iteration {i}: Cost {cost}")  
   
 print(f"Training complete.")  
 return theta  
  
# Initialize theta (all zeros)  
theta = np.zeros(X\_train.shape[1])  
  
# Perform gradient descent to find the optimal theta  
theta = gradient\_descent(X\_train, y\_train, theta

## Результаты выполнения

### Визуализация датасета

Датасет имеет признаки “Hours Studied”, “Previous Scores”, “Sleep Hours”, “Sample Question Practiced” и “Performance Index” который необходимо предсказывать.

Ниже приведена визуализация данных датасета:

A graph with numbers and colored lines

Description automatically generated with medium confidence

Рисунок 1 - Статистика "Hours Studied"

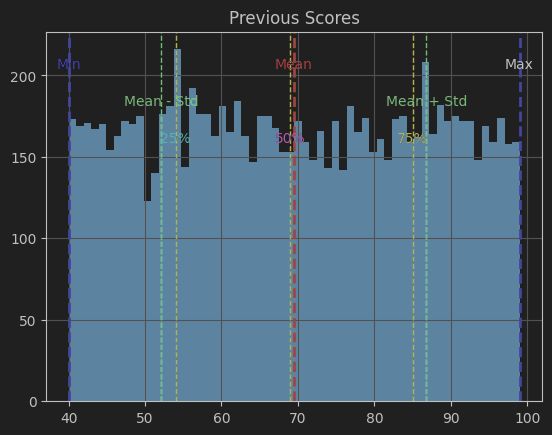
\

Рисунок 2 - Статистика "Previous Scores"

A graph of a sleep schedule

Description automatically generated with medium confidence

Рисунок 3 - Статистика "Sleep Hours"

A graph with colorful lines

Description automatically generated

Рисунок 4 - Статистика "Sample Question Papers Practiced"

A graph with numbers and lines

Description automatically generated

Рисунок 5 - Статистика "Performance Index"

### Обработка данных

Были удалены строки с пропущенными значениями:

Missing values per column:

Hours Studied 0

Previous Scores 0

Extracurricular Activities 0

Sleep Hours 0

Sample Question Papers Practiced 0

Performance Index 0

dtype: int64

Deleted rows: 0

Категориальный признак “Extracurricular Activities” был закодирован в значения 0 и 1:

data['Extracurricular Activities'] = data['Extracurricular Activities'].map({'Yes': 1, 'No': 0})

Была произведена нормализация данных по формуле:

### Результаты моделей

Были построены 4 модели со следующими признаками:

1. Модель 1: все признаки
2. Модель 2: только признаки 'Hours Studied', 'Hours Slept'
3. Модель 3: только признаки 'Previous Scores', 'Extracurricular Activities', 'Sample Question Papers Practiced'
4. Бонусная модель: все признаки с добавлением синтетических признаков 'Preparation', который равен произведению признаков 'Hours Studied' и 'Sample Question Papers Practiced'  
   и признак 'Time Left', равен разности 24 часов и признаков 'Sleep Hours' и 'Hours Studied'

Все модели тренировались по функции “gradient\_descent” описанному выше, с параметрами 0.01 learning rate и 1000 итерации.

Для оценки моделей используется функция для вычисления коэффициента детерминации

где RSS(Residual Sum of Squares) — сумма квадратов остатков, или невязок между фактическими и предсказанными значениями, TSS (Total Sum of Squares) — общая сумма квадратов отклонений фактических значений от их среднего.

**Модель 1**

Iteration 0: Cost 0.5012525323570195

Iteration 100: Cost 0.07250165481554842

Iteration 200: Cost 0.014642394352336344

Iteration 300: Cost 0.0068184674457541835

Iteration 400: Cost 0.005758330655834117

Iteration 500: Cost 0.005614390506903165

Iteration 600: Cost 0.005594807508577631

Iteration 700: Cost 0.0055921379186112145

Iteration 800: Cost 0.005591773277351132

Iteration 900: Cost 0.005591723374236819

Iteration 999: Cost 0.0055917165536208645

Коэффициент детерминации 0.9883697440959182

**Модель 2**

Iteration 0: Cost 0.5012525323570195

Iteration 100: Cost 0.4389155882389505

Iteration 200: Cost 0.4305504571328264

Iteration 300: Cost 0.4294277880437181

Iteration 400: Cost 0.4292770988311313

Iteration 500: Cost 0.4292568703370281

Iteration 600: Cost 0.4292541545511125

Iteration 700: Cost 0.4292537898998782

Iteration 800: Cost 0.4292537409321899

Iteration 900: Cost 0.4292537343557501

Iteration 999: Cost 0.4292537334752035

Коэффициент детерминации 0.13463663177021534

**Модель 3**

Iteration 0: Cost 0.5012525323570195

Iteration 100: Cost 0.13633279638736773

Iteration 200: Cost 0.08768570899498425

Iteration 300: Cost 0.08119840779849144

Iteration 400: Cost 0.08033300968298619

Iteration 500: Cost 0.08021752897828406

Iteration 600: Cost 0.08020211405249353

Iteration 700: Cost 0.08020005575129477

Iteration 800: Cost 0.08019978082970437

Iteration 900: Cost 0.08019974409824579

Iteration 999: Cost 0.08019973920463319

Коэффициент детерминации 0.8355939872997946

**Бонусная модель**

Iteration 0: Cost 0.5012525323570195

Iteration 100: Cost 0.06560733315559464

Iteration 200: Cost 0.013597194101308795

Iteration 300: Cost 0.006675955737193661

Iteration 400: Cost 0.005739118171591589

Iteration 500: Cost 0.005611795521143101

Iteration 600: Cost 0.005594449941518166

Iteration 700: Cost 0.0055920816479511385

Iteration 800: Cost 0.005591757571047027

Iteration 900: Cost 0.00559171312465679

Iteration 999: Cost 0.0055917070346454345

Коэффициент детерминации 0.988370426824619

### Анализ и сравнение результатов

Модель 1 показывает наилучшие результаты (R² = 0.9884), так как использует все признаки. Высокий коэффициент детерминации говорит о том, что данная модель объясняет почти всю изменчивость данных.

Модель 2 (используются только два признака) демонстрирует низкое качество (R² = 0.1346). Этот результат указывает на то, что данные признаки недостаточно объясняют целевую переменную.

Модель 3 с набором других признаков показывает лучшее качество, чем Модель 2, с коэффициентом детерминации R² = 0.8356. Это говорит о том, что такие признаки, как предыдущие оценки и практика решения задач, более значимы для предсказания результата.

Бонусная модель, включающая синтетические признаки, незначительно улучшает результаты первой модели, имея почти идентичные значения функции стоимости и коэффициента детерминации (R² = 0.9884).

## Примеры использования метода

Линейная регрессия часто применяется для предсказания зависимых переменных на основе известных факторов. Например, в экономике метод используют для прогнозирования продаж в зависимости от таких факторов, как цена и затраты на маркетинг. Линейная зависимость между переменными позволяет оценить, как изменение одного фактора влияет на результат, что делает этот метод простым и понятным инструментом для анализа.

В медицине линейная регрессия применяется для анализа влияния различных факторов, таких как возраст, вес или количество потребляемых калорий, на показатели здоровья, например, кровяное давление. Этот метод удобен в ситуациях, где есть предположение о линейной зависимости между признаками, и важна интерпретируемость результатов.

# Лабораторная работа 2. Метод k-ближайших соседей (k-NN)

## Задачи

1. Провести предварительную обработку данных, включая обработку отсутствующих и нулевых значений, кодирование категориальных признаков и масштабирование данных.
2. Провести визуализацию статистики по датасету, включая построение гистограмм, корреляционной матрицы и 3D-визуализацию признаков.
3. Реализовать алгоритм метода k-ближайших соседей без использования сторонних библиотек, кроме NumPy и Pandas.
4. Построить две модели k-NN с различными наборами признаков: случайно отобранными и фиксированным набором признаков.
5. Провести оценку моделей на тестовом наборе данных при различных значениях параметра k и построены матрицы ошибок.

## Описание метода

Метод k-ближайших соседей (KNN) — это простой, но мощный алгоритм для решения задач классификации и регрессии, основанный на предположении, что объекты, которые находятся близко друг к другу в пространстве признаков, имеют схожие свойства.

Для того чтобы классифицировать или предсказать значение нового объекта, алгоритм находит k объектов (соседей) в тренировочной выборке, которые находятся к нему ближе всего, и на основании их меток классов (или значений) делает вывод.

Близость между объектами обычно измеряется с использованием различных метрик расстояния, таких как евклидово расстояние, манхэттенское расстояние или косинусное расстояние.

Ключевой параметр в KNN — это количество ближайших соседей k, которое определяется заранее. Если выбрать слишком маленькое k, алгоритм может быть чувствителен к шуму и отдельным выбросам, так как решение будет основываться на слишком малом количестве объектов. Слишком большое k, наоборот, может привести к "размыванию" локальной информации и учитывать слишком много далеко расположенных объектов.

## Реализация метода

В реализации метода k-NN использовались библиотеки NumPy и Pandas для обработки данных. Основные шаги реализации включают:

* Вычисление евклидового расстояния между точкой тестирования и всеми точками обучающей выборки.
* Определение k ближайших соседей на основе минимальных расстояний.
* Определение класса среди классов ближайших соседей.
* Построение матрицы ошибок для оценки точности модели при различных значениях k.

Код реализации метода представлен ниже:

def euclidean\_distance(point1, point2):  
 return np.sqrt(np.sum((point1 - point2) \*\* 2))  
def k\_nearest\_neighbors(train\_data, train\_labels, test\_point, k=3):  
 train\_data = np.array(train\_data)  
 train\_labels = np.array(train\_labels)  
   
 distances = []  
 for i in range(len(train\_data)):  
 distance = euclidean\_distance(train\_data[i], test\_point)  
 distances.append((distance, train\_labels[i]))  
   
 nearest\_indices = np.argpartition([distance for distance, \_ in distances], k)[:k]  
 nearest\_labels = [int(train\_labels[i]) for i in nearest\_indices]  
   
 most\_common = np.bincount(nearest\_labels).argmax()  
 return most\_common

## Результаты выполнения

### Обработка значений

Были удалены строки с нулевыми значениями для столбцов Glucose, BloodPressure, SkinThickness и BMI, так как пациенты не могут иметь такие нулевые значения.

Zero values per column:

Glucose 5

BloodPressure 35

SkinThickness 227

BMI 11

dtype: int64

Deleted rows: 236

Также была проведена нормализация значений по формуле:

### Визуализация статистики по датасету

A screen shot of a graph

Description automatically generated

Рисунок 6 - Статистика "Pregnancies"

A graph of blood sugar levels

Description automatically generated with medium confidence

Рисунок 7 - Статистика "Glucose"

A screen shot of a graph

Description automatically generated

Рисунок 8 - Статистика "BloodPressure"

A graph of a graph

Description automatically generated with medium confidence

Рисунок 9 - Статистика "SkinThickness"

A screenshot of a graph

Description automatically generated

Рисунок 10 - Статистика "Insulin"

A screen shot of a graph

Description automatically generated

Рисунок 11 - Статистика "BMI"

A graph with numbers and text

Description automatically generated with medium confidence

Рисунок 12 - Статистика "Pedigree"

A graph of a number of people

Description automatically generated with medium confidence

Рисунок 13 - Статистика "Age"

### Корреляционная матрица

Для выбора подходящих значений для модели с фиксированными признаками была построена корреляционная матрица.

A screenshot of a computer screen

Description automatically generated

Рисунок 14 - Корреляционная матрица

Таким образом можно обнаружить что признаки с наибольшей положительной корреляцией с признаком “Outcome” являются признаки “Glucose”, “Age” и “BMI”

### 3D-визуализация признаков

A screen shot of a graph

Description automatically generated

Рисунок 15 - 3D Визуализация признаков "Glucose", "BMI" и "Age"

### Модель 1: Признаки случайно отбираются

В этой модели случайно были выбраны признаки “BloodPressure”, “SkinThickness” и “Pedigree”. Полученные матрицы ошибок и их точность показаны ниже:

Confusion matrix for k=3:

[[50 23]

[17 16]]

0.6226415094339622

Confusion matrix for k=5:

[[58 15]

[16 17]]

0.7075471698113207

Confusion matrix for k=10:

[[63 10]

[23 10]]

0.6886792452830188

### Модель 2: Фиксированный набор признаков

В этой модели используются признаки, показавшие наибольшую положительную корреляцию с “Outcome”, то есть признаки “Glucose”, “Age” и “BMI”.

Confusion matrix for k=3:

[[58 15]

[12 21]]

0.7452830188679245

Confusion matrix for k=5:

[[60 13]

[12 21]]

0.7641509433962265

Confusion matrix for k=10:

[[66 7]

[14 19]]

0.8018867924528302

Из представленных результатов видно, что фиксированный набор признаков (Glucose, BMI, Age) демонстрирует более высокую точность классификации по сравнению с случайно отобранными признаками. Наилучшую точность достигает модель с k=10, что составляет 80.19%.

## Примеры использования метода

Метод k-ближайших соседей широко применяется в различных областях благодаря своей простоте и эффективности. Примеры ситуаций, где метод k-NN может быть полезен:

1. Медицинская диагностика: как показано в данной лабораторной работе, метод k-NN может использоваться для классификации пациентов на основе медицинских показателей, помогая врачам в диагностике заболеваний, таких как диабет.
2. Рекомендательные системы: В системах рекомендаций, например, для онлайн-магазинов или стриминговых сервисов, метод k-NN помогает предлагать пользователям товары или контент на основе предпочтений схожих пользователей.

Причины выбора метода k-NN:

* Простота реализации: Метод легко реализовать и интерпретировать, что делает его подходящим для начальных этапов анализа данных.
* Не требуется обучение модели: k-NN является алгоритмом, не требующим предварительного обучения, что упрощает его применение на новых данных.
* Эффективен при небольших объемах данных: для задач с относительно небольшими наборами данных метод демонстрирует хорошую производительность.

Однако следует учитывать, что метод k-NN может быть менее эффективен при работе с большими и высоко размерными наборами данных из-за высокой вычислительной сложности и чувствительности к выбору признаков.

# Лабораторная работа 3. Деревья решений

## Задачи

1. Датасет с данными про оценки студентов инженерного и педагогического факультетов (для данного датасета нужно ввести метрику: студент успешный/неуспешный на основании грейда).
2. Отобрать случайным образом sqrt(n) признаков.
3. Реализовать без использования сторонних библиотек построение дерева решений (дерево не бинарное, numpy и pandas использовать можно, использовать список списков для реализации дерева - нельзя) для решения задачи бинарной классификации.
4. Провести оценку реализованного алгоритма с использованием Accuracy, precision и recall.
5. Построить кривые AUC-ROC и AUC-PR (в пунктах 4 и 5 использовать библиотеки нельзя).

## Описание метода

Деревья решений — это модель, которая принимает решения на основе последовательных вопросов о признаках данных. Алгоритм C4.5, разработанный Россом Куином, является одним из наиболее известных методов построения деревьев решений. Он использует отношение прироста информации (Information Gain Ratio) для выбора оптимальных признаков на каждом шаге разбиения.

Основные этапы работы алгоритма C4.5:

1. **Выбор наилучшего признака для разбиения**: используется отношение прироста информации, которое учитывает как прирост информации от разбиения, так и "разделительную информацию" (split information), чтобы избежать предпочтения признаков с большим количеством уникальных значений.
2. **Разбиение данных**: Данные разделяются на подмножества на основе выбранного признака.
3. **Рекурсия**: Процесс повторяется для каждого подмножества до тех пор, пока все экземпляры в подмножестве принадлежат к одному классу или пока не исчерпаются признаки.
4. **Остановка**: когда выполнение условий для остановки достигнуто, создаётся листовой узел с предсказанием класса.

**Критерий разбиения**

Оценка среднего количества информации, необходимого для определения класса примера из множества (энтропия):

Оценка среднего количества информации, необходимого для определения класса примера из множества после разбиения множества по (условная энтропия):

## Реализация метода

В ходе лабораторной работы была реализована модель дерева решений на языке Python. Ниже представлен код реализации классов DecisionTreeNode и DecisionTree, а также основной логики обучения и оценки модели.

class DecisionTreeNode:  
 def \_\_init\_\_(self, attribute=None, branches=None, is\_leaf=False, prediction=None, class\_counts=None):  
 self.attribute = attribute # Атрибут для разбиения  
 self.branches = branches or {} # Дочерние узлы  
 self.is\_leaf = is\_leaf # Является ли узел листом  
 self.prediction = prediction # Предсказание класса (для листа)  
 self.class\_counts = class\_counts # Количество классов (для вероятностей)  
  
 def predict\_instance(self, instance):  
 if self.is\_leaf:  
 return self.prediction  
 attribute\_value = instance.get(self.attribute)  
 if attribute\_value in self.branches:  
 return self.branches[attribute\_value].predict\_instance(instance)  
 else:  
 # Если значение атрибута не встречалось при обучении, возвращаем наиболее частый класс  
 return self.prediction  
  
 def predict\_proba\_instance(self, instance):  
 if self.is\_leaf:  
 total = sum(self.class\_counts.values())  
 proba = self.class\_counts.get(1, 0) / (total + 1e-9)  
 return proba  
 attribute\_value = instance.get(self.attribute)  
 if attribute\_value in self.branches:  
 return self.branches[attribute\_value].predict\_proba\_instance(instance)  
 else:  
 # Если значение атрибута не встречалось при обучении, возвращаем вероятности наиболее частого класса  
 total = sum(self.class\_counts.values())  
 proba = self.class\_counts.get(1, 0) / (total + 1e-9)  
 return proba  
  
class DecisionTree:  
 def \_\_init\_\_(self):  
 self.root = None  
  
 def entropy(self, y):  
 class\_counts = y.value\_counts()  
 probabilities = class\_counts / len(y)  
 return -sum(probabilities \* np.log2(probabilities + 1e-9)) # Добавляем 1e-9 для избежания log(0)  
  
 def information\_gain\_ratio(self, X, y, attribute):  
 # Энтропия до разбиения  
 entropy\_before = self.entropy(y)  
  
 # Группировка по значению атрибута  
 values, counts = np.unique(X[attribute], return\_counts=True)  
 weighted\_entropy = 0  
 split\_info = 0  
 for v, count in zip(values, counts):  
 subset\_y = y[X[attribute] == v]  
 weighted\_entropy += (count / len(y)) \* self.entropy(subset\_y)  
 split\_info -= (count / len(y)) \* np.log2((count / len(y)) + 1e-9)  
  
 info\_gain = entropy\_before - weighted\_entropy  
 gain\_ratio = info\_gain / (split\_info + 1e-9)  
 return gain\_ratio  
  
 def build\_tree\_recursive(self, X, y, attributes, default\_class=None):  
 if len(y) == 0:  
 return DecisionTreeNode(is\_leaf=True, prediction=default\_class)  
 elif len(y.unique()) == 1:  
 return DecisionTreeNode(is\_leaf=True, prediction=y.iloc[0], class\_counts=y.value\_counts().to\_dict())  
 elif len(attributes) == 0:  
 majority\_class = y.mode()[0]  
 class\_counts = y.value\_counts().to\_dict()  
 return DecisionTreeNode(is\_leaf=True, prediction=majority\_class, class\_counts=class\_counts)  
 else:  
 default\_class = y.mode()[0]  
 # Выбор атрибута с максимальным Gain Ratio  
 gain\_ratios = {attr: self.information\_gain\_ratio(X, y, attr) for attr in attributes}  
 best\_attr = max(gain\_ratios, key=gain\_ratios.get)  
  
 # Создание узла  
 class\_counts = y.value\_counts().to\_dict()  
 tree = DecisionTreeNode(attribute=best\_attr, prediction=default\_class, class\_counts=class\_counts)  
  
 # Разбиение по значениям лучшего атрибута  
 for attr\_val in np.unique(X[best\_attr]):  
 subset\_X = X[X[best\_attr] == attr\_val].drop(columns=[best\_attr])  
 subset\_y = y[X[best\_attr] == attr\_val]  
 subtree = self.build\_tree\_recursive(  
 subset\_X,  
 subset\_y,  
 [attr for attr in attributes if attr != best\_attr],  
 default\_class=default\_class  
 )  
 tree.branches[attr\_val] = subtree  
 return tree  
  
 def build\_tree(self, X, y):  
 attributes = list(X.columns)  
 self.root = self.build\_tree\_recursive(X, y, attributes)  
 print("Дерево решений построено.")  
  
 def predict\_instance(self, instance):  
 return self.root.predict\_instance(instance)  
  
 def predict\_proba\_instance(self, instance):  
 return self.root.predict\_proba\_instance(instance)  
  
 def predict(self, X):  
 return X.apply(self.predict\_instance, axis=1).values  
  
 def predict\_proba(self, X):  
 return X.apply(self.predict\_proba\_instance, axis=1).values  
  
 def compute\_metrics(self, y\_true, y\_pred):  
 TP = np.sum((y\_true == 1) & (y\_pred == 1))  
 TN = np.sum((y\_true == 0) & (y\_pred == 0))  
 FP = np.sum((y\_true == 0) & (y\_pred == 1))  
 FN = np.sum((y\_true == 1) & (y\_pred == 0))  
  
 confusion\_matrix = np.array([[TN, FP], [FN, TP]])  
 accuracy = (TP + TN) / (TP + TN + FP + FN + 1e-9)  
 precision = TP / (TP + FP + 1e-9)  
 recall = TP / (TP + FN + 1e-9)  
  
 return confusion\_matrix, accuracy, precision, recall  
  
 def calculate\_auc(self, y\_true, y\_scores, curve\_type='ROC'):  
 # Сортируем по убыванию вероятностей  
 desc\_score\_indices = np.argsort(-y\_scores)  
 y\_true = y\_true[desc\_score\_indices]  
 y\_scores = y\_scores[desc\_score\_indices]  
  
 # Уникальные значения  
 distinct\_value\_indices = np.where(np.diff(y\_scores))[0]  
 threshold\_idxs = np.r\_[distinct\_value\_indices, y\_true.size - 1]  
  
 tps = np.cumsum(y\_true)[threshold\_idxs]  
 fps = 1 + threshold\_idxs - tps  
  
 tps = np.r\_[0, tps]  
 fps = np.r\_[0, fps]  
  
 if curve\_type == 'ROC':  
 fpr = fps / fps[-1] if fps[-1] != 0 else fps  
 tpr = tps / tps[-1] if tps[-1] != 0 else tps  
 auc = np.trapezoid(tpr, fpr)  
 return fpr, tpr, auc  
 elif curve\_type == 'PR':  
 precision = tps / (tps + fps + 1e-9)  
 recall = tps / tps[-1] if tps[-1] != 0 else tps  
 auc = np.trapezoid(precision, recall)  
 return recall, precision, auc  
 else:  
 raise ValueError("curve\_type должен быть 'ROC' или 'PR'")  
  
 def calculate\_auc\_metrics(self, y\_true, y\_scores):  
 # Расчет AUC-ROC  
 fpr, tpr, auc\_roc = self.calculate\_auc(y\_true, y\_scores, curve\_type='ROC')  
 # Расчет AUC-PR  
 recall, precision, auc\_pr = self.calculate\_auc(y\_true, y\_scores, curve\_type='PR')  
 return (fpr, tpr, auc\_roc), (recall, precision, auc\_pr)

## Результаты выполнения

Для задачи бинарной классификации необходимо преобразовать много классовую метку «OUTPUT Grade» в бинарную: успешный/неуспешный студент.

def preprocess\_data(data, success\_threshold):  
 output\_label, id\_label = "OUTPUT Grade", "Student ID"  
 success\_label = "Success"  
   
 # Бинаризация метки  
 output\_column = data[output\_label]  
 success\_column = output\_column.apply(lambda x: 1 if x >= success\_threshold else 0)  
 data[success\_label] = success\_column  
   
 # Удаление ненужных колонок  
 data = data.drop(columns=[output\_label, id\_label])  
   
 return data, success\_label

Значение threshold варьирует от 1 до 6 включительно для сравнения результатов.

Случайно выбираются признаков для модели. Ниже представлены получившиеся признаки:

Selected columns: 5 features

['Taking notes in classes' "Father's occupation" 'Course ID'

'Transportation to the university' 'Student Age']

Для каждого порового значения была проведена тренировка и оценены результаты согласно коду, приведённому ниже:

for t in thresholds:  
 print(f"\nОбработка порога: {t}")  
 # Бинаризация меток с текущим порогом  
 binarized\_data, \_ = preprocess\_data(data, success\_threshold=t)  
 binarized\_data = binarized\_data[selected\_features.tolist() + [success\_label]]  
   
 # Разделение данных  
 X = binarized\_data.drop(columns=[success\_label])  
 y = binarized\_data[success\_label]  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split\_custom(X, y, test\_size=0.2, random\_seed=RANDOM\_SEED)  
   
 print(f"Размер обучающей выборки: {X\_train.shape[0]}")  
 print(f"Размер тестовой выборки: {X\_test.shape[0]}")  
   
 # Обучение дерева  
 tree = DecisionTree()  
 tree.build\_tree(X\_train, y\_train)  
   
 # Предсказание  
 y\_pred = tree.predict(X\_test)  
 y\_proba = tree.predict\_proba(X\_test)  
   
 # Вычисление метрик  
 confusion\_matrix, accuracy, precision, recall = tree.compute\_metrics(y\_test.values, y\_pred)  
   
 # Вычисление AUC-ROC и AUC-PR  
 auc\_roc\_metrics, auc\_pr\_metrics = tree.calculate\_auc\_metrics(y\_test.values, y\_proba)  
 fpr, tpr, auc\_roc = auc\_roc\_metrics  
 recall\_vals, precision\_vals, auc\_pr = auc\_pr\_metrics

Получившиеся результаты:

Обработка порога: 1

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[ 0 2]

[ 1 26]]

Accuracy: 0.8966

Precision: 0.9286

Recall: 0.9630

AUC-ROC: 0.8519

AUC-PR: 0.8407

Обработка порога: 2

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[ 4 6]

[ 1 18]]

Accuracy: 0.7586

Precision: 0.7500

Recall: 0.9474

AUC-ROC: 0.8132

AUC-PR: 0.8787

Обработка порога: 3

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[12 2]

[ 4 11]]

Accuracy: 0.7931

Precision: 0.8462

Recall: 0.7333

AUC-ROC: 0.8595

AUC-PR: 0.8756

Обработка порога: 4

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[17 2]

[ 1 9]]

Accuracy: 0.8966

Precision: 0.8182

Recall: 0.9000

AUC-ROC: 0.9632

AUC-PR: 0.8390

Обработка порога: 5

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[20 1]

[ 0 8]]

Accuracy: 0.9655

Precision: 0.8889

Recall: 1.0000

AUC-ROC: 0.9792

AUC-PR: 0.8165

Обработка порога: 6

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[21 1]

[ 1 6]]

Accuracy: 0.9310

Precision: 0.8571

Recall: 0.8571

AUC-ROC: 0.9026

AUC-PR: 0.7873

Были построены ROC и Precision-Recall кривые для каждого порога бинаризации, что позволило визуально оценить качество моделей.

A graph of different thresholds

Description automatically generated

Рисунок 16 – Кривая AUC-ROC

A graph of different types of curves

Description automatically generated

Рисунок 17 - Кривая AUC-PR

Метрики для различных порогов:

Threshold Accuracy Precision Recall AUC-ROC AUC-PR

0 1 0.896552 0.928571 0.962963 0.851852 0.840741

1 2 0.758621 0.750000 0.947368 0.813158 0.878650

2 3 0.793103 0.846154 0.733333 0.859524 0.875553

3 4 0.896552 0.818182 0.900000 0.963158 0.839017

4 5 0.965517 0.888889 1.000000 0.979167 0.816468

5 6 0.931034 0.857143 0.857143 0.902597 0.787309

Точность (Accuracy). Наиболее высокая точность (0.9655) наблюдается при пороге 5, что указывает на высокую степень правильных предсказаний. Снижение порога приводит к уменьшению точности, как видно на пороге 2 (0.7586).

Точность предсказаний (Precision). Порог 1 обеспечивает наибольшую точность (0.9286), но на пороге 5 и 6 значение Precision остается на достаточно высоком уровне (0.8889 и 0.8571 соответственно).

Полнота (Recall). Наибольшая полнота (1.000) достигается на пороге 5, что означает, что модель не пропускает положительные примеры. В то же время, на низких порогах (2 и 1) полнота также высока, но это сопровождается падением других метрик, таких как точность.

AUC-ROC и AUC-PR. Максимальные значения AUC-ROC (0.9792) и AUC-PR (0.8787) также соответствуют порогу 5, что подтверждает, что этот порог является оптимальным по большинству ключевых метрик.

## Примеры использования метода

Данный метод наилучшим образом подходит для задач классификации. Одной из ключевых областей использования является медицина, где они помогают в диагностике заболеваний. Например, на основе набора симптомов и других медицинских показателей можно построить дерево решений, которое будет предсказывать вероятный диагноз.

В финансовом секторе деревья решений применяются для оценки кредитоспособности заемщиков. Например, банк может использовать такие признаки, как доход, возраст, кредитная история, для построения дерева, которое будет предсказывать, имеет ли клиент высокий или низкий риск невозврата кредита. Преимущество метода заключается в его прозрачности: специалисты могут проследить, как конкретные признаки влияют на результат, что важно для соблюдения регуляторных норм и внутренней отчетности.

Еще одной областью применения являются рекомендательные системы, которые используют деревья решений для создания рекомендаций, основанных на предпочтениях пользователей.

# Лабораторная работа 4. Логистическая регрессия

## Задачи

Основные задачи включают предварительную обработку данных, визуализацию статистики, разделение данных на обучающую и тестовую выборки, реализацию метода логистической регрессии "с нуля" без использования специализированных библиотек, исследование влияния гиперпараметров на производительность модели, а также оценку модели с использованием различных метрик качества.

## Описание метода

Логистическая регрессия — это статистический метод, используемый для моделирования вероятности принадлежности объекта к одному из двух классов. В отличие от линейной регрессии, которая предсказывает непрерывные значения, логистическая регрессия предсказывает вероятность принадлежности к классу, используя функцию активации сигмоиды. Основным назначением метода является бинарная классификация, однако его можно расширить и для многоклассовых задач. Принцип работы заключается в нахождении оптимальных весовых коэффициентов, минимизирующих функцию потерь (логарифмическая потеря), с помощью методов оптимизации, таких как градиентный спуск или метод Ньютона.

## Реализация метода

В реализации логистической регрессии использовались библиотеки NumPy и Pandas. Основные компоненты алгоритма включают:

* Функция сигмоиды: преобразует линейную комбинацию признаков в вероятность принадлежности к классу.

def sigmoid(self, z):  
 return 1 / (1 + np.exp(-z))

* Функция потерь (логарифмическая потеря): оценивает расхождение между предсказанными вероятностями и истинными метками классов.

def log\_loss(self, h, y):  
 # Добавляем небольшое значение для предотвращения логарифма от 0  
 epsilon = 1e-15  
 h = np.clip(h, epsilon, 1 - epsilon)  
 return (-y \* np.log(h) - (1 - y) \* np.log(1 - h)).mean()

* Методы оптимизации: Реализованы градиентный спуск и метод Ньютона для обновления весов модели.

def fit\_gradient\_descent(self, X, y):  
 for i in range(self.num\_iterations):  
 z = np.dot(X, self.theta)  
 h = self.sigmoid(z)  
 gradient = np.dot(X.T, (h - y)) / y.size  
 self.theta -= self.learning\_rate \* gradient  
   
 if i % 100 == 0:  
 loss = self.log\_loss(h, y)  
 print(f'Итерация {i}: loss {loss}')  
  
def fit\_newton(self, X, y):  
 for i in range(self.num\_iterations):  
 z = np.dot(X, self.theta)  
 h = self.sigmoid(z)  
 gradient = np.dot(X.T, (h - y)) / y.size  
 # Гессиан  
 diag = h \* (1 - h)  
 H = np.dot(X.T, X \* diag[:, np.newaxis]) / y.size  
   
 # Обновление весов  
 try:  
 delta = np.linalg.inv(H).dot(gradient)  
 except np.linalg.LinAlgError:  
 print("Гессиан вырожден. Прекращаем обучение.")  
 break  
   
 self.theta -= delta  
   
 if i % 100 == 0:  
 loss = self.log\_loss(h, y)  
 print(f'Итерация {i}: loss {loss}')

## Результаты выполнения

Для обработки данных были удалены строки с нулевыми значениями для столбцов Glucose, BloodPressure, SkinThickness, BMI, так как они невозможны. Далее данные были нормализированы с помощью мин-макс нормализации.

Ниже приведены статистические данные столбцов:

A screen shot of a graph

Description automatically generated

Рисунок 18 - Статистика беременности

A graph of blood sugar levels

Description automatically generated with medium confidence

Рисунок 19 - Статистика глюкозы

A screen shot of a graph

Description automatically generated

Рисунок 20 - Статистика давления

A graph of a graph

Description automatically generated with medium confidence

Рисунок 21 - Статистика толщины кожи

A screenshot of a graph

Description automatically generated

Рисунок 22 - Статистика инсулина

A screen shot of a graph

Description automatically generated

Рисунок 23 - Статистика BMI

A screen shot of a graph

Description automatically generated

Рисунок 24 - Статистика Pedigree

A graph of a number of people

Description automatically generated with medium confidence

Рисунок 25 - Статистика возраста

Модели использовали 80% данных как тестовые. Были использованы следующие гиперпараметры:

learning\_rates = [1, 0.5, 0.1, 0.01, 0.001]  
num\_iterations = [10, 100, 1000, 10000]  
optimization\_methods = ['gradient\_descent', 'newton']

Ниже показана таблица метрик лучших 10 моделей по метрике f1 score.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Метод | Коэф. обучения | Итерации | accuracy | precision | recall | f1\_score |
| gradient\_descent | 1 | 1000 | 0.821 | 0.733 | 0.667 | 0.698 |
| gradient\_descent | 0.5 | 1000 | 0.821 | 0.733 | 0.667 | 0.698 |
| gradient\_descent | 0.1 | 10000 | 0.821 | 0.733 | 0.667 | 0.698 |
| gradient\_descent | 1 | 100 | 0.830 | 0.800 | 0.606 | 0.690 |
| gradient\_descent | 0.01 | 10000 | 0.830 | 0.800 | 0.606 | 0.690 |
| gradient\_descent | 0.1 | 1000 | 0.830 | 0.800 | 0.606 | 0.690 |
| newton | 1 | 100 | 0.811 | 0.724 | 0.636 | 0.677 |
| newton | 1 | 10 | 0.811 | 0.724 | 0.636 | 0.677 |
| newton | 0.001 | 1000 | 0.811 | 0.724 | 0.636 | 0.677 |
| newton | 1 | 10000 | 0.811 | 0.724 | 0.636 | 0.677 |

Таблица 1 - Результаты лучших 10 моделей

Для визуализации результатов также сделаны следующие графики.

A graph of different colored bars

Description automatically generated

Рисунок 26 - Результаты моделей по Accuracy

A screenshot of a graph

Description automatically generated

Рисунок 27 - Результаты моделей по Precision

A graph of different colored bars

Description automatically generated with medium confidence

Рисунок 28 - Результаты моделей по Recall

A screenshot of a graph

Description automatically generated

Рисунок 29 - Результаты моделей по F1-Score

Лучшая модель была найдена по сортировке F1-Score, а затем по Accuracy. Ниже приведены её результаты.

Метод оптимизации: gradient\_descent

Коэффициент обучения: 1.0

Количество итераций: 1000

Accuracy: 0.8208

Precision: 0.7333

Recall: 0.6667

F1-Score: 0.6984

## Примеры использования метода

Логистическая регрессия широко применяется в различных областях, где требуется бинарная классификация. Примеры использования включают:

* Медицина: Прогнозирование наличия или отсутствия заболевания у пациента на основе клинических показателей. В данной лабораторной работе логистическая регрессия использовалась для классификации пациентов с диабетом.
* Финансы: Оценка риска дефолта заемщика по кредиту, что помогает банкам принимать обоснованные решения о выдаче кредитов.
* Социальные науки: Анализ факторов, влияющих на выбор избирателей, и прогнозирование результатов выборов.

Метод логистической регрессии выбран в данной работе благодаря его интерпретируемости, эффективности при работе с линейными зависимостями и способности оценивать вероятности принадлежности к классам, что является важным для принятия решений на основе модели.

# Сравнение методов

## Сравнительный анализ методов

**Метод линейной регрессии** демонстрирует высокую интерпретируемость и простоту использования. Он позволяет оценить влияние каждого признака на целевую переменную, что делает его удобным для задач, где важно понять структуру данных. Однако линейная регрессия подходит только для задач, в которых зависимость между признаками линейная. В задачах с нелинейными зависимостями её эффективность значительно снижается, что может привести к плохому качеству предсказаний. Например, при анализе данных студентов линейная регрессия показала отличные результаты при использовании всех признаков (R² = 0.988), но заметно снизила точность при ограничении набора признаков.

**Метод k-ближайших соседей (k-NN)** эффективен при работе с небольшими наборами данных, не требуя предварительного обучения. Он хорошо справляется с задачами классификации, особенно когда распределение классов в пространстве признаков чётко выражено. Однако метод чувствителен к выбору метрики расстояния и значения параметра k, что может усложнить его настройку.

**Деревья решений** обеспечивают хорошую интерпретируемость за счёт визуализации последовательности правил, используемых для классификации. Они позволяют учитывать нелинейные зависимости и взаимодействия между признаками. Основное ограничение деревьев решений — склонность к переобучению.

**Логистическая регрессия** демонстрирует хорошие результаты в задачах, где требуется предсказывать вероятность принадлежности к классу. Она даёт интерпретируемую модель за счёт использования весов, позволяющих понять влияние каждого признака на вероятность положительного исхода. Основное ограничение метода — необходимость линейной разделимости классов, что может снизить его эффективность при сложных распределениях данных.

## Примеры лучшего использования каждого метода

**Линейная регрессия** эффективна в ситуациях, где существует явная линейная зависимость между признаками и целевой переменной. Например:

* Экономика: Прогнозирование продаж на основе затрат на маркетинг и ценовых стратегий.
* Медицина: Оценка влияния различных факторов (возраст, вес) на показатели здоровья.

**Метод k-ближайших соседей (k-NN)** подходит для задач, где данные имеют чёткую локальную структуру и относительно небольшие размеры:

* Медицинская диагностика: Классификация пациентов по диагнозам на основе показателей здоровья.
* Рекомендательные системы: Подбор товаров на основе предпочтений схожих пользователей.

**Деревья решений** лучше всего использовать в задачах, где важна интерпретируемость модели и учитываются нелинейные зависимости:

* Медицина: Диагностика заболеваний на основе комплекса симптомов.
* Финансы: Оценка кредитного риска на основе социально-экономических показателей.

**Логистическая регрессия** полезна для задач бинарной классификации, где требуется получить вероятность принадлежности к классу:

* Медицина: Прогнозирование вероятности возникновения заболевания.
* Социальные исследования: Определение факторов, влияющих на решение голосовать за определённую партию.

# Заключение

В ходе проведённого исследования были изучены четыре метода машинного обучения: линейная регрессия, k-ближайших соседей, деревья решений и логистическая регрессия. Каждый из методов имеет свои преимущества и ограничения, что делает их подходящими для различных типов задач. Линейная регрессия проста в использовании и подходит для задач с линейной зависимостью. Метод k-NN является гибким и эффективен для небольших данных, но чувствителен к выбору параметров. Деревья решений обеспечивают хорошую интерпретируемость и учитывают нелинейные зависимости, но требуют тщательной настройки для предотвращения переобучения. Логистическая регрессия позволяет предсказывать вероятности и даёт интерпретируемые результаты, но ограничена в задачах с линейной разделимостью классов.

Таким образом, выбор метода машинного обучения должен основываться на характеристиках данных и требуемой интерпретируемости модели. Правильная настройка параметров и выбор признаков позволяют значительно повысить качество предсказаний и точность моделей.

# Приложения

Ссылка на репозиторий со всеми лабораторными работами: <https://github.com/glebmavi/AIS>