## Анализ HTML и CSS кода

### I. Структура и содержание HTML

HTML код определяет структуру и содержание веб-страницы, представляя собой основу для отображения интерактивного инструмента анализа данных. Давайте разберём логику его построения:

1. **Базовая структура:**
   * Документ начинается с объявления <!DOCTYPE html> и тега <html lang="ru">, определяющих тип и язык документа (русский).
   * Далее следуют блоки <head> и <body>, содержащие метаинформацию и видимое содержимое страницы соответственно.
2. **Заголовок и контейнер:**
   * Внутри <body> размещается главный заголовок <h1> с текстом “EDA анализ и построение бинарного дерева решений”.
   * Основное содержимое обернуто в <div class="container">, ограничивающий ширину страницы для удобства просмотра.
3. **Аккордеон:**
   * Центральным элементом страницы является аккордеон (<div class="accordion">), содержащий несколько секций (<div class="accordion-item">).
   * Каждая секция аккордеона имеет заголовок (<h2>) и сворачиваемый/разворачиваемый блок содержимого (<div class="accordion-collapse">).
4. **Секции аккордеона:**
   * Первая секция “Загрузка данных” содержит поле для выбора файла (<input type="file">) и JavaScript код, обрабатывающий загрузку и чтение данных.
   * Последующие секции (“EDA анализ 1”, “EDA анализ 2”, “Генетический алгоритм”, “Результирующее дерево решений”) изначально пусты и заполняются динамически с помощью JavaScript по мере выполнения анализа.
5. **Динамическое содержимое:**
   * JavaScript код создает и добавляет на страницу различные элементы:
     + Кнопки для запуска алгоритмов и отображения результатов.
     + Списки для выбора признаков.
     + Графики (гистограммы, корреляционная матрица, дерево решений), используя библиотеки D3.js и Plot.
   * Такой подход позволяет создавать интерактивный интерфейс, реагирующий на действия пользователя и отображающий результаты анализа динамически.

### II. Стилизация с помощью CSS

В данном коде используется CSS фреймворк Bootstrap для стилизации элементов страницы. Подключение Bootstrap осуществляется путем добавления ссылки на файл bootstrap.min.css в блоке <head>.

1. **Готовые стили Bootstrap:**
   * Bootstrap предоставляет набор готовых стилей для распространенных элементов веб-страницы, таких как контейнеры, кнопки, формы, аккордеоны и т.д.
   * В коде используются классы стилей Bootstrap, например:
     + container - для ограничения ширины содержимого страницы.
     + accordion - для создания аккордеона.
     + btn - для стилизации кнопок.
   * Применение готовых стилей упрощает разработку и обеспечивает единообразный внешний вид элементов.
2. **Пользовательские стили:**
   * Код не содержит явного определения пользовательских стилей, однако JavaScript динамически добавляет стили к некоторым элементам, например:
     + margin-top, margin-bottom, display, font-size - для label в истории поколений.
     + margin-top, border, width - для div в истории поколений.
   * Динамическое добавление стилей позволяет гибко управлять внешним видом элементов в зависимости от данных и результатов анализа.
3. **Адаптивность:**
   * Bootstrap автоматически адаптирует страницу к различным размерам экрана (компьютеры, планшеты, смартфоны) благодаря использованию медиа-запросов.
   * Это обеспечивает удобство просмотра и использования инструмента на различных устройствах.

В целом, CSS код, основанный на Bootstrap, обеспечивает простой и адаптивный дизайн страницы, фокусируясь на удобстве использования и восприятия информации. Динамическое добавление стилей с помощью JavaScript позволяет гибко управлять внешним видом элементов в зависимости от контекста.

## Анализ JavaScript кода

JavaScript код в данном примере реализует всю логику интерактивного инструмента: от загрузки и обработки данных до визуализации результатов анализа и работы генетического алгоритма. Давайте проанализируем его основные функции:

**1. Инициализация и обработка событий:**

* В начале кода определяются глобальные переменные для хранения данных, настроек, результатов анализа и истории.
* К элементу загрузки файла (<input type="file">) привязывается обработчик события, срабатывающий при выборе файла пользователем.

**2. Чтение и подготовка данных:**

* При выборе файла запускается функция, которая считывает данные из CSV-файла с помощью библиотеки Papa Parse.
* Полученные данные преобразуются в удобный для обработки формат:
  + Извлекаются заголовки столбцов.
  + Строковые значения преобразуются в числовые, где это необходимо.
* Происходит подготовка интерфейса: создаются динамические элементы для выбора признаков, запуска анализа и отображения результатов.

**3. EDA анализ:**

* По умолчанию отображаются гистограммы для всех числовых признаков, позволяющие визуально оценить их распределение.
* Пользователь может выбирать признаки для анализа с помощью чекбоксов.
* При нажатии кнопки “Построить корреляционную матрицу” строится и отображается матрица, визуализирующая корреляции между выбранными признаками.
* Используются библиотеки D3.js и Plot для визуализации графиков.

**4. Генетический алгоритм:**

* Алгоритм запускается при нажатии кнопки “Запустить генетический алгоритм”.
* Пользователь может настраивать параметры алгоритма (размер популяции, вероятности мутации и др.) с помощью ползунков.
* В процессе работы алгоритма на странице отображается информация о текущем поколении и лучшем найденном решении, а также визуализируется дерево решений.
* Для визуализации дерева решений используется библиотека D3-Graphviz.

**5. Визуализация результатов:**

* Результатом работы генетического алгоритма является дерево решений, которое визуализируется на странице.
* Дерево представлено в виде графа, где каждый узел отображает условие разделения данных, а листья - принадлежность к классу.
* Пользователь может просматривать историю работы алгоритма, где отображаются лучшие решения, найденные в каждом поколении.

**В целом, JavaScript код реализует следующий сценарий:**

1. Загрузка и подготовка данных.
2. Предоставление инструментов для EDA анализа.
3. Запуск генетического алгоритма с возможностью настройки параметров.
4. Визуализация процесса обучения алгоритма.
5. Отображение результатов анализа в виде дерева решений и истории обучения.

Код написан с использованием современных подходов к разработке веб-приложений, таких как:

* **Модульность:** разбиение кода на функции для повышения читаемости и повторного использования.
* **Динамическое обновление DOM:** изменение содержимого страницы без перезагрузки для создания интерактивного интерфейса.
* **Асинхронность:** использование async/await для обработки длительных операций (например, чтение файла) без блокировки интерфейса.

Важно отметить, что код содержит подробные комментарии, которые значительно упрощают его понимание.

## Анализ кода: Разведочный анализ данных (EDA)

В данном коде EDA представлен как начальный этап перед запуском генетического алгоритма. Он позволяет пользователю изучить данные, выявить закономерности и взаимосвязи между признаками, что может быть полезно для выбора оптимальных параметров алгоритма или интерпретации результатов.

**1. Визуализация распределений:**

* Сразу после загрузки данных строится набор гистограмм – по одной для каждого числового признака.
* Гистограммы позволяют быстро оценить:
  + **Форму распределения:** симметричное, скошенное, унимодальное, бимодальное и т.д.
  + **Центр распределения:** где находится большинство значений.
  + **Разброс значений:** насколько сильно данные разбросаны вокруг центра.
  + **Наличие выбросов:** значений, сильно отличающихся от основной массы данных.
* Дополнительно на гистограммы накладываются линии, отображающие распределение значений для каждого класса (0 и 1), что помогает визуально оценить, насколько хорошо признак разделяет данные по классам.

**2. Интерактивный выбор признаков:**

* Пользователь может выбирать, какие признаки будут использованы для построения модели с помощью набора чекбоксов.
* Это позволяет:
  + Сконцентрироваться на наиболее информативных признаках.
  + Исключить признаки, которые могут внести шум или искажения в модель.

**3. Корреляционная матрица:**

* При нажатии кнопки “Построить корреляционную матрицу” вычисляется и отображается матрица, показывающая попарные корреляции между выбранными признаками.
* Корреляция измеряется коэффициентом корреляции Пирсона, который принимает значения от -1 до 1:
  + **Близкие к 1 значения** указывают на сильную положительную корреляцию (при увеличении одного признака другой также увеличивается).
  + **Близкие к -1 значения** указывают на сильную отрицательную корреляцию (при увеличении одного признака другой уменьшается).
  + **Значения близкие к 0** говорят об отсутствии линейной зависимости между признаками.
* Визуализация корреляционной матрицы помогает:
  + Идентифицировать группы сильно коррелирующих признаков (что может быть полезно для уменьшения размерности данных).
  + Обнаружить признаки, которые не коррелируют с целевой переменной (и, возможно, исключить их из анализа).

**В целом, EDA в данном коде реализует базовый набор инструментов для анализа данных:**

* Визуализация распределений отдельных признаков.
* Интерактивный выбор признаков.
* Анализ корреляций между признаками.

Важно отметить, что EDA – это итеративный процесс, и в зависимости от результатов анализа, пользователю может понадобиться вернуться к предыдущим этапам, например, для загрузки новых данных, выбора других признаков или изменения параметров алгоритма.

## Анализ кода: Генетический алгоритм

JavaScript код реализует генетический алгоритм для построения бинарного дерева решений, оптимизируя его структуру для наилучшего разделения данных на классы. Разберём реализованные в коде методы и приёмы:

**1. Представление генома:**

* **Структура:** Геном представлен в виде рекурсивной структуры данных - бинарного дерева.
  + Каждый узел дерева содержит:
    - **Признак (class):** на основе которого производится разделение данных в этом узле.
    - **Операцию сравнения (operation):** функция, сравнивающая значение признака с пороговым значением (threshold). Операция включает в себя как сам порог, так и направление сравнения - больше (>) или меньше (<).
    - **Ссылки на дочерние узлы (left, right):** куда переходит алгоритм в зависимости от результата сравнения.
* **Создание:**
  + Начальный геном (create\_genome) генерируется случайным образом: выбираются случайный признак, случайный порог и направление сравнения.
  + Дочерние узлы (left, right) изначально пустые и создаются по мере необходимости в процессе мутации.

**2. Функция приспособленности (score\_ant):**

* **Цель:** Оценить качество (эффективность) дерева решений, представленного геномом.
* **Механизм:**
  + Для каждого объекта данных из набора выполняется проход по дереву решений, начиная с корня.
  + В каждом узле проверяется условие разделения (сравнение значения признака с порогом).
  + В зависимости от результата сравнения происходит переход к левому или правому дочернему узлу.
  + Процесс повторяется, пока не будет достигнут лист дерева (узел без потомков).
  + Результат классификации для данного объекта сравнивается с его истинным классом.
* **Подсчёт:**
  + Функция подсчитывает количество объектов, классифицированных верно.
* **Штраф за сложность:**
  + От результата вычитается штраф, пропорциональный размеру (количеству узлов) и глубине дерева.
  + Это сделано для того, чтобы избежать переобучения и получить более простое и интерпретируемое дерево.

**3. Операторы генетического алгоритма:**

* **Селекция (select\_ants):**
  + Из текущей популяции (“муравьев”) выбирается определенное количество лучших особей (с наибольшим значением функции приспособленности) для дальнейшего размножения.
* **Мутация (mutate\_genome, mutate\_ant):**
  + Применяются различные виды мутаций для внесения изменений в геном:
    - **Изменение признака или порога:** Случайно выбирается новый признак или генерируется новый порог для узла.
    - **Добавление узла:** К случайно выбранному узлу добавляется левый или правый потомок.
    - **Удаление поддерева:** Случайно выбранное поддерево удаляется, а его место занимает лист.
    - **Замена поддерева:** Случайным образом генерируется новое поддерево и заменяет существующее.
* **Скрещивание:** В коде не реализовано.

**4. Параметры алгоритма:**

* Код позволяет пользователю настраивать следующие параметры с помощью ползунков:
  + **Количество особей (cnt\_ants):** Размер популяции.
  + **Количество копируемых особей (copy\_ants):** Сколько лучших особей переходит в следующее поколение без изменений (элитизм).
  + **Вероятность мутации признака (p\_mutate\_ch):** Вероятность изменить признак в узле при мутации.
  + **Максимальное отклонение при мутации (p\_mutate\_op\_rate):** Насколько сильно может измениться порог при мутации.
  + **Множитель штрафа за размер дерева (p\_penalty):** Коэффициент, регулирующий силу штрафа за количество узлов в дереве.
  + **Степень влияния глубины дерева на штраф (p\_penalty\_depth):** Насколько сильно глубина дерева влияет на штраф.

**5. Остановка алгоритма:**

* Алгоритм работает в бесконечном цикле, пока пользователь не нажмёт кнопку “Остановить генетический алгоритм”.
* При остановке отображается лучшее найденное решение (дерево с максимальной приспособленностью) и история обучения.

**Особенности реализации:**

* Код использует метафору “муравьиного алгоритма”, где каждый “муравей” представляет собой дерево решений.
* Вместо явного скрещивания используется копирование лучших особей (элитизм) и мутация для создания нового поколения.
* Механизм штрафа за сложность дерева помогает бороться с переобучением и получать более простые и интерпретируемые модели.

## Анализ кода:

### V. Построение дерева решений

После завершения работы генетического алгоритма, код визуализирует лучшее найденное дерево решений. Процесс построения визуализации включает:

1. **Преобразование генома в формат DOT:**
   * Функция genome\_to\_dot рекурсивно обходит структуру дерева, представленную геномом.
   * Для каждого узла формируется строка в формате DOT, описывающая:
     + Уникальный идентификатор узла (генерируется случайным образом).
     + Текстовую метку узла, включающую:
       - Название признака, используемого для разделения данных.
       - Операцию сравнения (больше > или меньше <).
       - Значение порога, используемое в операции.
     + Связи с дочерними узлами (если есть).
   * Для листьев (узлов без потомков) указываются метки классов (“Рис Гонён” и “Рис Жасмин”).
2. **Визуализация с помощью D3-Graphviz:**
   * Функция svg\_from\_genome получает на вход геном и преобразует его в формат DOT.
   * Создаётся новый элемент <div> для размещения визуализации.
   * Используя библиотеку D3-Graphviz, DOT-строка преобразуется в SVG-изображение графа, представляющего дерево решений.
   * Полученное SVG-изображение добавляется в созданный ранее <div>, который, в свою очередь, отображается на странице.

**Результат:** Пользователь видит визуальное представление дерева решений, где:

* Каждый узел представляет собой условие разделения данных на основе значения признака.
* Ребра графа соответствуют исходам сравнения в узлах (больше или меньше порога).
* Листья (конечные узлы) отображают классы, к которым относятся объекты, удовлетворяющие всем условиям на пути от корня до листа.

### VI. Дополнительные особенности

Код обладает некоторыми дополнительными особенностями, повышающими его удобство и информативность:

1. **Интерактивные элементы управления:**
   * Ползунки позволяют пользователю настраивать параметры генетического алгоритма “на лету” без необходимости изменения кода.
   * Кнопки запускают и останавливают алгоритм, предоставляя пользователю контроль над процессом обучения.
2. **Отображение истории обучения:**
   * Код сохраняет информацию о лучшем найденном дереве решений в каждом поколении генетического алгоритма.
   * При нажатии кнопки “Показать историю” на странице отображается серия визуализаций деревьев, найденных в процессе обучения, с указанием их приспособленности.
   * Это позволяет пользователю:
     + Проследить за эволюцией дерева решений в процессе обучения.
     + Увидеть, как меняется структура дерева с каждым поколением.
     + Оценить скорость сходимости алгоритма к оптимальному решению.
3. **Комментарии в коде:**
   * Код содержит подробные комментарии, объясняющие логику работы каждой функции и важных блоков кода.
   * Это значительно упрощает понимание кода и его модификацию при необходимости.

## Матрица корреляций: инструмент для понимания взаимосвязей данных

В процессе анализа данных часто важно понимать, как различные признаки (переменные) связаны между собой. Существует ли закономерность: когда один признак увеличивается, другой тоже увеличивается (или уменьшается)? Ответить на эти вопросы помогает **корреляция**.

### Что такое корреляция?

Корреляция – это статистическая мера, которая показывает силу и направление связи между двумя переменными. Она выражается числом, называемым **коэффициентом корреляции**, который принимает значения от -1 до 1.

* **Коэффициент корреляции 1** означает идеальную положительную корреляцию: когда одна переменная растёт, другая растёт строго пропорционально.
* **Коэффициент корреляции -1** означает идеальную отрицательную корреляцию: когда одна переменная растёт, другая уменьшается строго пропорционально.
* **Коэффициент корреляции 0** означает отсутствие линейной корреляции: изменение одной переменной никак не связано с изменением другой.

Важно отметить, что корреляция **не** означает причинно-следственную связь. Даже если две переменные сильно коррелируют, это не значит, что одна из них является причиной другой.

### Матрица корреляций

Когда у нас есть много признаков, удобно представить информацию об их корреляции в виде таблицы, которая называется **матрицей корреляций**.

* **Строки и столбцы:** Каждая строка и каждый столбец матрицы соответствуют одному признаку.
* **Ячейки:** В ячейке на пересечении строки *i* и столбца *j* записывается коэффициент корреляции между признаком *i* и признаком *j*.
* **Диагональ:** Ячейки на главной диагонали всегда равны 1, так как каждый признак идеально коррелирует сам с собой.
* **Симметричность:** Матрица корреляций симметрична относительно главной диагонали, поскольку корреляция между признаком *i* и признаком *j* равна корреляции между признаком *j* и признаком *i*.

### Зачем нужна матрица корреляций?

Матрица корреляций – это мощный инструмент анализа данных, который позволяет:

1. **Визуализировать взаимосвязи:** Сразу увидеть, какие признаки связаны между собой, а какие – нет. Это особенно полезно при большом количестве признаков, когда сложно анализировать корреляции попарно.
2. **Выявить мультиколлинеарность:** Это ситуация, когда несколько признаков сильно коррелируют между собой. Мультиколлинеарность может быть проблемой для некоторых алгоритмов машинного обучения, так как ухудшает интерпретируемость модели и снижает ее устойчивость.
3. **Отбор признаков:** Корреляция может быть использована для выбора наиболее информативных признаков для построения модели. Например, можно исключить признаки, которые слабо коррелируют с целевой переменной или сильно коррелируют друг с другом.
4. **Понимание структуры данных:** Анализ корреляций может помочь выявить скрытые закономерности и зависимости в данных, что может быть полезно для формулирования гипотез и дальнейшего исследования.

### Визуализация матрицы корреляций

Часто матрицу корреляций представляют в виде тепловой карты, где:

* **Цвет ячейки** отражает значение коэффициента корреляции:
  + Синий/зелёный - отрицательная корреляция.
  + Белый - отсутствие корреляции.
  + Красный/оранжевый - положительная корреляция.
* **Интенсивность цвета** показывает силу корреляции: чем ярче цвет, тем сильнее связь.

### Пример использования в коде

В анализируемом коде матрица корреляций используется для визуализации взаимосвязей между признаками выбранными пользователем для анализа. Это позволяет:

* Оценить, какие признаки наиболее сильно связаны между собой.
* Выявить группы признаков, которые несут похожую информацию (что может быть полезно для уменьшения размерности данных).
* Сделать предположения о том, какие признаки могут быть наиболее важными для прогнозирования целевой переменной.

## Гистограммы: взгляд на распределение данных

В коде, который мы анализируем, гистограммы играют важную роль на этапе разведочного анализа данных (EDA). Они предоставляют визуальное представление о распределении значений каждого числового признака. Давайте разберёмся, как они работают и почему это важно.

### Что такое гистограмма?

Гистограмма — это тип графика, который показывает распределение частот встречаемости значений в наборе данных.

* **Ось X (горизонтальная):** Разбита на интервалы (bins), которые охватывают весь диапазон значений признака.
* **Ось Y (вертикальная):** Показывает, сколько объектов данных попадает в каждый интервал.

Таким образом, гистограмма наглядно демонстрирует, какие значения встречаются чаще, а какие — реже.

### Гистограммы в коде

В данном коде сразу после загрузки данных для каждого числового признака строится гистограмма. На графике дополнительно отображаются линии, показывающие распределение значений признака отдельно для каждого класса (0 и 1).

**Пример:**

* Представим, что мы анализируем данные о сортах риса, и один из признаков — это длина зерна.
* Гистограмма для этого признака покажет, сколько сортов имеют длину зерна в диапазоне, например, от 5 до 6 мм, сколько — от 6 до 7 мм и т.д.
* Дополнительно на графике будут линии, показывающие распределение длины зерна для сортов “Рис Гонён” (класс 0) и “Рис Жасмин” (класс 1).

### Зачем нужны гистограммы?

1. **Понимание формы распределения:**
   * Гистограмма позволяет увидеть, как распределены значения признака:
     + **Симметричное распределение:** значения распределены равномерно вокруг центра.
     + **Скошенное распределение:** значения сконцентрированы с одной стороны от центра.
     + **Унимодальное распределение:** один ярко выраженный пик.
     + **Бимодальное распределение:** два ярко выраженных пика.
   * Форма распределения может быть важна для выбора подходящего алгоритма машинного обучения.
2. **Обнаружение выбросов:**
   * Выбросы — это значения, которые сильно отличаются от основной массы данных.
   * На гистограмме выбросы выглядят как отдельные столбики, расположенные далеко от основного распределения.
   * Выбросы могут быть ошибками в данных или указывать на интересные особенности, требующие отдельного изучения.
3. **Сравнение распределений:**
   * Сравнивая гистограммы для разных классов (как в данном коде), можно понять, насколько хорошо признак разделяет данные по классам.
   * Если распределения для разных классов сильно пересекаются, то признак плохо подходит для классификации.
   * И наоборот, если распределения практически не пересекаются, то признак является хорошим кандидатом для построения модели.

### Гистограммы в контексте задачи

В данном случае гистограммы используются для:

1. **Первичного знакомства с данными:** Позволяют быстро оценить характер распределения каждого признака.
2. **Выявления потенциально информативных признаков:** Признаки, гистограммы которых показывают существенные различия в распределении для разных классов, могут быть особенно полезны для построения модели классификации.
3. **Поиска выбросов:** Выбросы могут негативно влиять на работу некоторых алгоритмов, поэтому их важно обнаружить на ранних этапах анализа.

В целом, гистограммы — это простой, но мощный инструмент EDA, который помогает получить ценную информацию о данных перед построением модели.

## Как ползунки влияют на работу генетического алгоритма

В коде реализовано 6 ползунков, которые позволяют пользователю динамически изменять ключевые параметры генетического алгоритма. Разберём, как именно каждый ползунок влияет на процесс обучения модели:

**1. Количество особей (cnt\_ants):**

* **Что делает:** Определяет размер популяции “муравьев”, где каждый муравей представляет собой дерево решений.
* **Влияние:**
  + **Большая популяция:**
    - **(+)** Повышает разнообразие решений, увеличивая шансы нахождения глобального оптимума.
    - **(-)** Требует больше вычислительных ресурсов и времени на обучение.
  + **Маленькая популяция:**
    - **(+)** Быстрее обучается.
    - **(-)** Повышает риск “застревания” в локальном оптимуме.

**2. Количество копируемых особей (copy\_ants):**

* **Что делает:** Определяет, сколько лучших “муравьев” (деревьев) из текущего поколения перейдут в следующее поколение без изменений (элитизм).
* **Влияние:**
  + **Большое количество:**
    - **(+)** Ускоряет сходимость алгоритма, сохраняя лучшие найденные решения.
    - **(-)** Может снизить разнообразие и привести к преждевременной сходимости к локальному оптимуму.
  + **Маленькое количество:**
    - **(+)** Повышает разнообразие, исследуя больше вариантов.
    - **(-)** Может замедлить сходимость.

**3. Вероятность мутации признака (p\_mutate\_ch):**

* **Что делает:** Задаёт вероятность того, что при мутации дерева решений произойдет замена признака, используемого для разделения данных в узле.
* **Влияние:**
  + **Высокая вероятность:**
    - **(+)** Позволяет алгоритму активнее исследовать пространство признаков и находить неочевидные закономерности.
    - **(-)** Может привести к нестабильности обучения и “разрушению” хороших решений.
  + **Низкая вероятность:**
    - **(+)** Способствует сохранению уже найденных удачных комбинаций признаков.
    - **(-)** Может ограничить способность алгоритма находить новые решения.

**4. Максимальное отклонение при мутации (p\_mutate\_op\_rate):**

* **Что делает:** Определяет максимальное значение, на которое может измениться порог сравнения в узле при мутации.
* **Влияние:**
  + **Большое отклонение:**
    - **(+)** Позволяет делать более “резкие” изменения в структуре дерева, что может быть полезно для выхода из локальных оптимумов.
    - **(-)** Уменьшает точность поиска, повышая случайность.
  + **Маленькое отклонение:**
    - **(+)** Делает поиск более точным и сфокусированным.
    - **(-)** Может замедлить сходимость, особенно на ранних этапах обучения.

**5. Множитель штрафа за размер дерева (p\_penalty):**

* **Что делает:** Регулирует силу штрафа, который вычитается из значения функции приспособленности за каждый узел в дереве.
* **Влияние:**
  + **Большой штраф:**
    - **(+)** Способствует построению более простых и интерпретируемых моделей, предотвращая переобучение.
    - **(-)** Может привести к построению слишком простых моделей, которые не смогут уловить сложные зависимости в данных.
  + **Маленький штраф:**
    - **(+)** Позволяет алгоритму строить более сложные модели, потенциально достигая более высокой точности.
    - **(-)** Повышает риск переобучения, когда модель “запоминает” обучающие данные, но плохо работает на новых данных.

**6. Степень влияния глубины дерева на штраф (p\_penalty\_depth):**

* **Что делает:** Определяет, насколько сильно глубина дерева (максимальное количество узлов от корня до листа) влияет на штраф за сложность.
* **Влияние:**
  + **Высокая степень:**
    - **(+)** Способствует построению более “широких” и менее “глубоких” деревьев, что может улучшить интерпретируемость модели.
    - **(-)** Может ограничить возможности алгоритма в поиске сложных закономерностей.
  + **Низкая степень:**
    - **(+)** Даёт алгоритму больше свободы в построении глубоких деревьев, что может быть полезно для решения сложных задач.
    - **(-)** Повышает риск переобучения.

**Важно отметить:**

* Оптимальные значения параметров зависят от конкретной задачи и данных.
* Выбор параметров – это итеративный процесс, который может потребовать экспериментов и анализа результатов.
* Визуализация процесса обучения (история поколений) и результатов (дерево решений) помогает пользователю лучше понять влияние параметров и выбрать наилучшую конфигурацию.

## Инструкция по работе с программой “EDA анализ и построение бинарного дерева решений”

Данная программа позволяет загружать набор данных, проводить его разведочный анализ (EDA) и строить модель классификации в виде бинарного дерева решений с помощью генетического алгоритма.

**Шаг 1: Загрузка данных**

1. Нажмите кнопку “Обзор” в секции “Загрузка данных”.
2. Выберите файл с данными в формате CSV. Данные должны быть организованы в табличном виде, где каждая строка представляет собой объект, а каждый столбец – признак. Один из столбцов должен содержать целевую переменную (метку класса, 0 или 1). Ожидается, что целевая переменная находится в столбце Class и принимает только эти значения – 0 и 1.
3. Дождитесь загрузки данных.

**Шаг 2: EDA анализ**

1. **Просмотр гистограмм:** После загрузки данных автоматически строятся гистограммы для каждого числового признака. Изучите их, чтобы понять распределение значений и выявить потенциально информативные признаки.
2. **Выбор признаков:** Отметьте галочками те признаки, которые вы хотите использовать для построения модели. Вы можете снять галочки с признаков, которые кажутся вам неинформативными или избыточными.
3. **Построение корреляционной матрицы:** Нажмите кнопку “Построить корреляционную матрицу”, чтобы визуализировать взаимосвязи между выбранными признаками. Используйте матрицу для выявления мультиколлинеарности и выбора наиболее информативных признаков.

**Шаг 3: Настройка и запуск генетического алгоритма**

1. **Настройка параметров:** Используйте ползунки в секции “Генетический алгоритм”, чтобы настроить параметры алгоритма:
   * *Количество особей*
   * *Количество копируемых особей*
   * *Вероятность мутации признака*
   * *Максимальное отклонение при мутации*
   * *Множитель штрафа за размер дерева*
   * *Степень влияния глубины дерева на штраф*
2. **Запуск алгоритма:** Нажмите кнопку “Запустить генетический алгоритм”.
3. **Мониторинг обучения:** В процессе работы алгоритма следите за информацией о текущем поколении и значении функции приспособленности лучшего решения. Во время работы алгоритма можно изменять значения
   1. *Вероятность мутации признака*
   2. *Максимальное отклонение при мутации*
   3. *Множитель штрафа за размер дерева*
   4. *Степень влияния глубины дерева на штраф*

**Шаг 4: Анализ результатов**

1. **Остановка алгоритма:** Нажмите кнопку “Остановить генетический алгоритм”, когда посчитаете нужным (например, при достижении желаемого качества решения или по истечении заданного времени).
2. **Просмотр дерева решений:** Визуализируйте лучшее найденное дерево решений в секции “Результирующее дерево решений”. Проанализируйте структуру дерева, чтобы понять, какие признаки и условия используются для классификации.
3. **Просмотр истории обучения:** Нажмите кнопку “Показать историю”, чтобы просмотреть, как менялось дерево решений в процессе обучения. Это поможет вам оценить скорость сходимости алгоритма и влияние параметров на результат.

**Советы:**

* Экспериментируйте с разными наборами признаков и параметрами алгоритма, чтобы найти наилучшее решение для вашей задачи.
* Визуализация – ваш друг! Активно используйте гистограммы, корреляционную матрицу, дерево решений и историю обучения, чтобы лучше понимать данные и результаты работы алгоритма.