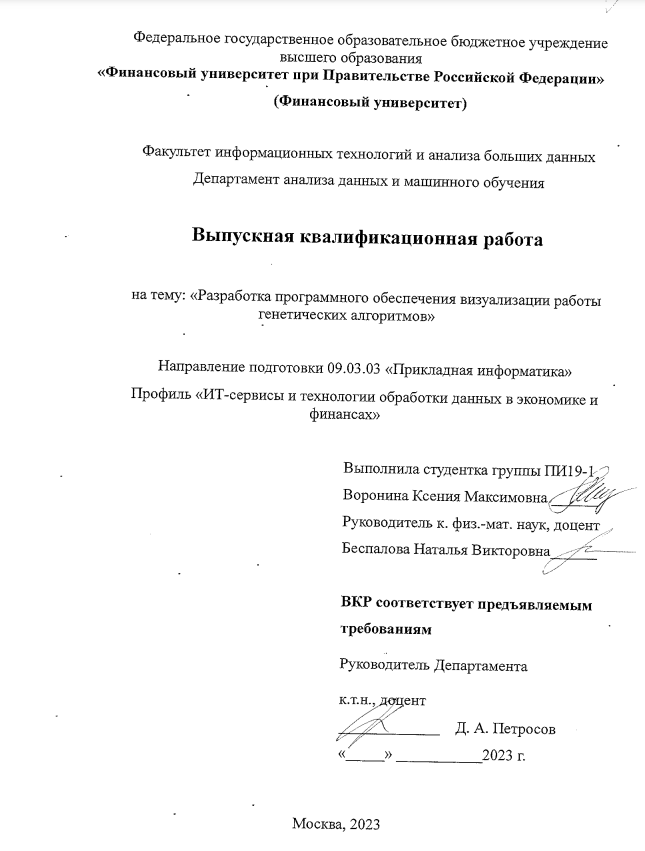
****

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

«Разработка программного обеспечения визуализации работы генетических алгоритмов»

**Оглавление**

[**ВВЕДЕНИЕ** 3](#_Toc135152406)

[**1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ** 5](#_Toc135152407)

[1.1 Генетические алгоритмы 5](#_Toc135152408)

[1.2 Обзор существующих решений визуализации работы генетических алгоритмов 17](#_Toc135152409)

[**2. ОПИСАНИЕ РАЗРАБАТЫВАЕМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** 22](#_Toc135152410)

[2.1 Функциональные возможности программного обеспечения 22](#_Toc135152411)

[2.2 Функционал подсистем и варианты использования. 24](#_Toc135152412)

[2.3 Выбор средств разработки 28](#_Toc135152413)

[2.4 Описание математического аппарата разрабатываемого ПО 31](#_Toc135152414)

[2.5 Описание и предобработка датасета кредитного скоринга. 34](#_Toc135152415)

[**3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ВИЗУАЛИЗАЦИИ РАБОТЫ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ** 47](#_Toc135152416)

[3.1 Логическая структура серверной части программного обеспечения 47](#_Toc135152417)

[3.2 Логическая структура клиентской части программного обеспечения 51](#_Toc135152418)

[3.3 Описание графического интерфейса 55](#_Toc135152419)

[3.4 Результаты обучения скоринг-модели 57](#_Toc135152420)

[**Заключение** 69](#_Toc135152421)

[**Список литературы** 70](#_Toc135152422)

[**Приложение 1. Файл js.js** 73](#_Toc135152423)

[**Приложение 2. Файл server.py** 78](#_Toc135152424)

[**Приложение 3. Файл alg.py** 79](#_Toc135152425)

# **ВВЕДЕНИЕ**

Генетические алгоритмы – это поисковые алгоритмы, которые берут своё начало в теории эволюции Дарвина и на сегодняшний день являются одним из механизмов решения задач оптимизации и моделирования.

Основываясь на процессах естественного отбора и воспроизводства, такие алгоритмы способны находить быстрые и качественные решения задач поиска, обучения и оптимизации, когда традиционные методы не могут предоставить адекватные результаты за приемлемое время.

В наше время генетические алгоритмы доказывают свою конкурентоспособность в решении сложных оптимизационных и поисковых задач. Зачастую в практических приложениях математические модели имеют сложную структуру, поэтому применять традиционные методы типа ветвей и границ, динамического или линейного программирования очень затруднено. Генетические алгоритмы позволяют решать задачи классификации, поиска оптимальных вариантов, прогнозирования и незаменимы в случаях, когда решение задачи основано на интуиции или опыте, а не на строгом ее описании. Таким образом, актуальность выбранной темы объясняется тем, что генетические алгоритмы в некоторых задачах предпочтительнее, чем классические, а порой и незаменимы.

Принцип действия генетических алгоритмов проще понять путем рассмотрения и получения промежуточных результатов вычислений, а также имея возможность анализировать переходы между шагами алгоритма. Поэтому в рамках данной работы было принято решение разработать программное обеспечение визуализации работы генетических алгоритмов для повышения наглядности процессов поиска и оптимизации в решении некоторых задач.

Целью выпускной квалификационной работы является разработка программного обеспечения визуализации работы генетических алгоритмов.

Для достижения данной цели необходимо:

* Понять принцип работы генетических алгоритмов;
* Рассмотреть существующие решения в данной области;
* Понять основную логику работы алгоритма на примере нескольких задач поиска и оптимизации;
* Решить задачу с наглядным представлением поверхности исследуемой функции и решений, найденных на текущей итерации генетическим алгоритмом;
* С помощью языков HTML, CSS, JavaScript и Python разработать web-приложение, отображающее работу генетического алгоритма в решаемой задаче.

# **1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **1.1 Генетические алгоритмы**

В 1859 году английский натуралист и путешественник Чарльз Дарвин опубликовал свой знаменитый труд «Происхождение видов», в котором сформулировал революционную для того времени мысль, что изменение всего живого на Земле происходит путем естественного отбора. Биологическая эволюция занимает сотни миллионов лет, поэтому мы не можем увидеть на протяжении своей жизни значимых эволюционных изменений живых существ, а следовательно, и полностью доказать справедливость гипотезы Дарвина. Или всё же можем?

Потребовалось почти сто лет с момента публикации «Происхождения видов», чтобы доказать справедливость его идеи. В 1954 году итальянский математик Нильс Баричелли выполнил симуляцию эволюции на компьютере института перспективных исследований Принстонского университета, тем самым дав импульс для исследований в этой научной области. Апогеем этих научных изысканий стала разработка Джоном Холландом вместе с его студентами и коллегами из университета Мичиган эволюционного алгоритма, который мы называем генетическим алгоритмом.

Генетические алгоритмы – это алгоритмы поиска, идеи работы которых основаны на естественном отборе и генетике [1, 2]. Генетические алгоритмы работают, реализуя простой вариант эволюции Дарвина, принципы которой приведены ниже:

* Изменчивость. Свойство всех организмов приобретать новые признаки в процессе развития.
* Наследственность. Свойство всех организмов передавать особенности строения и жизнедеятельности из поколения в поколение.
* Естественный отбор. Процесс сохранения и размножения особей наследственными чертами, полезными в существующих условиях.

Эволюция сохраняет популяцию особей, которые отличаются друг от друга. Те, кто лучше приспособлен к окружающей среде, имеют больше шансов на выживание, размножение и передачу своих признаков следующему поколению. Так популяция от поколения к поколению становится все более приспособленной к окружающей среде и встающим на ее пути трудностям.

Важным механизмом эволюции является скрещивание, или рекомбинация, – когда потомок приобретает комбинацию признаков своих родителей. Скрещивание помогает поддерживать разнообразие популяции и со временем закреплять лучшие признаки. Кроме того, важную роль в эволюции играют мутации – случайные вариации признаков, – поскольку они вносят изменения, благодаря которым популяция время от времени совершает скачок в развитии.

Цель генетических алгоритмов – найти оптимальное решение некоторой задачи. Если дарвиновская эволюция развивает популяцию отдельных особей, то генетические алгоритмы развивают популяцию потенциальных решений данной задачи, называемых индивидуумами. Эти решения итеративно оцениваются и используются для создания нового поколения решений. Те, что лучше проявили себя при решении задачи, имеют больше шансов пройти отбор и передать свои качества следующему поколению. Так постепенно потенциальные решения совершенствуются в решении поставленной задачи [3, 4].

Основные компоненты генетических алгоритмов следующие:

* Генотип;

В природе генотип – это совокупность генов, которые сгруппированы в хромосомы, которые несут комбинацию генов родителей. В случае генетических алгоритмов у каждого индивидуума есть хромосома, представляющая собой двоичную строку, где каждый бит есть один ген.

* Популяция;

Генетический алгоритм хранит набор возможных решений поставленной задачи. Этот набор и называется популяцией индивидуумов, его можно рассматривать как коллекцию хромосом.

На рисунке 1 изображена популяция индивидуумов, представленная двоично-кодированными хромосомами:

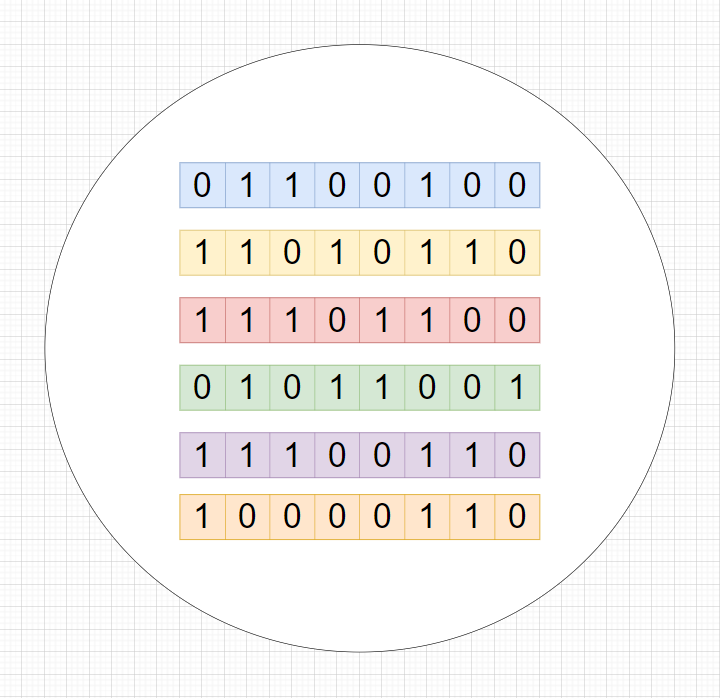


Рисунок 1 Популяция индивидуумов

* Функция приспособленности (фитнес-функция);

Функция приспособленности оценивает на каждой итерации индивидуумов, чтобы отобрать тех, которые дают наилучшие решения и будут представлены в следующем поколении. Это функция и есть задача, которую мы пытаемся решить или оптимизировать.

* Отбор (селекция);

После вычисления приспособленности индивидуумов начинается процесс отбора, с помощью которого определяется, какие индивидуумы будут представлены в следующем поколении для воспроизводства. Все просто – у кого приспособленность выше, те и будут отобраны с наибольшей вероятностью.

* Скрещивание;

Скрещивание (или рекомбинация) – процесс, при котором меняются части хромосом родителей для создания пары новых индивидуумов (потомков).

На рисунке 2 изображен процесс одноточечного скрещивания двух двоично-кодированных хромосом:

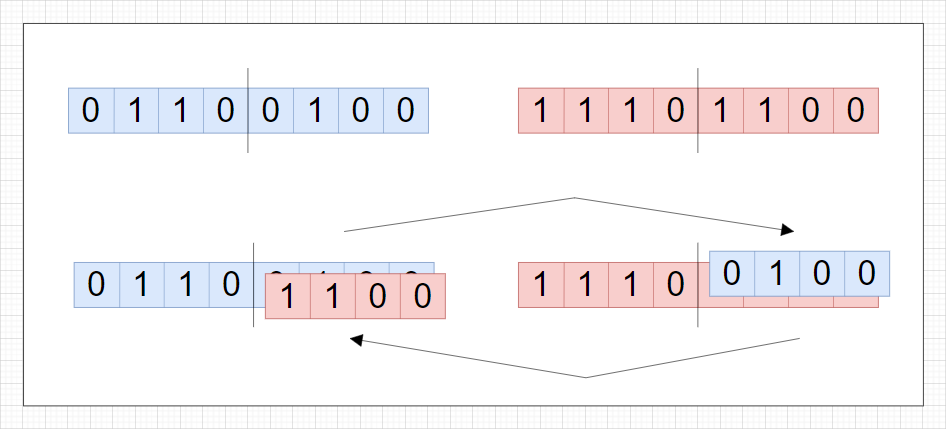


Рисунок 2 Процесс одноточечного скрещивания двух двоично-кодированных хромосом

* Мутация.

Чтобы стимулировать поиск в нерассмотренных областях множества решений, необходимо случайным образом обновлять популяцию. Это можно сделать, внеся в хромосомы новые сочетания генов. Такие изменения называю мутациями, и они могут проявляться как случайное изменение гена.

На рисунке 3 изображено применение оператора мутации к двоично-кодированной хромосоме:

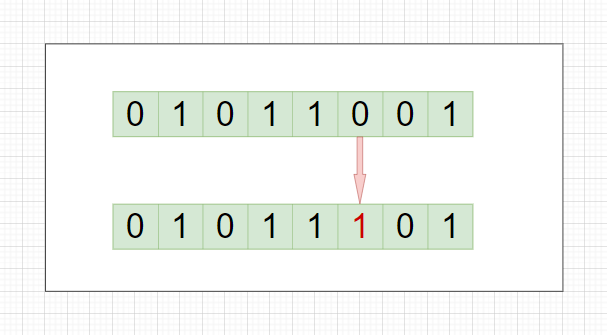


Рисунок 3 Применение оператора мутации к двоично-кодированной хромосоме

Важно понимать отличия генетических алгоритмов от традиционных способов поиска и оптимизации. Первым из них является сама популяция как основа алгоритма: цель ГА – найти совокупность потенциальных решений, а не единственное решение, поэтому на каждой итерации алгоритм хранит множество возможных решений. Например, известный и популярный алгоритм градиентного спуска хранит одно решение, которое итеративно улучшается. Вторым отличием ГА от традиционных алгоритмов является генетическое представление решений, т.е. кодирование потенциальных решений в, так называемые, хромосомы. Это является своеобразным побочным эффектом, который отделяет поиск решений от исходной предметной области. Генетические алгоритмы не пытаются интерпретировать хромосомы. Третье отличие – использование функции приспособленности. Генетические алгоритмы анализируют только значение, которое возвращает эта функция. Нет необходимости работать с производными, что позволяет использовать функции, которые трудно продифференцировать. Последним важным отличием является вероятностное поведение генетических алгоритмов: при переходе к новому поколению применяются вероятностные правила. Например, вероятность попадание индивидуума в следующее поколение тем выше, чем больше значение фитнес-функции, но это не значит, что слабо приспособленные индивидуумы не могут быть отобраны (вероятность этого ниже, но она есть). Также случайность присутствует в процессах мутации и скрещивания.

Рассмотренные особенности генетических алгоритмов помогают определить их преимущества перед традиционными алгоритмами [4, 5]:

* Способность глобальной оптимизации;

Существуют задачи оптимизации, имеющие точки локального максимума и минимума. Эти точки необязательно представляют наилучшие решения в глобальном плане (рис. 4). Традиционные алгоритмы оптимизации часто склонны останавливать поиск решений в локальных точках в то время, как генетические алгоритмы, используя множество потенциальных решений, процессы скрещивания и мутации, менее подвержены этой проблеме.

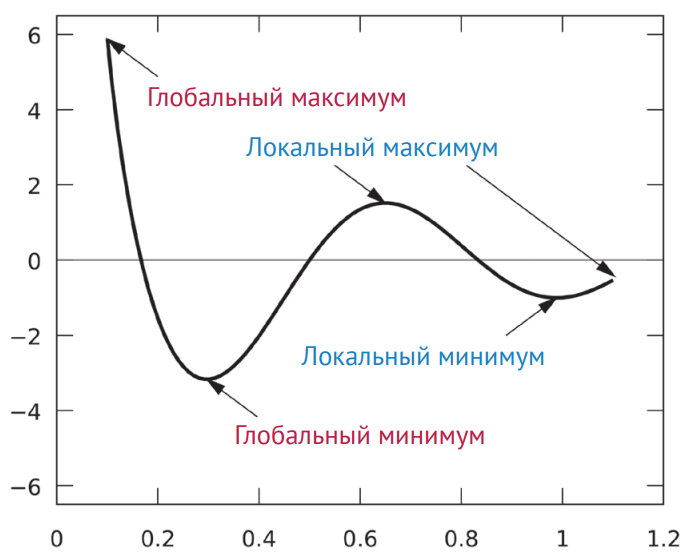


Рисунок 4 Локальные и глобальные макс/мин функции

* Применимость к задачам со сложным математическим представлением и к задачам, которые не имеют математического представления;

Генетические алгоритмы, как уже было сказано ранее, используют только значение функции приспособленности, другие свойства (например, производные) несущественны, что позволяем применять ГА к задачам со сложным математическим представлением, а также к задачам с большим числом параметров. Генетические алгоритмы можно применять к задачам, которые не имеют математического представления (например, когда оценка приспособленности формируется на мнении человека) [6].

* Устойчивость к шуму;

Благодаря повторяющимся операциям сборки и оценивании индивидуумов, генетические алгоритмы устойчивы к шуму, когда при близких истинных значениях входных параметров результаты их измерений могут довольно сильно различаться.

* Поддержка распределенной обработки;

Фитнес-функция вычисляется независимо для каждого индивидуума, что означает возможность обрабатывать всех индивидуумов одновременно. Операции отбора, мутации и скрещивания также могут выполняться одновременно для пар индивидуумов. Это позволяет распараллеливать обработку.

* Пригодность к непрерывному обучению.

Так как процесс эволюции не прекращается, а окружающие условия могут меняться, большим плюсом генетических алгоритмов является их способность непрерывно работать в этих условиях.

Генетические алгоритмы помимо преимуществ имеют и свои ограничения:

* Специальные определения;

Чтобы применить генетический алгоритм к некоторой задаче, необходимо определить фитнес-функцию, структуру хромосом, операторы отбора, мутации и скрещивания. Иногда это не просто и может занимать много времени. На сегодняшний день многие определения стандартизированы, что упрощает задачу.

* Настройка гиперпараметров;

Поведение генетических алгоритмов контролируется набором гиперпараметров, например, размером популяции и скоростью мутации. Точных правил для выбора значений гиперпараметров не существует.

* Большой объем счетных операций;

В случаях работы с большими популяциями генетический алгоритм подразумевает большой объем вычислений, что может увеличить время поиска приемлемого решения. Но эту проблему можно сгладить с помощью грамотного выбора гиперпараметров и распределенных вычислений.

* Опасность преждевременной сходимости;

Генетический алгоритм, хоть и менее склонен застревать в локальных максимумах, всё же может не найти глобальный максимум в случае, когда приспособленность одного индивидуума сильно больше, чем у остальной популяции. Такой индивидуум может продублироваться очень много раз, и, кроме него, в популяции никого не останется. Именно поэтому важно поддерживать разнообразие популяции.

* Отсутствие гарантированного решения.

Зная основные компоненты генетических алгоритмов, преимущества и ограничения, следует понять его базовую структуру и познакомиться с различными реализациями процессов отбора, скрещивания и мутации.

Базовая структура генетического алгоритма выглядит следующим образом (рис. 5):

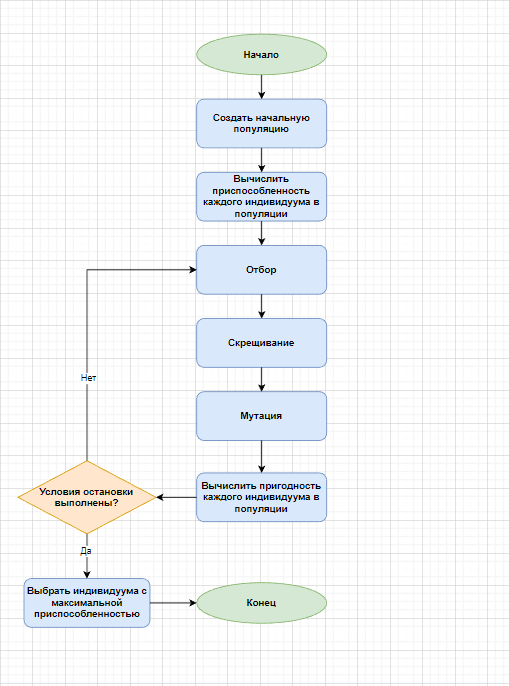


Рисунок 5 Структура генетического алгоритма

Опишем основные шаги работы генетического алгоритма [7-8]:

1. Создание начальной популяции;

Начальная популяция представляет собой случайно выбранные потенциальные решения (индивидуумы), т.е. набор хромосом в соответствующем заданной задаче формате.

1. Вычисление приспособленности;

На этом шаге вычисляется фитнес-функция для каждого индивидуума в начальной популяции. Далее функция приспособленности вычисляет значения после применения операторов отбора, скрещивания и мутации.

1. Отбор;

Процесс отбора отвечает за нахождение лучших индивидуумов в текущей популяции. Существует несколько операторов отбора:

1. Турнирная селекция;

На каждом этапе отбора из популяции выбираются несколько индивидуумов. Тот, у кого приспособленности больше отбирается в следующее поколение.

1. Отбор усечением;

Хромосомы сортируются по значению функции приспособленности, зачем отбирается нужное количество наиболее приспособленных индивидуумов.

1. Отбор пропорционально приспособленности (правило рулетки).

Понять, как работает данный метод отбора очень просто: вероятность попадания индивидуума в следующее поколение прямо пропорциональна его приспособленности.

1. Скрещивание;

Процесс скрещивания создает потомка отобранных индивидуумов. Существует несколько операторов скрещивания:

1. Одноточечное скрещивание;

В данном виде рекомбинации случайным образом выбирается точка скрещивания (точка разреза) – позиция в хромосомах обоих родителей. Гены, расположенные справа от этой точки в обоих случаях, обмениваются друг с другом. В итоге мы получаем двух потомков, которые несут генетическую информацию обоих родителей.

1. Двухточечное и многоточечное скрещивание;

В данном методе рекомбинации случайным образом выбираются по две точки скрещивания в каждой хромосоме. Происходит обмен хромосомами аналогично одноточечному скрещиванию (рис. 6). Этот метод можно реализовать, используя два одноточечных скрещивания с разными точками разреза. Обобщением этого метода является k-точечное скрещивание, где k – целое положительное число.

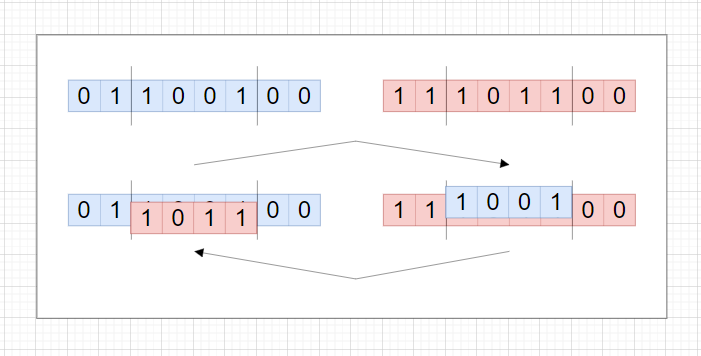


Рисунок 6 Двухточечное скрещивание

1. Равномерное скрещивание;

Для обмена каждый ген обоих родителей определяется путем случайного выбора с равномерным распределением (рис. 7). При таком виде скрещивания, когда выбирается 50 % генов, оба родителя имеют одинаковые шансы повлиять на потомков.

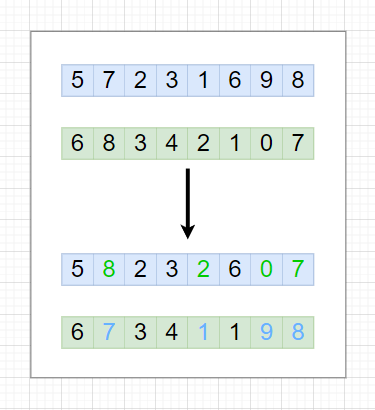


Рисунок 7 Равномерное скрещивание

1. Мутации;

Процесс мутации создает случайные изменения в генах нового индивидуума. Стоит отметить, что это процесс вероятностный и обычно вероятность мутации мала. Примеры операторов мутации:

1. Инвертирование бита;

Данный способ мутации предполагает инвертацию случайно выбранного гена его двоичным дополнением (рис. 8).

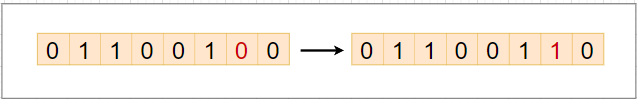


Рисунок 8 Инвертирование бита

1. Мутация обменом;

При применении этого метода случайно выбираются два гена и меняются местами их значения (рис. 9).

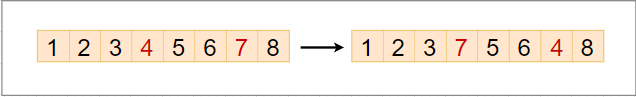


Рисунок 9 Мутация обменом

1. Мутация обращением;

При использовании данного метода мутации выбирается последовательность генов, после чего порядок генов в ней меняется на противоположный (рис. 10).

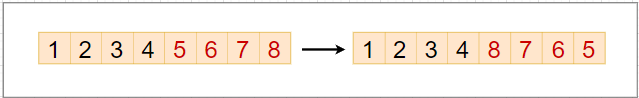


Рисунок 10 Мутация обращением

1. Мутация перетасовкой.

Данный вид мутации предполагает изменение случайным образом порядка генов в случайно выбранной последовательности генов (рис. 11).

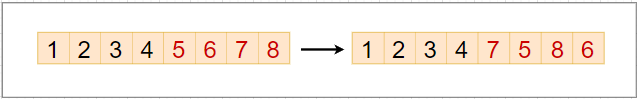


Рисунок 11 Мутация перетасовкой

1. Проверка условий остановки.

Два самых распространенных условия остановки: достижение максимального количества поколений и отсутствие заметных улучшений на нескольких последних поколениях.

## **1.2 Обзор существующих решений визуализации работы генетических алгоритмов**

На сегодняшний день существует мало web-приложений для визуализации генетических алгоритмов. Наиболее интересное из них – *«HTML5 Genetic Algorithm 2D Car Thingy»* [10]. Данная программа с помощью генетического алгоритма превращается случайные двухколесные фигуры в автомобили (рис. 12).

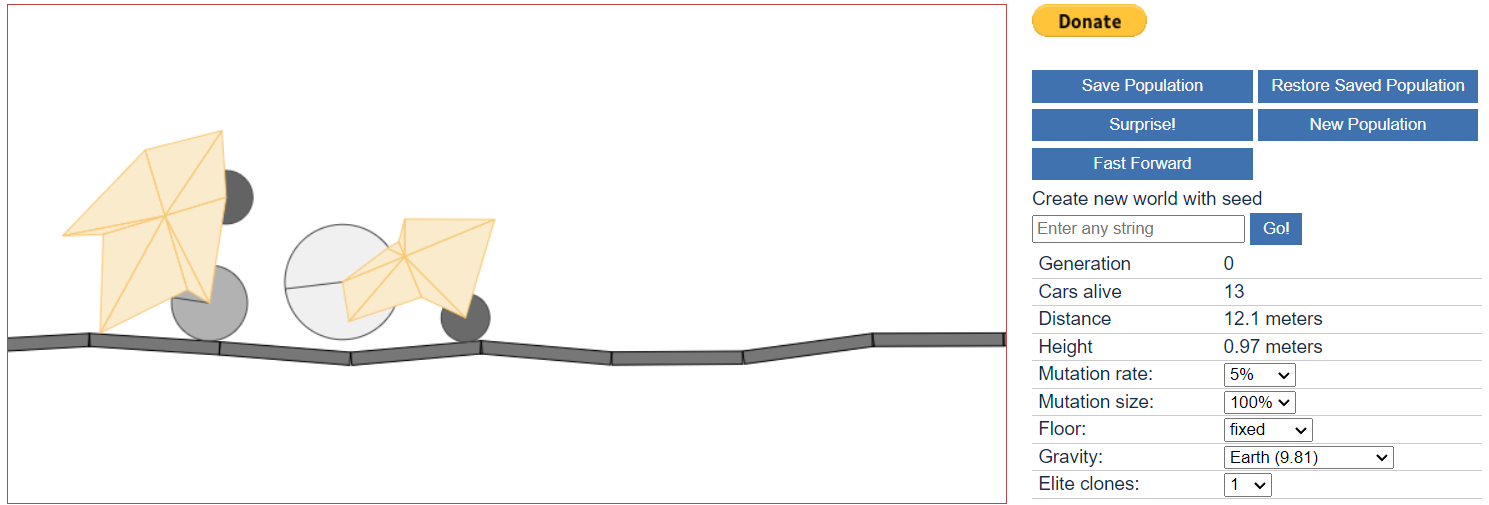


Рисунок 12 «HTML5 Genetic Algorithm 2D Car Thingy»

В приложении реализованы функции сохранения текущей популяции, восстановления ранее сохраненной популяции, отключение рисования и ускорение симуляции (это помогает сократить время работы оптимизатора), а также перезапуск всей популяции автомобилей (рис. 13).

Приложение в реальном времени обновляет данные о номере популяции, количестве «выживших» индивидуумов, наилучшей дистанции. Можно задать вероятность мутации каждого гена у каждого индивидуума и диапазон мутации, выбрать коэффициент свободного падения и изменчивость трассы.

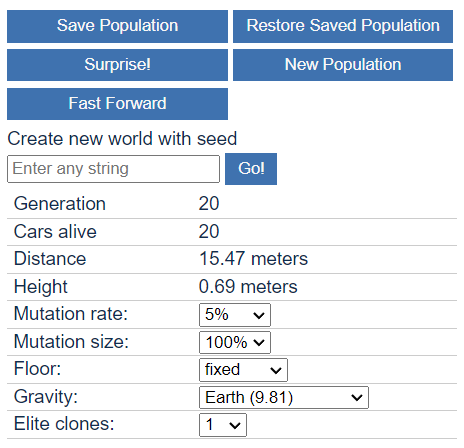


Рисунок 13 Функции приложения «HTML5 Genetic Algorithm 2D Car Thingy»

Помимо этого, на графиках мы можем увидеть по каждой популяции лучшее значение функции приспособленности (красная линия), среднее значение функции приспособленности для десяти лучших индивидуумов (зеленая линия), среднее значение функции приспособленности для всех индивидуумов в популяции (рис. 14).

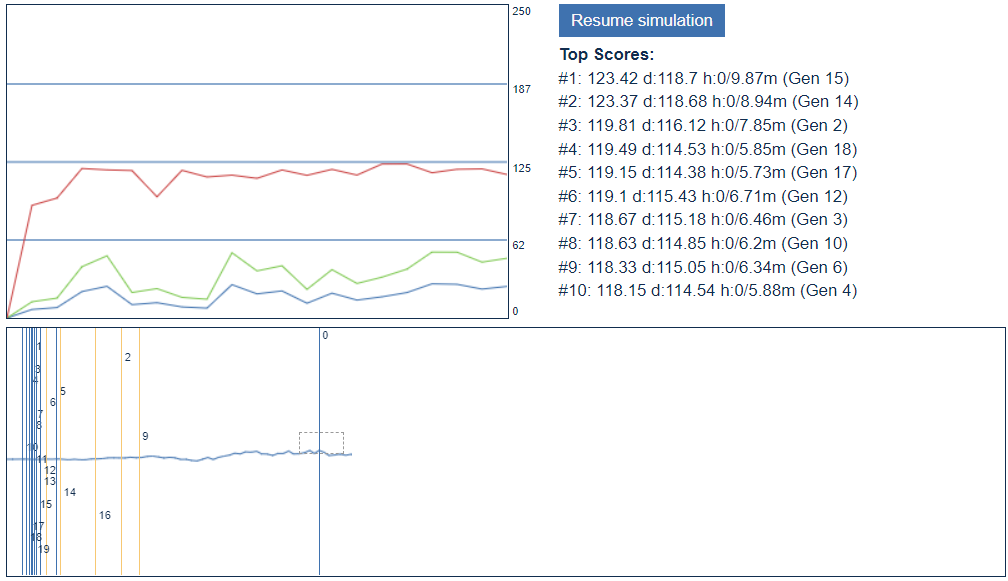


Рисунок 14 Графики работы алгоритма

Стоит сказать несколько слов о desktop-приложениях для визуализации работы генетических алгоритмов [11]. Наиболее известное приложение – *«**Cyber Biology»* (рис. 15).

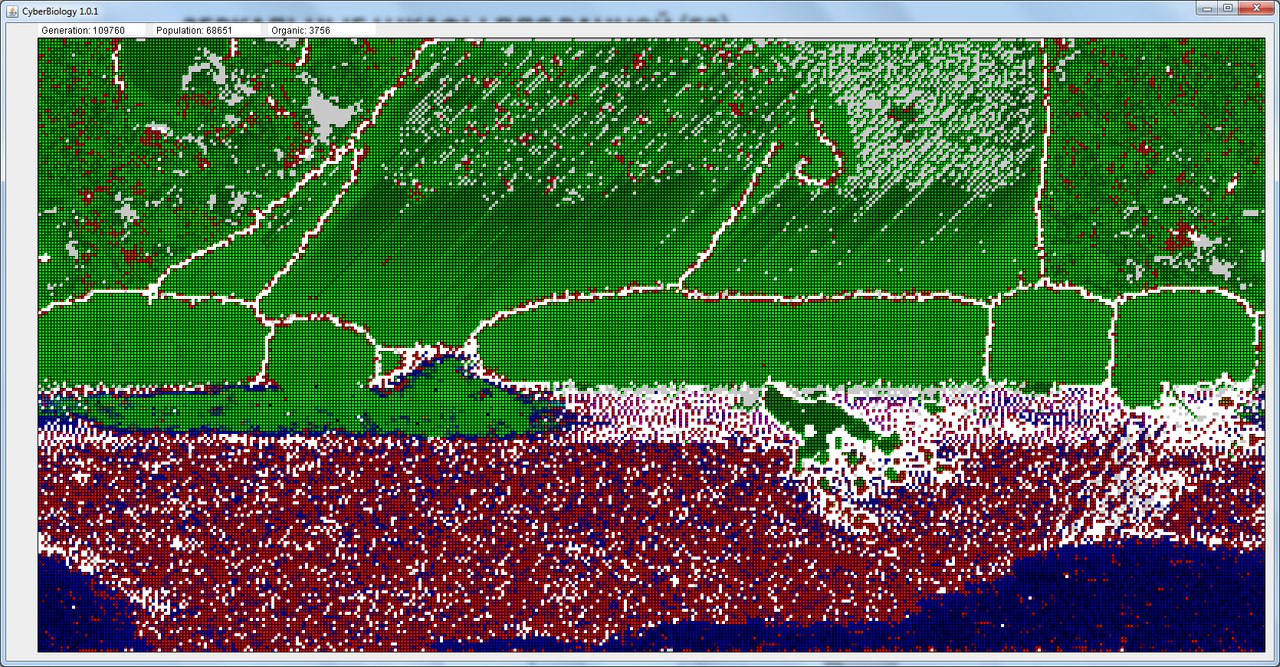


Рисунок 15 Приложение «Cyber Biology»

Это компьютерная среда, позволяющая эмулировать «искусственную жизнь». В приложении создается виртуальная прямоугольная среда, в которой действуют определенные правила, каждый организм в популяции может погибнуть, выжить и дать потомство, которое будет отличаться от него. После чего по принципам естественного отбора происходит развитие данной среды. Поле приложения является моделью водоема в вертикальном разрезе, то есть, чем выше находится организм, тем больше энергии он может получить от солнца. Также в нижней части поля есть минералы, которые также дают энергию. Особи накапливают энергию, а затем, накопив достаточное количество, от них отпочковывается их клон. Клон, благодаря мутации, незначительно изменяет свой геном. Также организмы не умирают от старости, но могут погибнуть, если запасы энергии достигнут нуля или организм будет поглощен другой особью.

Ещё одним интересным приложением является программная среда ***«****Genetic Algorithm Walkers****»*** (рис. 16), в которой эмулируются ходящие манекены.

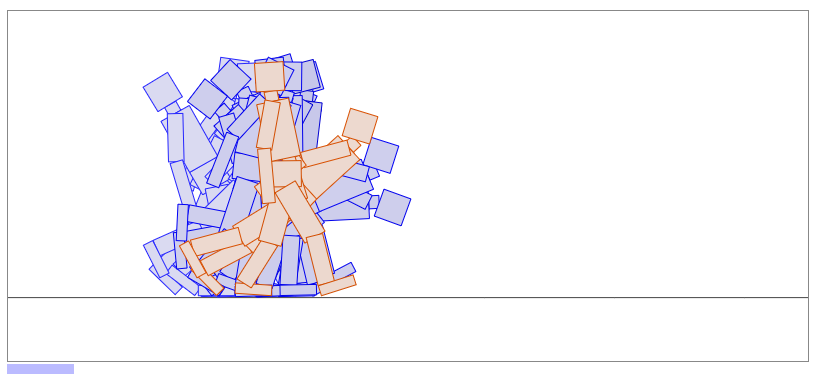


Рисунок 16 Программная среда «Genetic Algorithm Walkers»

В роли популяции здесь выступает набор манекенов с разными параметрами, которые пытаются пройти наиболее длинный путь. Функция приспособленности рассчитывается на основе того, насколько высоко находится голова манекена по отношению к его ногам. Также к этому добавляется каждый правильный шаг, совершаемый манекеном. К концу эволюции получается манекен, способный пройти путь определенной длины.

На мой взгляд, наиболее полным по предоставляемым возможностям (например, визуализации вычисления фитнес-функции, индивидуумов в популяции, процессов мутации и скрещивания) является *«HTML5 Genetic Algorithm 2D Car Thingy».* Существующие решения не соответствуют всем желаемым требованиям, поэтому разработка нового программного обеспечения визуализации работы генетических алгоритмов будет интересной и актуальной задачей.

# **2. ОПИСАНИЕ РАЗРАБАТЫВАЕМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

В данной работе будет решаться задача кредитного скоринга. С помощью машинного обучение необходимо построить скоринг-модель для клиентов банка, которая предсказывала бы вероятность дефолта клиента. Каждая модель машинного обучения имеет свои гиперпараметры, которые влияют на скорость и качество обучения. С помощью генетического алгоритма необходимо найти оптимальные гиперпараметры модели, и именно этот процесс нахождения гиперпараметров будет наглядно продемонстрирован [12-13].

Для достижения цели работы разрабатываемое программное обеспечение должно включать в себя следующие функциональные возможности:

* Выбор настроек генетического алгоритма;
* Пошаговая демонстрация работы алгоритма (вывод индивидуумов каждой популяции и отображение лучших и худших);
* Построение графиков работы генетического алгоритма;
* Вывод информации о работе модели кредитного скоринга.

Проектирование и разработка программного обеспечения визуализации работы генетического алгоритма включает в себя следующие этапы:

* Выбор языков программирования;
* Выбор платформ разработки;
* Разработка серверной части программного обеспечения;
* Разработка клиентской части программного обеспечения;
* Написание программного кода.

## **2.1 Функциональные возможности программного обеспечения**

Программное обеспечение визуализации работы генетических алгоритмов представляет собой web-приложение, которые включает в себя форму выбора настроек ГА, прямоугольное поле, на котором располагаются индивидуумы каждого поколения, и прямоугольное поле, на котором отображаются метрики качества обученной модели кредитного скоринга и графики работы ГА.

Пользователю предлагается выбор следующих настроек работы генетического алгоритма:

1. Модель машинного обучения;
2. Оператор отбора;
3. Оператор скрещивания;
4. Вероятность скрещивания (от 0 до 1 с шагом 0,05);
5. Оператор мутации;
6. Вероятность мутации (от 0 до 1 с шагом 0,05);
7. Максимальное количество популяций;
8. Максимальное количество индивидуумов в популяции.

Программное обеспечение поддерживает различные виды селекций:

1. Турнирный отбор;
2. Пропорциональный отбор (правило рулетки);
3. Случайный отбор;
4. Отбор усечением.

Следующие виды скрещивания:

1. Одноточечное скрещивание;
2. Двухточечное скрещивание;
3. Равномерное скрещивание.

А также несколько видов мутации:

1. Новое значение в одном из генов;
2. Новые значения во всех генах;
3. Мутация численных генов с применением random.gauss().

Каждый индивидуум имеет гены, которые являются гиперпараметрами выбранной модели машинного обучения. Например, индивидуум логистической регрессии LogisticRegression () будет иметь следующие гены:

* Penalty;
* C;
* Max\_iter.

Фитнесс-функция равна метрике качества обученной модели кредитного скоринга. Модель обучается с наилучшими гиперпараметрами, т.е. с наилучшим индивидуумом ГА.

## **2.2 Функционал подсистем и варианты использования.**

Для графического моделирования функциональной модели, отображающей структуру системы и ее функции, а также объекты и потоки информации была построены диаграмма IDEF0 (рис. 17, рис. 18) [15].

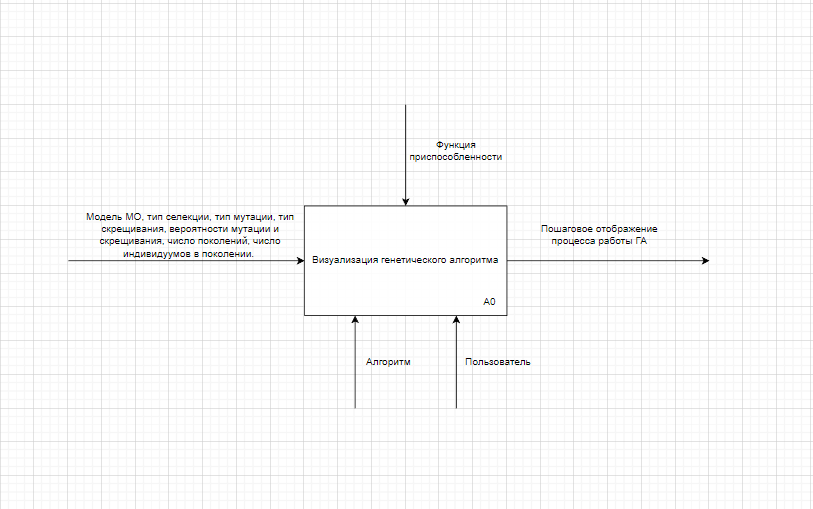


Рисунок 17 Диаграмма IDEF0

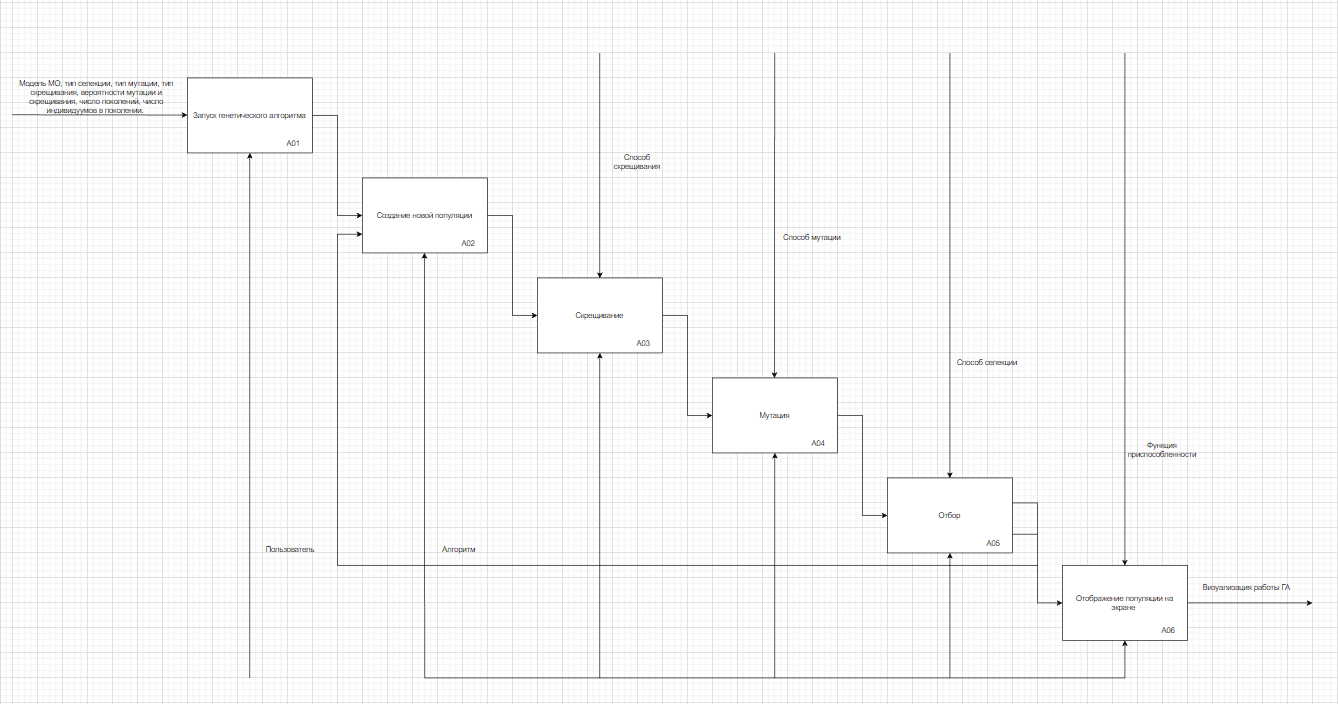


Рисунок 18 Диаграмма уровня A0 IDEF0

Разрабатываемое web-приложение включает в себя клиентскую и серверную части. Функционал каждой подсистемы программного обеспечения описан ниже.

Функционал клиентской части приложения:

1. Выбор настроек алгоритма;
2. Запуск генетического алгоритма;
3. Пошаговое отображение популяций генетического алгоритма;
4. Вывод статистики работы генетического алгоритма в виде графиков;
5. Вывод результатов обучения модели кредитного скоринга;
6. Очистка полей отображения результатов работы генетического алгоритма при повторном запуске;
7. Остановка генетического алгоритма.

Функционал серверной части приложения:

1. Загрузка датасета кредитного скоринга;
2. Создание генетического алгоритма с выбранными пользователем параметрами;
3. Выполнение селекции;
4. Выполнение скрещивания;
5. Выполнение мутации;
6. Вычисление функции приспособленности;
7. Обучение модели кредитного скоринга с наилучшими гиперпараметрами, подобранными генетическим алгоритмом;
8. Построение статистики работы генетического алгоритма;
9. Вычисление метрик качества модели машинного обучения.

Для работы с web-приложением необходимо разработать следующие формы:

1. Форма для работы с алгоритмом:

* Форма для выбора параметров алгоритма;
* Форма для запуска и остановки алгоритма.

1. Форма визуализации алгоритма:

* Форма визуализации популяций;
* Форма отображения генов;
* Форма вывода результатов работы модели кредитного скоринга.

Весь функционал, доступный пользователю, представлен на диаграмме вариантов использования, которая была построена с помощью программного обеспечения для рисования графиков Draw.io (рис. 19):

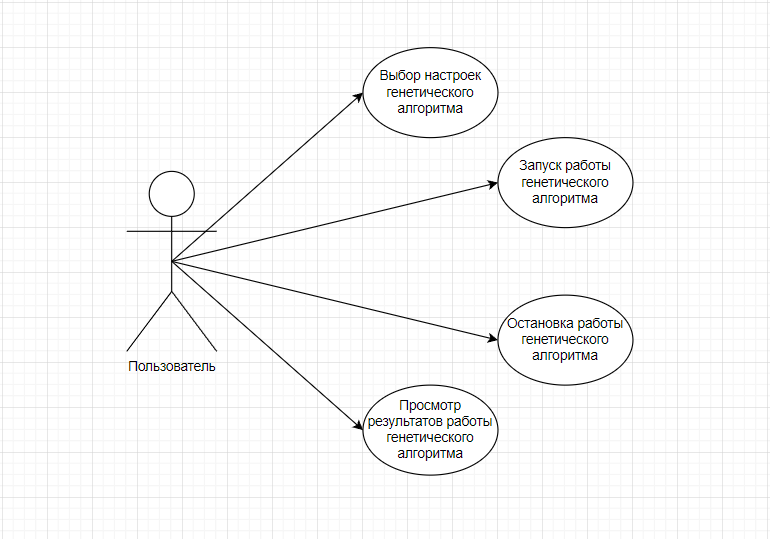


Рисунок 19 Диаграмма вариантов использования

Пользователь может выбирать настройки генетического алгоритма или оставлять их по умолчанию, запускать и останавливать генетический алгоритм, просматривать результаты работы ГА в виде пошагового отображения популяций индивидуумов и результатов обучения модели.

На рисунке 20 представлена диаграмма последовательности на основе сценария запуска и обработки генетического алгоритма.

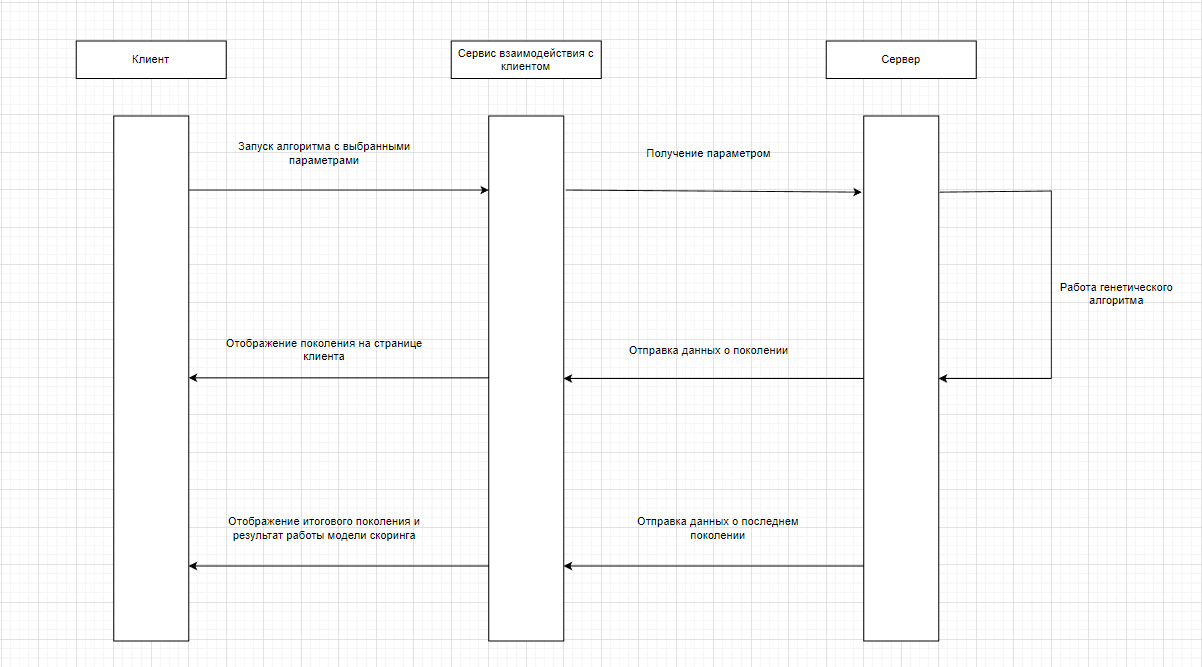


Рисунок 20 Диаграмма последовательности

## **2.3 Выбор средств разработки**

Технологии и средства разработки программного обеспечения выбирались в соответствии со следующими критериями [17-18]:

* Возможность удобной реализации генетического алгоритма и пошаговой визуализации его работы;
* Возможность построения серверной части приложения;
* Возможность построения клиентской части с пошаговой визуализацией работы генетического алгоритма.

Для реализации серверной части программного обеспечения был выбран язык Python. Он является одним из самых популярных в области анализа данных с использованием машинного обучения и искусственного интеллекта. Python имеет множество плюсов: простой и понятный синтаксис; гибкость и мультиплатформенность; наличие большого числа библиотек с уже готовыми программными решениями и конструкциями; большое сообщество программистов и подробная документация.

Для создания серверной части программного обеспечения было принято использовать основные библиотеки:

1. DEAP;

DEAP (Distributed Evolutionary Algorithms in Python) – это библиотека на языке Python для реализации эволюционных алгоритмов. Она предоставляет широкий спектр инструментов для решения задач оптимизации, включая генетические алгоритмы, эволюционные стратегии, генетическое программирование и другие. DEAP была разработана с учетом возможности распараллеливания вычислений, что позволяет использовать ее для решения сложных задач оптимизации. Библиотека также предоставляет графические инструменты для визуализации процесса оптимизации и результатов [30-31].

1. Sklearn;

Scikit-learn (sklearn) – это библиотека на языке Python для машинного обучения и анализа данных. Она предоставляет огромный спектр инструментов для решения задач классификации, регрессии, кластеризации, извлечения признаков и других. Sklearn позволяет интеграцию с другими библиотеками Python, такими как NumPy, SciPy и Pandas [28].

1. Matplotlib.

Matplotlib - это библиотека на языке Python для создания графиков и визуализации данных. Она позволяет создавать различные типы графиков, включая линейные, столбчатые, круговые, гистограммы, диаграммы рассеяния и многое другое [29].

Было принято решение, что клиентская часть программного обеспечения будет представлять собой web-приложение, так как оно не требует установки, все обновления происходят на сервере и доставляются пользователям сразу; оно будет доступно из любой точки мира и с любого устройства; на сегодняшний день браузерные приложения становятся популярнее, удобнее и охватывают большую аудиторию. Для реализации клиентской части программного обеспечения были выбраны языки HTML, CSS и язык JavaScript. HTML – это язык гипертекстовой разметки страницы. Он представляет собой стандарт разметки страниц для интернета, браузеры интерпретируют его и отображают уже как сформированный документ. В дополнение к HTML используется CSS – это каскадные таблицы стилей, это язык, который направлен на описание внешнего вида документа. JavaScript – мультипарадигменный язык программирования. Он поддерживает объектно-ориентированный, императивный и функциональный стили, является одним из самых распространенных языков, область его применения обширна и практически безгранична. JavaScript работает во всех браузерах, он очень удобен для создания web-приложений. Именно поэтому данный стек был выбран для создания клиентской части программного обеспечения [20-24].

Взаимодействие между клиентской и серверной частями осуществляется с помощью WebSocket – протокола, который позволяет установить двустороннее клиент-серверное соединение, используя один и тот же TCP-сокет. Это значит, что данные могут передаваться между клиентом и сервером в режиме реального времени, без необходимости постоянно устанавливать новое соединение. Когда клиент отправляет запрос на сервер, используя WebSocket, он отправляет специальный заголовок, который указывает на то, что он хочет установить WebSocket-соединение. Если сервер поддерживает этот протокол, он отправляет ответ с таким же заголовком, и соединение устанавливается. После установки соединения клиент и сервер могут обмениваться данными в режиме реального времени, используя методы send() и onmessage(). Когда клиент отправляет данные на сервер с помощью метода send(), сервер получает эти данные в виде сообщения, которое можно обработать и отправить обратно клиенту с помощью того же метода send(). Когда клиент получает сообщение от сервера, срабатывает метод onmessage(), который позволяет обработать полученные данные [25].

WebSocket является очень удобным протоколом для реализации приложений, где необходимо обмениваться данными в режиме реального времени. Он также обеспечивает более низкую задержку и меньший объем трафика по сравнению с традиционными HTTP-запросами.

В качестве платформ для разработки программного обеспечения были выбраны Visual Studio Code и JupyterNotebook.

Visual Studio Code – текстовый редактор, позволяющий работать с широким набором файлов, а также с разными языками программирования. Позволяет устанавливать различные расширения, такие как отладчик для браузера Google Chrome, набор инструментов Emmet и т.д. Является кроссплатформенным и может быть установлен на Linux, MacOS и Windows.

JupyterNotebook – это веб-приложение, своеобразная среда разработки, для создания вычислительных документов, передачи и запуска кода. JupyterNotebook имеет простой интерфейс, ориентированный на работу с документами. Чаще всего сервис используют для Python, но Jupyter также поддерживает языки Ruby, R, MATLAB и другие [26-27].

## **2.4 Описание математического аппарата разрабатываемого ПО**

Для создания скоринг-модели с помощью библиотек sklearn и xgboost были выбраны три модели:

* LogisticRegression ();

Logistic Regression — это распространенный метод классификации, который относится к линейным классификаторам. Он базируется на вероятностях принадлежности наблюдений к определенному классу, которые моделируются с помощью логистической функции, являющейся кумулятивной функцией распределения.

Логистическая регрессия используется для прогнозирования вероятности возникновения некоторого события по значениям множества признаков. Для этого вводится зависимая переменная �Y, принимающая лишь одно из двух значений — как правило, это числа 0 (событие не произошло) и 1 (событие произошло), и множество независимых переменных (признаки) — X1, X2 … Xn�1,�2,...,��, на основе значений которых требуется вычислить вероятность принятия того или иного значения зависимой переменной. Логистическая функция выглядит следующим образом (рис. 21):

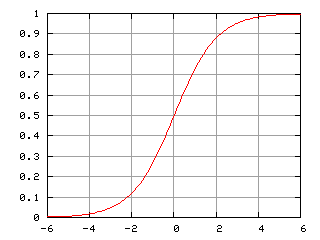


Рисунок 21 Логистическая функция

Логистическая регрессия выполняется быстро и относительно несложно.

* RandomForestClassifier ();

RandomForestClassifier — ансамблевый метод обучения, который создает несколько деревьев решений из выбранного случайным образом подмножества обучающего набора, затем объединяет результаты из разных деревьев решений, чтобы определить окончательный класс наблюдений из тестовой выборки. Цель ансамблевых моделей — объединить прогнозы нескольких базовых моделей, построенных с заданным алгоритмом обучения, чтобы повысить надежность по сравнению с одной оценкой. Предполагается, что результат обучения нескольких моделей более точен, чем результат обучения только одной модели. Существует множество методов ансамбля, которые, как показывает практика, повышают точность при использовании с передовыми алгоритмами машинного обучения.

RandomForestClassifier рассматривается как высокоточный и надежный метод из-за количества деревьев решений, участвующих в процессе (рис. 22).

Алгоритм состоит из четырех этапов:

1. Создание случайных выборок из заданного набора данных;
2. Для каждой выборки построение дерева решений и получение результата предсказания, используя данное дерево;
3. Проведение голосования за каждый полученный прогноз;
4. Выбор предсказания с наибольшим количеством голосов в качестве окончательного результата.

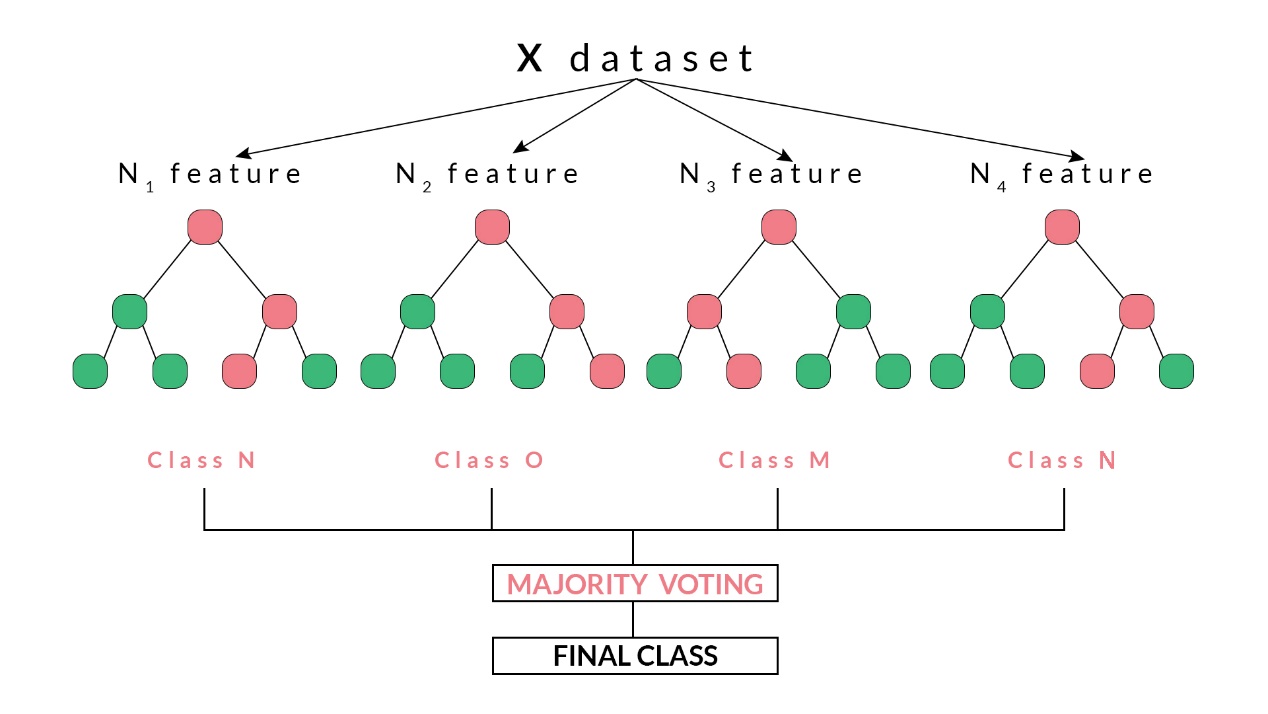


Рисунок 22 Алгоритм случайного леса

* GradientBoostingClassifier ().

GradientBoostingClassifier – это алгоритм машинного обучения, который используется для задач классификации. Он является одним из наиболее эффективных алгоритмов, используемых в области машинного обучения.

Основная идея GradientBoostingClassifier заключается в том, чтобы построить ансамбль слабых моделей (например, деревьев решений), которые будут последовательно улучшать предсказания. Каждое новое дерево строится таким образом, чтобы исправлять ошибки предыдущих деревьев. Таким образом, GradientBoostingClassifier позволяет создать композицию моделей, которая может достичь высокой точности предсказания.

Одним из основных преимуществ GradientBoostingClassifier является его способность работать с большим количеством признаков и обрабатывать нелинейные зависимости между ними. Кроме того, этот алгоритм хорошо работает с выбросами и шумами в данных.

Однако GradientBoostingClassifier имеет и некоторые недостатки. Он может быть чувствителен к переобучению и требует тщательной настройки гиперпараметров. Кроме того, этот алгоритм может быть медленным в обучении и требовать больших объемов данных для достижения хорошей точности.

## **2.5 Описание и предобработка датасета кредитного скоринга.**

Выбранный датасет кредитного скоринга представляет собой набор данных о заемщиках, которые подавали заявки на кредит в банке. Каждая строка в датасете соответствует одному заемщику, а каждый столбец содержит информацию о различных параметрах, таких как возраст, доход, семейное положение, наличие задолженностей и т.д.

Датасет содержит следующие столбцы (рис. 23):

1. BAD:

В этой колонке указывается, был ли кредит возвращен полностью или по нему был допущен дефолт. Если кредит был полностью оплачен, значение в этом столбце будет равно 0, а если по кредиту был допущен дефолт или списание средств, значение будет равно 1.

1. LOAN:

В этом столбце представлена сумма денег, взятых в долг заемщиком.

1. MORTDUE:

В этом столбце представлена сумма задолженности по ипотеке заемщика.

1. VALUE:

В этом столбце представлена оценочная стоимость имущества заемщика.

1. REASON:

В этой колонке указывается причина предоставления кредита, например, консолидация долга, улучшение жилищных условий или другие цели.

1. JOB:

В этом столбце указана профессия или статус занятости заемщика.

1. YOJ:

В этом столбце указано количество лет, в течение которых заемщик работал.

1. DEROG:

В этой колонке указано количество замечаний в кредитном отчете заемщика.

1. DELINQ:

В этом столбце указано, сколько раз заемщик задерживал платеж.

1. CLAGE:

В этом столбце указан возраст самой старой кредитной линии заемщика.

1. NINQ:

В этом столбце указано количество кредитных линий, открытых заемщиком за последние шесть месяцев.

1. CLNO:

В этом столбце представлено общее количество кредитных линий, имеющихся у заемщика.

1. DEBTINC:

В этом столбце представлено отношение долга заемщика к доходу, которое рассчитывается путем деления общей суммы ежемесячных платежей заемщика по долгу на его ежемесячный доход.

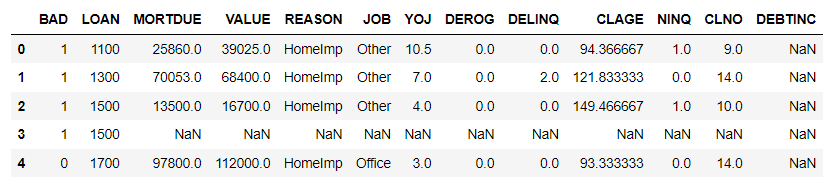


Рисунок 23 Столбцы датасета

Цель этого датасета - определить вероятность того, что заемщик вернет кредит в срок. Для этого используется кредитный скоринг, который вычисляется на основе различных параметров, таких как кредитная история, уровень дохода, возраст и другие. Датасет кредитного скоринга будет использован для обучения моделей машинного обучения, которые могут предсказывать вероятность того, что заемщик вернет кредит в срок. Это помогает банкам принимать более обоснованные решения о выдаче кредитов и снижает риски невозврата долгов [32].

Предобработка данных является важным этапом в машинном обучении, который помогает улучшить качество моделей и снизить вероятность ошибок. Она включает в себя различные процессы, такие как очистка данных, обработка пропущенных значений, нормализация и кодирование категориальных признаков.

Первый шаг в предобработке данных – это очистка данных от выбросов и ошибок. Это может быть выполнено путем удаления некоторых строк или столбцов, которые содержат некорректные значения. Далее, необходимо обработать пропущенные значения, которые могут возникнуть из-за ошибок ввода или отсутствия информации. Это может быть выполнено путем заполнения пропущенных значений средними или медианными значениями.

С помощью функций info() и isnull().sum() библиотеки pandas была получена информация о каждом столбце: тип данных, количество не пустых записей, количество пропущенных значений (рис. 24). Всего в датасете 5960 сточек. Как видно, в нем пропущено много значений, во всех столбцах есть значения Null.

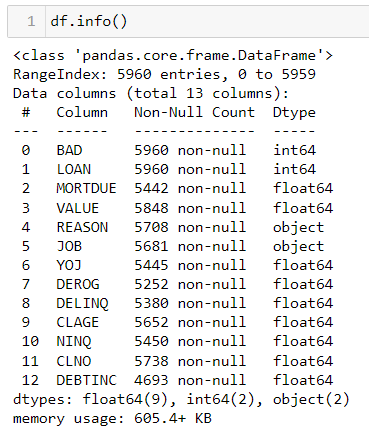
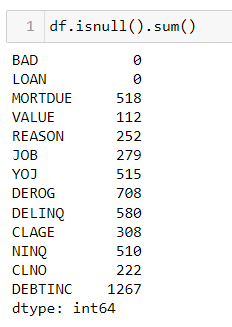


Рисунок 24 Работа функций info() и isnull().sum()

В столбцах VALUE (оценочная стоимость имущества), DEROG (количество замечаний в кредитном отчете заемщика), DELINQ (количество задержек платежей) пропущенные значения были заменены на 0, так как было сделано предположение, что значение Null может означать отсутствие имущества, замечаний и задержек платежей (рис. 25).

В остальных столбцах пропущенные значения были заполнены медианными значения, так как предполагается, что причина предоставления кредита, место работы и стаж не могут отсутствовать или быть равными нулю.

Столбец DEPTINC (отношение долга заемщика к доходу) был удален, так как в нем пропущено много значений, удаление или замена которых может привести к уменьшению качества модели и увеличению вероятности ошибок.

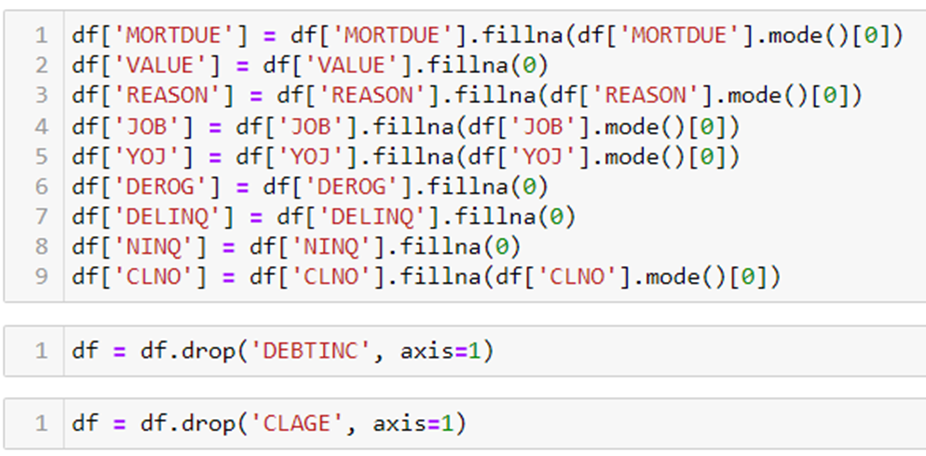


Рисунок 25 Замена пропущенных значений

Далее, необходимо нормализовать данные, чтобы они имели одинаковый масштаб и были более устойчивы к шумам и выбросам. Это может быть выполнено путем стандартизации данных или масштабирования данных до определенного диапазона.

Кроме того, категориальные признаки должны быть закодированы в числовые значения, чтобы модель могла работать с ними. Это может быть выполнено путем преобразования категориальных признаков в бинарные флаги или использованием методов, таких как кодирование порядковых значений или кодирование средним значением.

Для начала определим, какие столбцы являются категориальными, а какие – числовыми. Для этого используем функцию nunique(), которая подсчитывает количество уникальных значений по каждому столбцу. Столбцы REASON и JOB отнесем к категориальным признакам, остальные столбцы будет считать числовыми, так как количество уникальных значений достаточно большое (рис. 26).

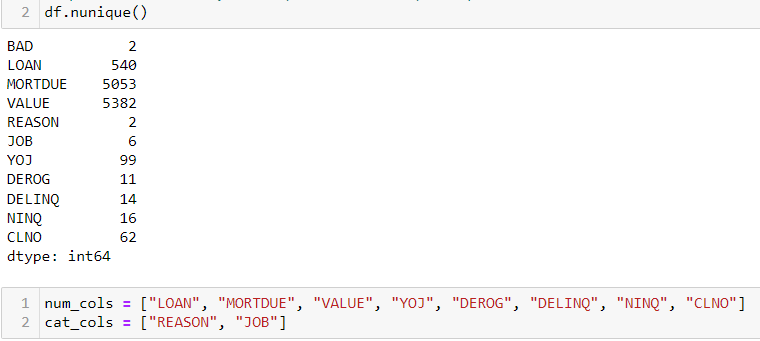


Рисунок 26 Категориальные и численные признаки

Также при обучении моделей будем учитывать, что целевая переменная является несбалансированной (рис. 27). Количество дефолтов по кредитам, очевидно, меньше, чем количество успешно погашенных кредитов.

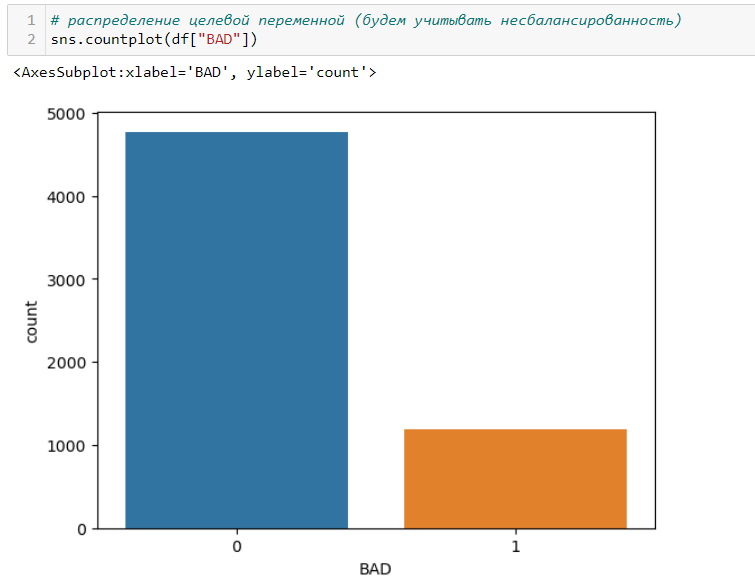


Рисунок 27 Распределение целевой переменной

Далее был проведен анализ числовых признаков. Для каждого столбца были построены гистограммы, которые показывают распределение переменной. На рисунке 28 показано распределение переменной LOAN (сумма денег, взятых в долг заемщиком). А также с помощью функции describe() была получена информация о среднем значении, максимальном и минимальном значении, стандартном отклонении, а также информация о квантилях распределения (рис. 29).

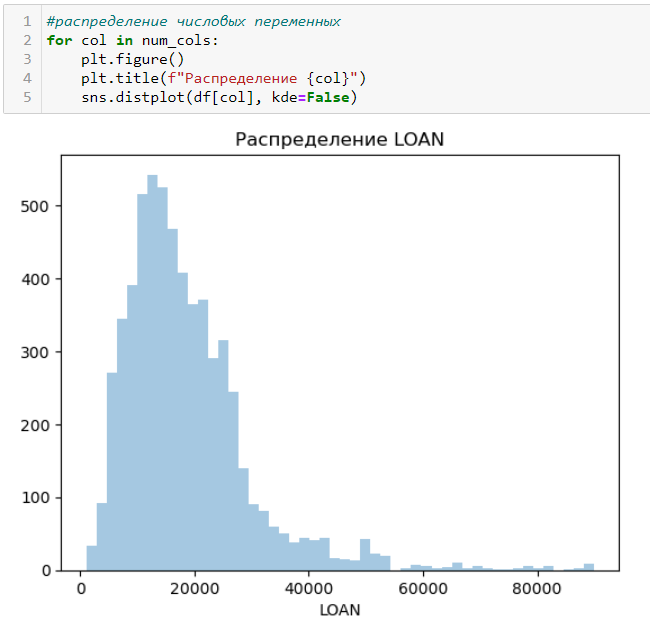


Рисунок 28 Распределение переменной LOAN

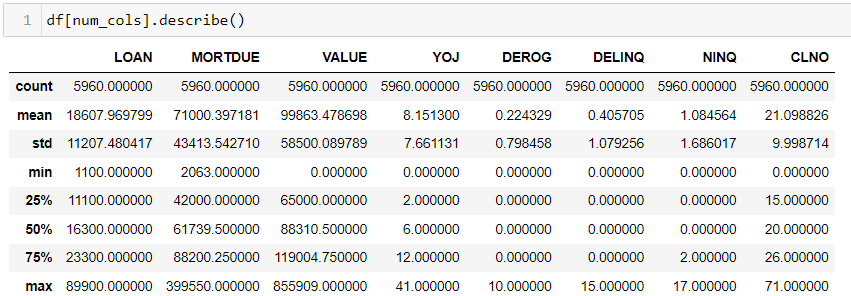


Рисунок 29 Статистическая информация по столбцам

Логарифмирование числовых признаков может быть полезным в моделях машинного обучения, когда распределение данных сильно смещено в одну сторону или имеет большой разброс значений. Логарифмирование может помочь уменьшить разброс значений и сделать распределение более нормальным, что может улучшить производительность модели. Кроме того, логарифмирование может помочь выявить скрытые зависимости между признаками и целевой переменной, которые могут быть неочевидными при использовании исходных значений признаков.

Прологарифмируем числовые переменные, так как все имеют смещение влево или вправо. На рисунке 30 показано распределение переменной LOAN после логарифмирования.

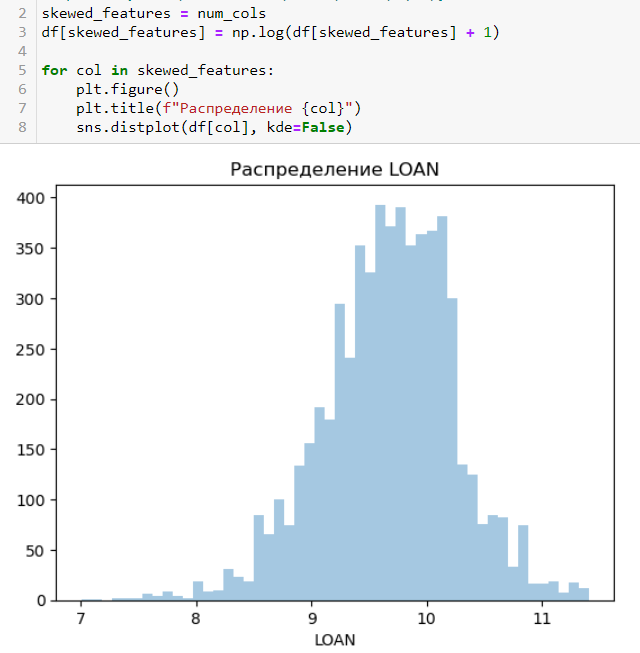


Рисунок 30 Распределение переменной LOAN после логарифмирования

На следующем этапе предобработки была проверена корреляция признаков между собой. Эта проверка важна для избежания мультиколлинеарности – явления, когда два или более признаков сильно коррелируют между собой. Мультиколлинеарности может привести к неустойчивости модели и снижению ее производительности, а также затруднить интерпретацию результатов.

Если два признака сильно коррелируют между собой, то они содержат похожую информацию и могут быть заменены одним признаком или объединены в новый признак. Проверка корреляции признаков позволяет выявить такие зависимости и определить, какие признаки следует использовать в модели.

Признаки в данном датасете слабо коррелируют между собой, поэтому для обучения модели можно оставить все (рис. 31).

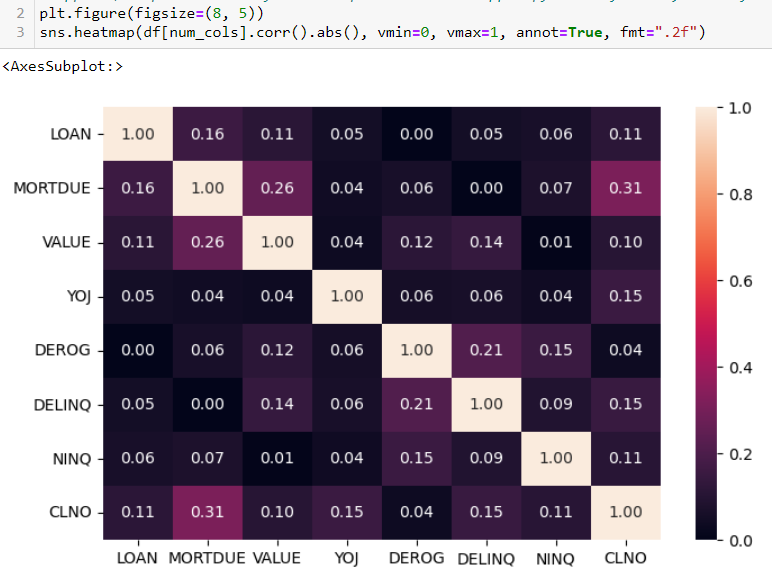


Рисунок 31 Проверка корреляции признаков

На следующем шаге предобработки построим гистограммы распределения по целевой переменной. Это поможет визуально оценить выборку на аномальные значения и признаки, которые не несут информацию для нашего случая. На рисунке 32 показано распределение для первых четырех числовых признаков.

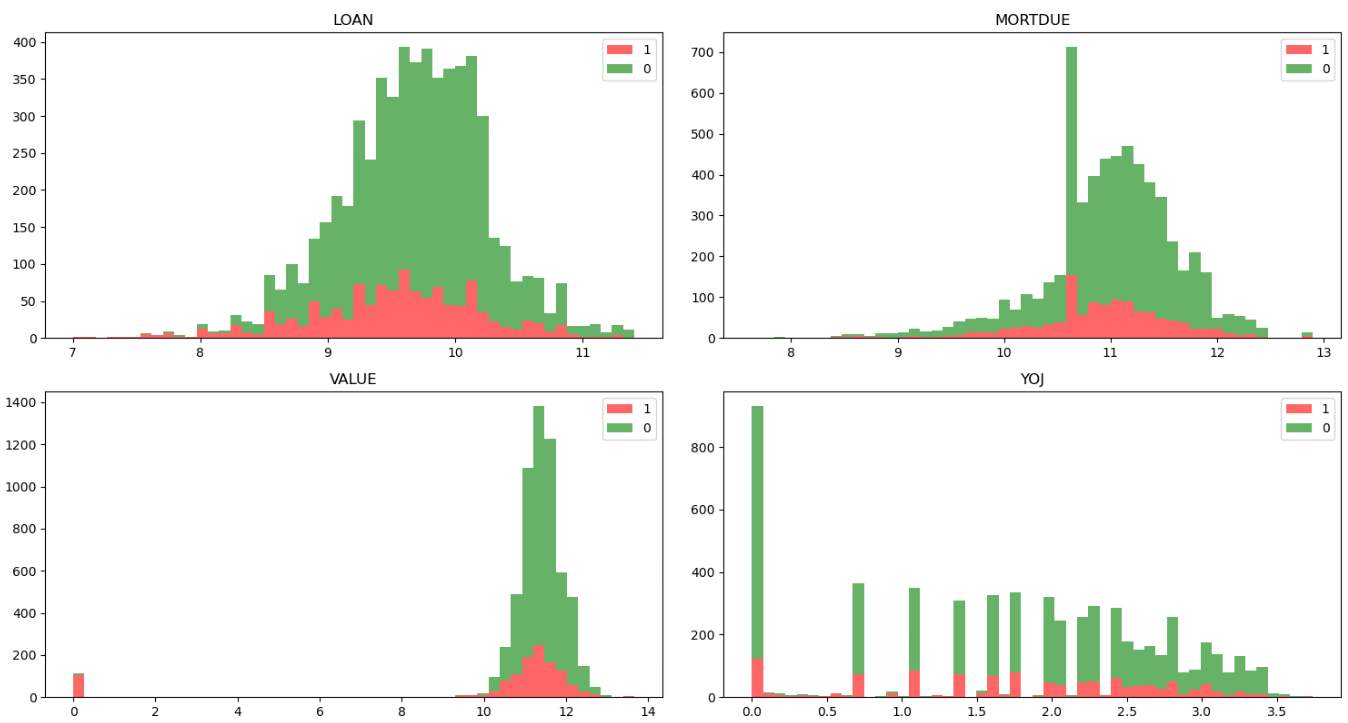


Рисунок 32 Гистограммы распределения по целевой переменной

Опираясь на полученные гистограммы, нельзя точно сказать, какие признаки более значимы, поэтому с помощью дисперсионного анализа определим значимость числовых признаков (рис. 33).

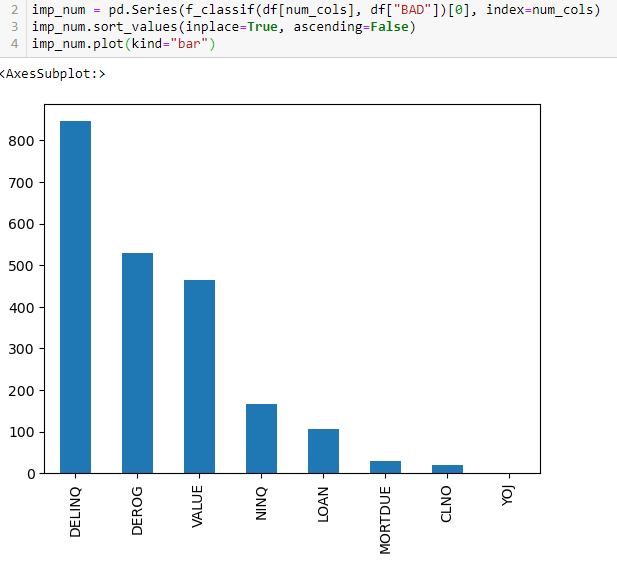


Рисунок 33 Значимость числовых признаков

Наиболее значимыми оказались признаки DELING, DEROG и VALUE, а менее значимым признак YOJ.

На следующем шаге был проведен анализ категориальных признаков (REASON и JOB). На рисунке 34 показано распределение значений по этим признакам.

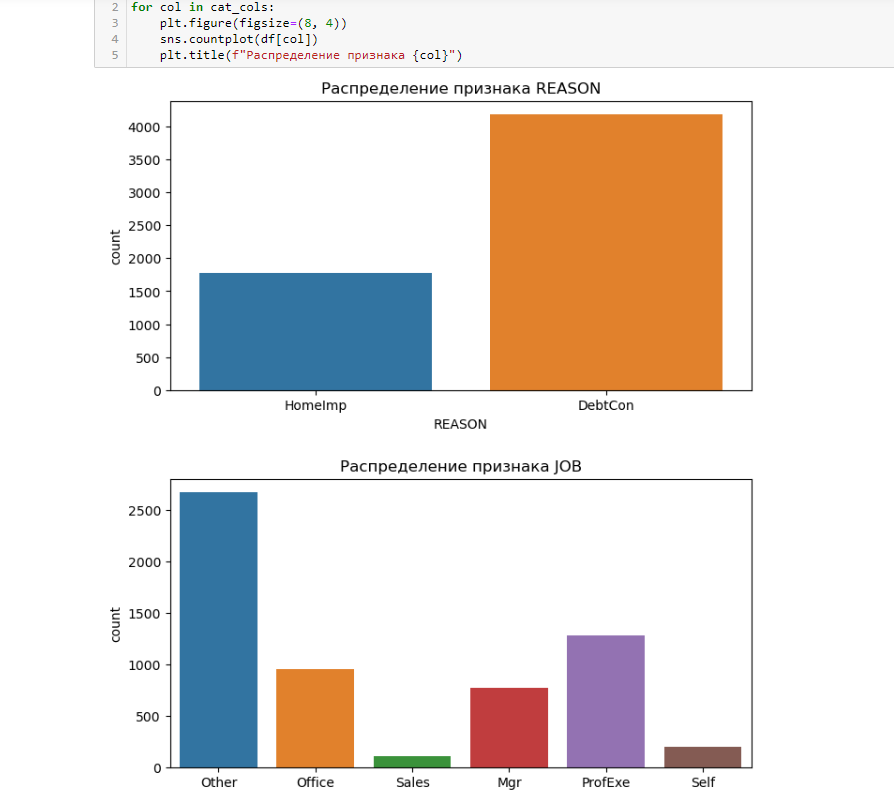


Рисунок 34 Распределение категориальных признаков

Распределение категориальных признаков в зависимости от целевого признака позволяет выявить связь между ними и определить, какие категории признаков влияют на целевую переменную. На рисунке 35 показано распределение категориальных признаков от целевой переменной BAD.

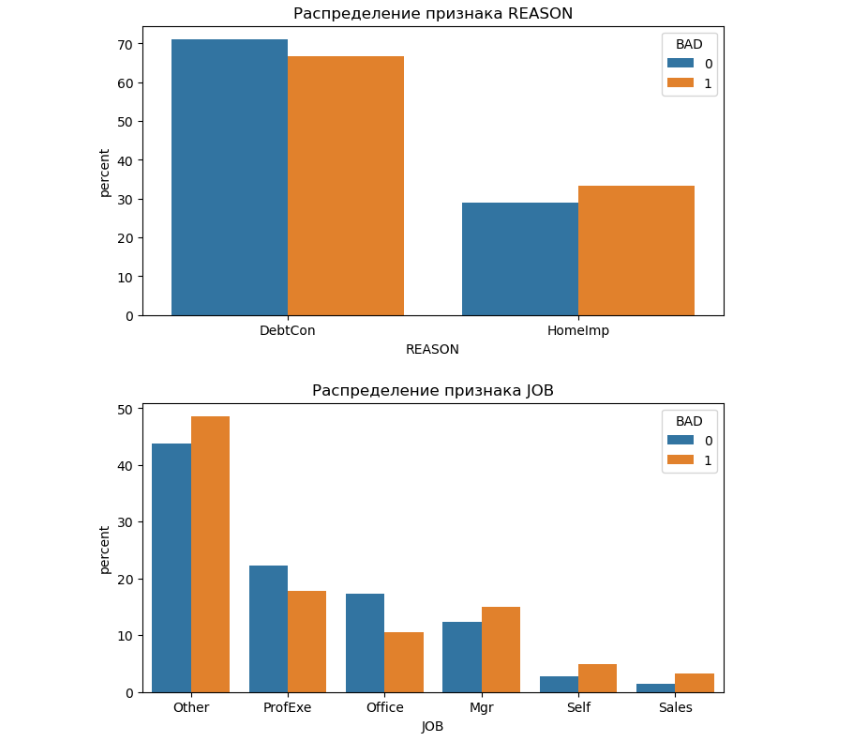


Рисунок 35 Распределение категориальных признаков от целевой переменной

Так как алгоритмы классификации, которые будут использованы работают только с числовыми признаками, столбцы REASON и JOB нужно преобразовать в числовые признаки. Для этого используется функции библиотеки sklearn OrdinalEncoder() (рис. 36).

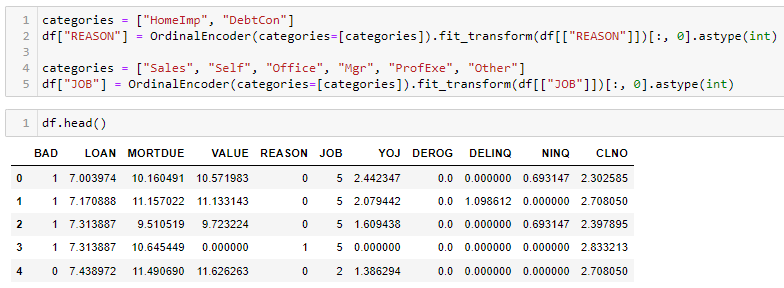


Рисунок 36 Преобразование категориальных признаков в числовые

Так же, как и для числовых признаков, определим значимость категориальных признаков с помощью дисперсионного анализа (рис. 37). Место работы больше влияет на целевую переменную, чем причина предоставления кредита, что вполне логично.

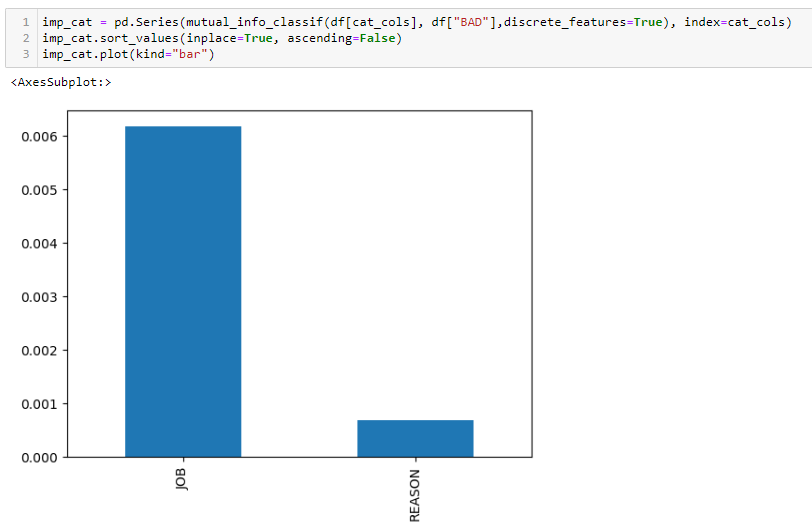


Рисунок 37 Значимость категориальных признаков

В целом, предобработка данных является важным шагом в машинном обучении, который помогает улучшить качество моделей и снизить вероятность ошибок. Она может быть выполнена с помощью различных методов и инструментов, в зависимости от характеристик данных и задачи машинного обучения. После предобработки данных можно приступать к построению и обучению моделей [33-35].

# **3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ВИЗУАЛИЗАЦИИ РАБОТЫ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ**

## **3.1 Логическая структура серверной части программного обеспечения**

Серверная часть программного обеспечения была написана на языке Python c помощью библиотек websockets.server, DEAP и sklearn.

Websockets.server позволяет создавать серверные приложения, использующие протокол WebSocket, который обеспечивает более эффективную двустороннюю связь между сервером и клиентом, чем традиционные HTTP-запросы и ответы. Он позволяет установить постоянное соединение между сервером и клиентом, что позволяет обмениваться данными в режиме реального времени. Websockets.server предоставляет простой и удобный интерфейс для создания серверных приложений на основе протокола WebSocket.

Websockets на Python работает с помощью асинхронной библиотеки asyncio. При подключении клиента к серверу, сервер и клиент устанавливают постоянное соединение, которое позволяет обмениваться данными в режиме реального времени. Когда клиент отправляет сообщение на сервер, сервер вызывает функцию-обработчик, которая может обрабатывать сообщения от нескольких клиентов одновременно.

На рисунке 38 представлена схема взаимодействия между клиентом и сервером.



Рисунок 38 Схема взаимодействия между клиентом и сервером

Реализация генетического алгоритма разбита на три модуля (alg, alg2, alg3). В каждом файле реализован генетический алгоритм для одной из моделей машинного обучение в решаемой задаче кредитного скоринга (рис. 39).

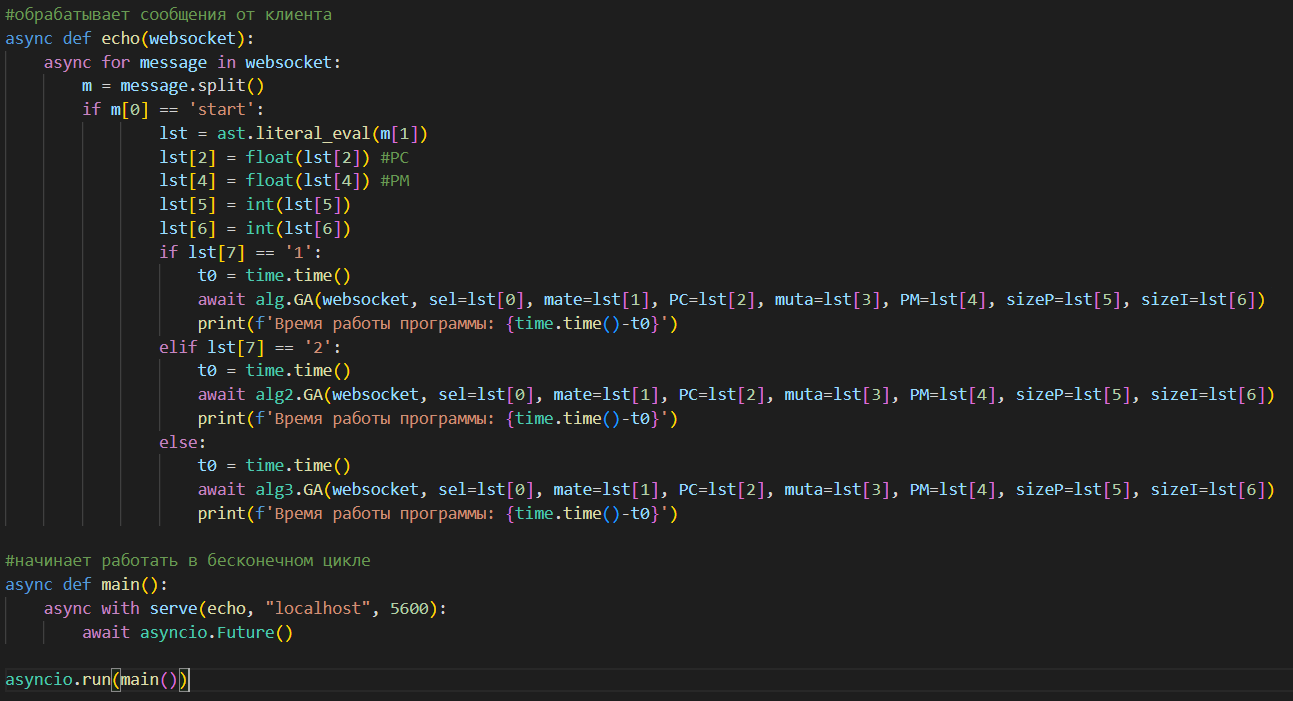


Рисунок 39 Модуль alg.py

Модули реализации генетического алгоритма содержать в себе три функции:

1. maxFitness(individual, X\_train, X\_test, y\_train, y\_test) – функция для реализации фитнес-функции;
2. mutate(individual, muta) – функция для реализации разных видов мутации;
3. GA(socket, sel='1', mate='1', PC=0.9, muta='1', PM=0.1, sizeP=5, sizeI=10) – основная функция запуска генетического алгоритма (рис. 40).



Рисунок 40 Функция GA() модуля alg.py

Гены каждого индивидуума являются гиперпараметрами для модели машинного обучения. Необходимо найти такую модель, которая в решаемой задаче кредитного скоринга давала бы наилучшие метрики качества. Фитнес-функция представляет собой метрику f1\_score() библиотеки sklearn (рис. 41). F1\_score - это функция библиотеки sklearn, которая используется для оценки качества бинарной классификации. Она вычисляет F1-меру, которая является гармоническим средним между точностью (precision) и полнотой (recall). F1-мера показывает, насколько хорошо модель может правильно классифицировать объекты обоих классов, учитывая как ложно положительные, так и ложно отрицательные результаты.

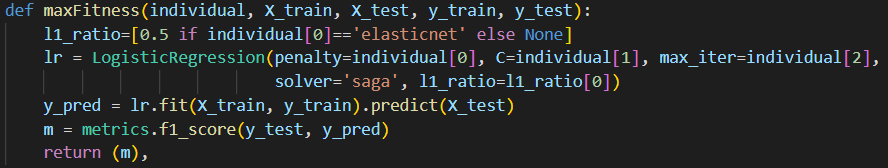


Рисунок 41 Реализация функции приспособленности

С помощью функции eaSimple() библиотеки algorithms была создана структура генетического алгоритма. Algorithms.eaSimple() реализует простой генетический алгоритм для решения задач оптимизации. Она принимает на вход популяцию, функцию приспособленности, размер популяции, вероятность скрещивания и мутации, количество поколений и дополнительные параметры (рис. 42).

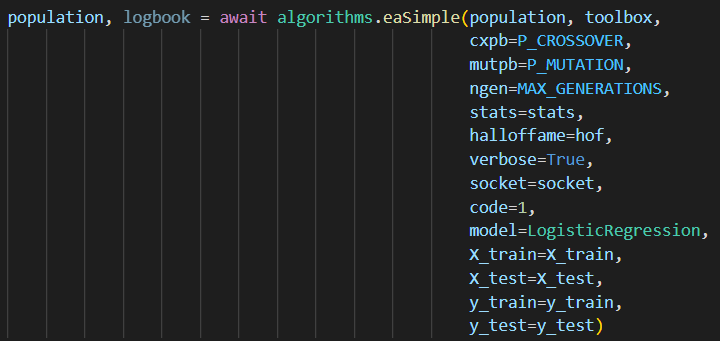


Рисунок 42 Реализация ГА с помощью algorithms.eaSimple()

Алгоритм начинается с создания начальной популяции случайных решений. Затем происходит оценка каждого решения с помощью заданной фитнес-функции. Далее происходит цикл поколений, на каждом из которых происходят шаги селекции, скрещивания, мутации, оценки функцией приспособленности и замены новыми потомками части старой популяции.

Алгоритм продолжается до тех пор, пока не будет достигнуто заданное количество поколений или не будет найдено решение, удовлетворяющее критерию остановки. В конце работы алгоритма возвращается лучшее найденное решение.

Общая схема серверной части программного обеспечения выглядит следующим образом (рис. 43):

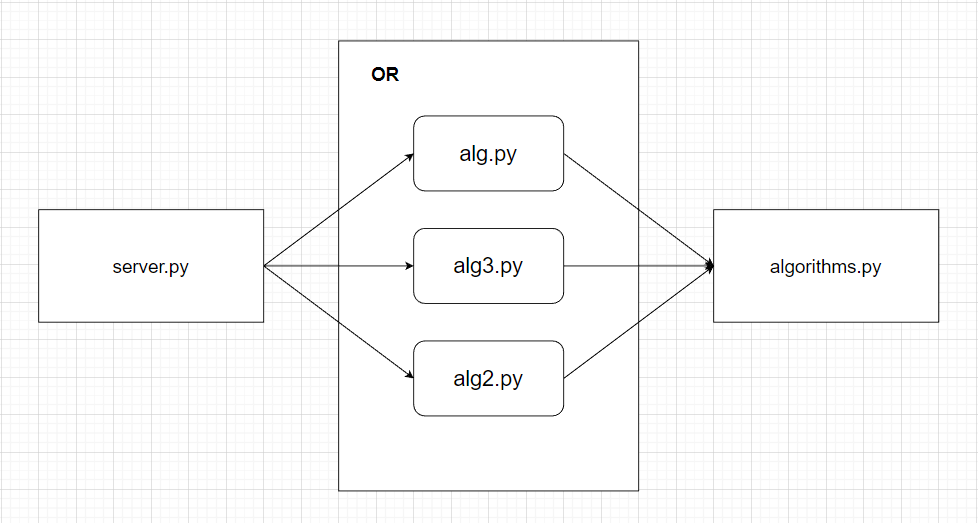


Рисунок 43 Схема серверной части программного обеспечения

## **3.2 Логическая структура клиентской части программного обеспечения**

Для реализации клиентской части программного обеспечения были использованы языки JavaScript, HTML и CSS.

С помощью протокола WebSocket в языке JavaScript было установлено соединение с сервером для передачи параметров генетического алгоритма и получения результатов работы алгоритма. WebSocket позволяет пересылать любые данные между клиентом и сервером, на любой домен, безопасно и почти без лишнего сетевого трафика.

Для использования WebSocket в JavaScript необходимо создать объект WebSocket, передав ему URL-адрес сервера (рис. 44). После этого можно добавлять обработчики событий для открытия соединения, закрытия соединения и получения сообщений от сервера.

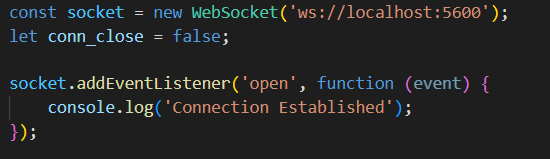


Рисунок 44 Объект WebSocket

Файл js.js включает в себя следующие основные функции и обработчики событий:

* socket.addEventListener('open', …);

Данный обработчик сообщает пользователю, что соединение с сервером установлено. В консоль выводится сообщение 'Connection Established'.

* socket.addEventListener('message', …);

Данный обработчик является основным, так как с помощью него происходит получение информации о популяциях и работе генетического алгоритма. Первое слово в приходящем сообщении от сервера – команда, с помощью которой обработчик понимает, какой блок действий ему необходимо выполнить. Работа происходит со следующими командами:

1. stats – получение статистики по популяции (максимальное и среднее значения функции приспособленности, стандартное отклонение);
2. report – получение информации о работе модели кредитного скоринга и отображение результатов на странице html;
3. gen – получение информации о индивидуумах по каждой популяции, отображение популяции и индивидуумов.

* function go();

Данная функция запускается при нажатии пользователем на кнопку «Запустить». Она отправляет серверу сообщение о начале работы алгоритма и передает ему выбранные пользователем параметры (рис. 45).

* const contactServer = () => {};

Данная функция обрабатывает запуск алгоритма при повторном нажатии на кнопку «Запустить». Она обновляет и очищает страницу html, чтобы установить повторное соединение с сервером без его перезагрузки.

* const stop = () => {}.

Данная функция принудительно останавливает работу алгоритма.

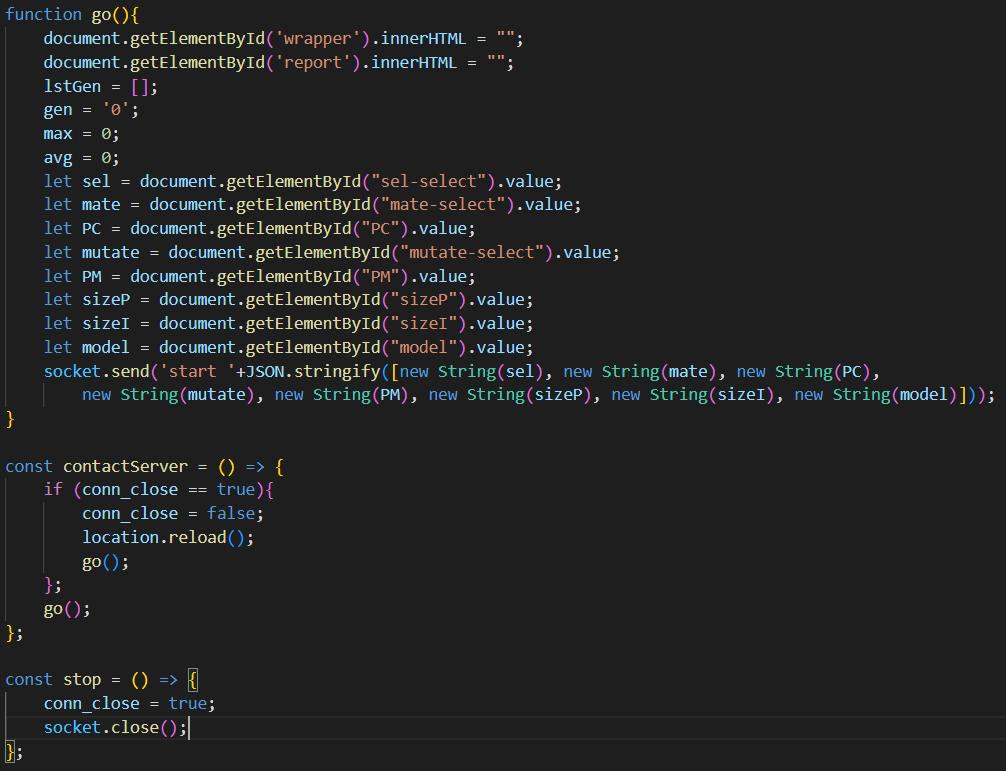


Рисунок 45 Функции модуля js.js

Общая схема клиентской части программного обеспечения выглядит следующим образом (рис. 46):

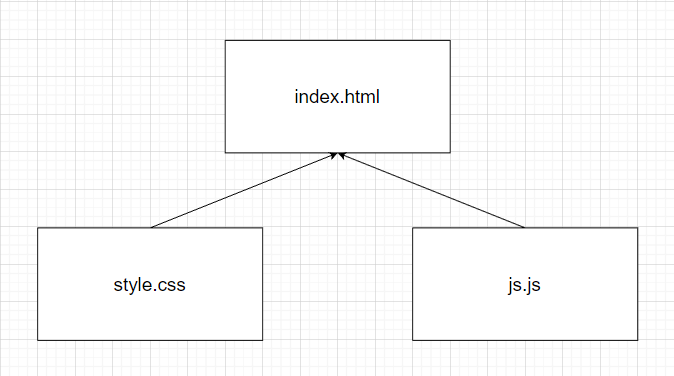


Рисунок 46 Схема клиентской части программного обеспечения

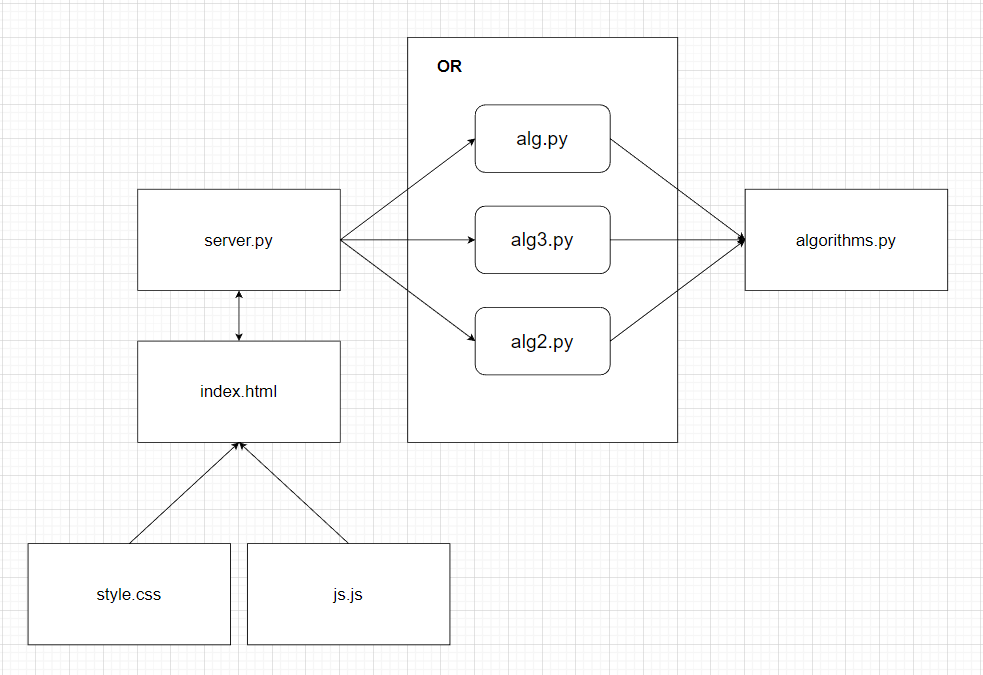
Общая схема взаимодействия модулей программного обеспечения представлена на рисунке 47: 

Рисунок 47 Общая схема взаимодействия модулей программного обеспечения

## **Описание графического интерфейса**

Для того, чтобы получить доступ к web-приложению, необходимо открыть файл index.html в браузере и запустить сервер server.py из консоли или среды разработки.

Web-приложение представляет собой одностраничный сайт, который включает в себя несколько форм: форму для выбора параметров алгоритма, форму для запуска и остановки алгоритма, форму визуализации алгоритма, форму вывода результатов работы модели кредитного скоринга.

Форма для выбора параметров алгоритма состоит из элементов input разного типа. Пользователь может выбрать параметры генетического алгоритма, а далее с помощью формы запуска и остановки алгоритма, которая включает в себя две кнопки, запустить алгоритм (рис. 48).

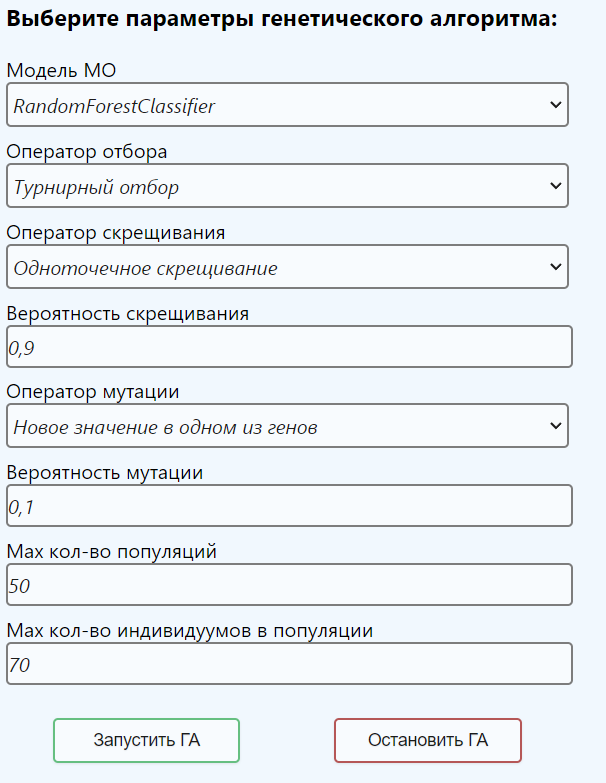


Рисунок 48 Форма выбора параметров генетического алгоритма

Форма визуализации алгоритма представляет собой прямоугольное поле, в котором отображаются популяции генетического алгоритма. Каждая популяция состроит из заданного пользователем количества индивидуумов. Индивидуумы отображаются в виде квадратов, которые в зависимости от значения функции приспособленности окрашиваются в красный или зеленый цвет (рис. 49).



Рисунок 49 Форма визуализации алгоритма

Форма вывода результатов работы генетического алгоритма и обучения модели кредитного скоринга представляет собой прямоугольное поле, в котором отображаются график работы алгоритма («Зависимость макс/средней приспособленности от поколения») и метрики качества модели машинного обучения (рис. 50).

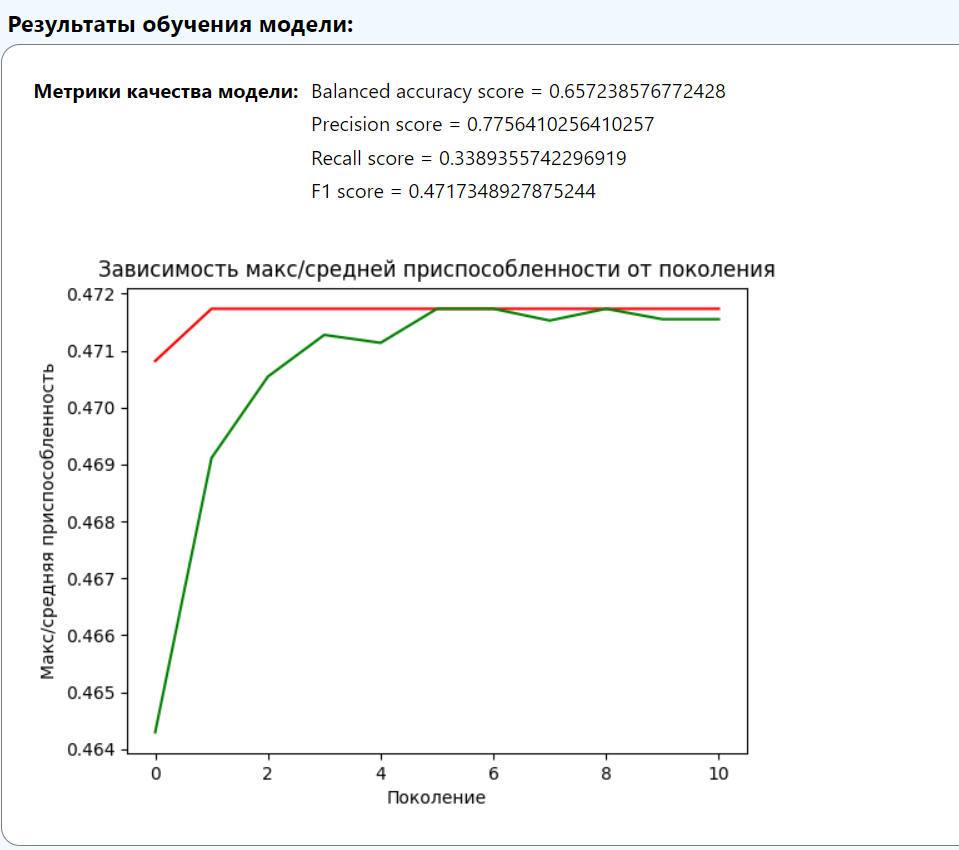


Рисунок 50 Форма вывода результатов

На данном графике видно, как зеленая линия средней приспособленности приближается к красной линии максимальной приспособленности. После пятого поколения и средняя, и максимальная приспособленности перестают расти. Это значит, что достигнуто оптимальное решение, а колебания зеленой линии объясняются мутациями в поколениях.

## **Результаты обучения скоринг-модели**

В ходе выполнения работы было проведено много запусков генетического алгоритма с разными параметрами, и в результате была составлена таблица результатов работы (рис. 51).

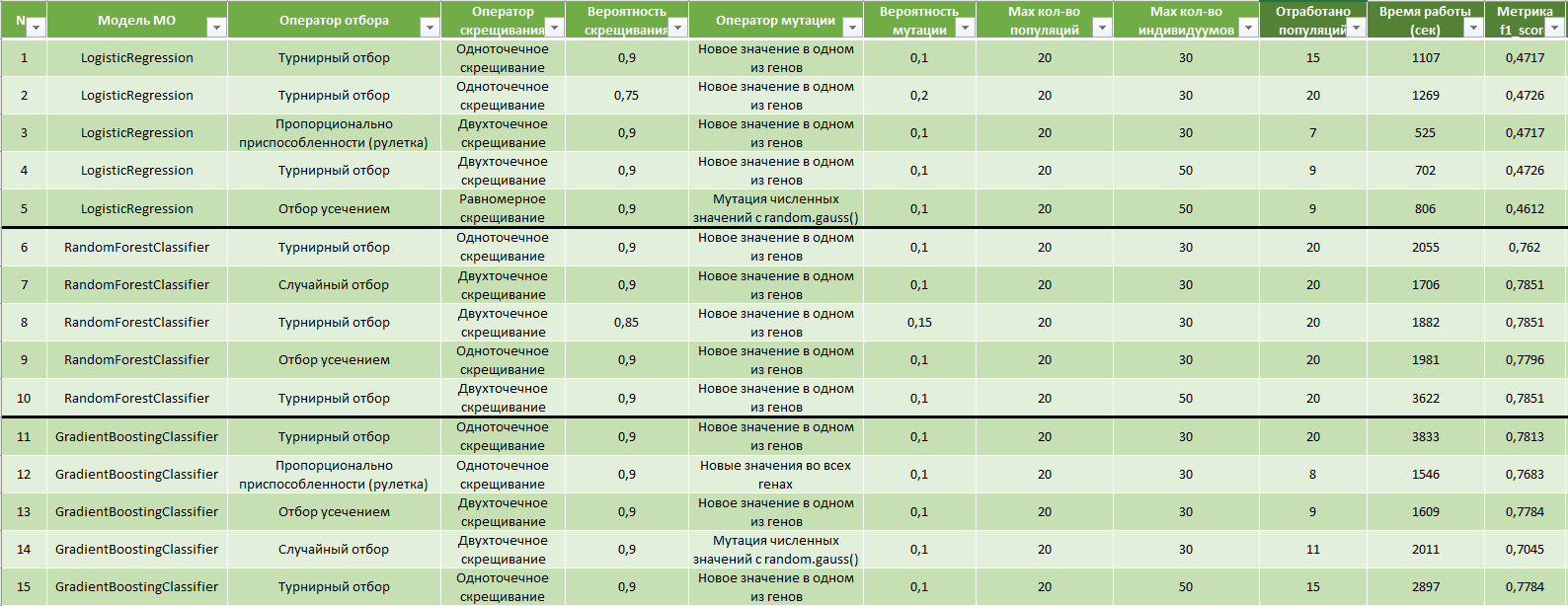


Рисунок 51 Таблица результатов

Сравнение моделей проводилось по критерию «время-точность». Необходимо было выбрать модель машинного обучения, которая при наименьшем затраченном времени работы давала бы наилучшие метрики качества и находила бы оптимальные гиперпараметры.

Будем считать, что модель находит оптимальные гиперпараметры, когда зеленая линия средней приспособленности приблизилась к красной линии максимальной приспособленности и перестала сильно колебаться.

Для модели логистической регрессии подбирались следующие гиперпараметры:

* Penalty – штраф (или регуляризация) направлен на уменьшение ошибки обобщения модели и предназначен для сдерживания и регулирования переобучения. Техника не поощряет изучение более сложной модели, чтобы избежать риска переобучения.;
* C – сила регуляризации, которая работает со штрафом, чтобы регулировать переобучение. Меньшие значения указывают на более сильную регуляризацию, а большое значение указывает модели на то, что она должна придать большой вес обучающим данным.;
* Max\_iter – максимальное количество итераций.

На рисунке 52 представлен график обучения логистической регрессии со следующими параметрами генетического алгоритма:

* Оператор отбора – турнирный отбор;
* Оператор скрещивания – одноточечное скрещивание;
* Оператор мутации – новое значение в одном из генов;
* Вероятность скрещивания = 0,9;
* Вероятность мутации = 0,1;
* Количество индивидуумов в популяции = 50;
* Максимальное количество популяций = 20.

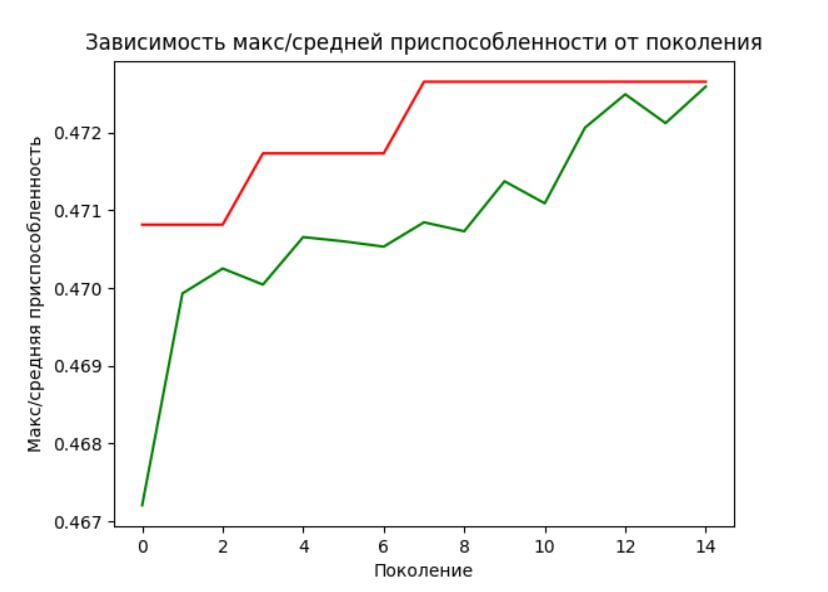


Рисунок 52 График работы логистической регрессии №1

Обратим внимание, как зеленая линия приближается к значению максимальной приспособленности. Несмотря на то, что максимальное значение было найдено на 7 поколении, среднее значение по поколениям продолжает расти и уже на 14 поколении оно почти равно максимальному значению. Метрика f1\_score такой модели равна 0,4726, а метрика balances\_accuracy равна 0,657.

Другую ситуацию можно увидеть при обучении логистической регрессии со следующими параметрами генетического алгоритма (рис. 53):

* Оператор отбора – отбор усечением;
* Оператор скрещивания – равномерное скрещивание;
* Оператор мутации – мутация численных значений с random.gauss();
* Вероятность скрещивания = 0,9;
* Вероятность мутации = 0,1;
* Количество индивидуумов в популяции = 30;
* Максимальное количество популяций = 20.

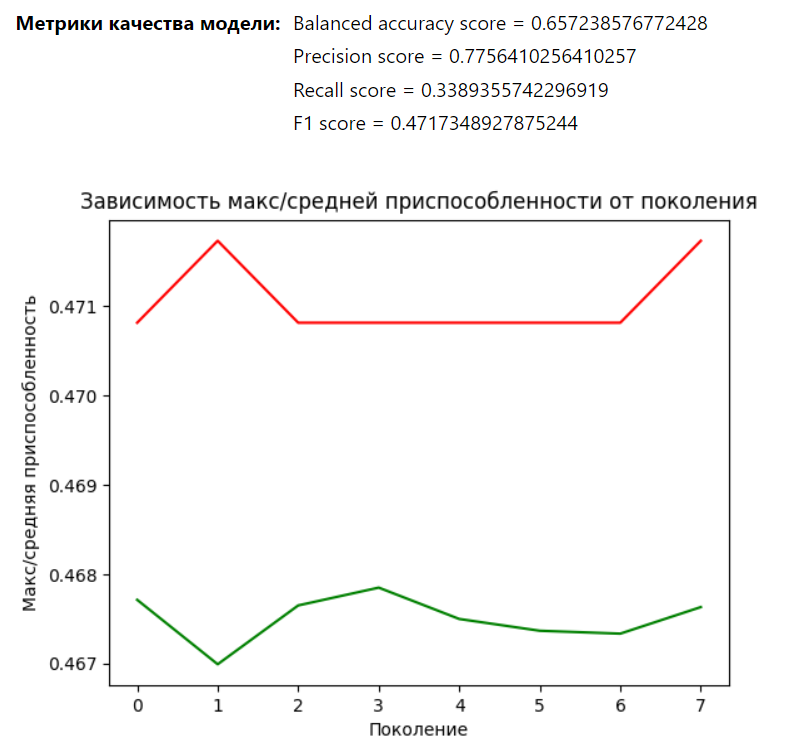


Рисунок 53 График работы логистической регрессии №2

Данная модель остановилась после 7 поколения, так как значение средней приспособленности с 4 поколения перестало значительно меняться. В программе предусмотрена остановка алгоритма, если разница между последними тремя значениями средней приспособленности меньше 0,0001. Данную модель можно считать неудачной, так как метрики качества ниже (f1\_score равна 0,4612, balances\_accuracy равна 0,626), чем у других моделей и значение средней приспособленности не приблизилось к значению максимальной приспособленности. Было проведено еще несколько запусков моделей с выбором данных операторов, результаты обучения были также меньше, чем в первом примере. Можно сделать вывод, что отбор усечением, равномерное скрещивание и мутация численных значений с random.gauss() не являются оптимальными параметрами для обучения логистической регрессии.

На рисунке 54 представлены графики обучения двух моделей логистической регрессии.

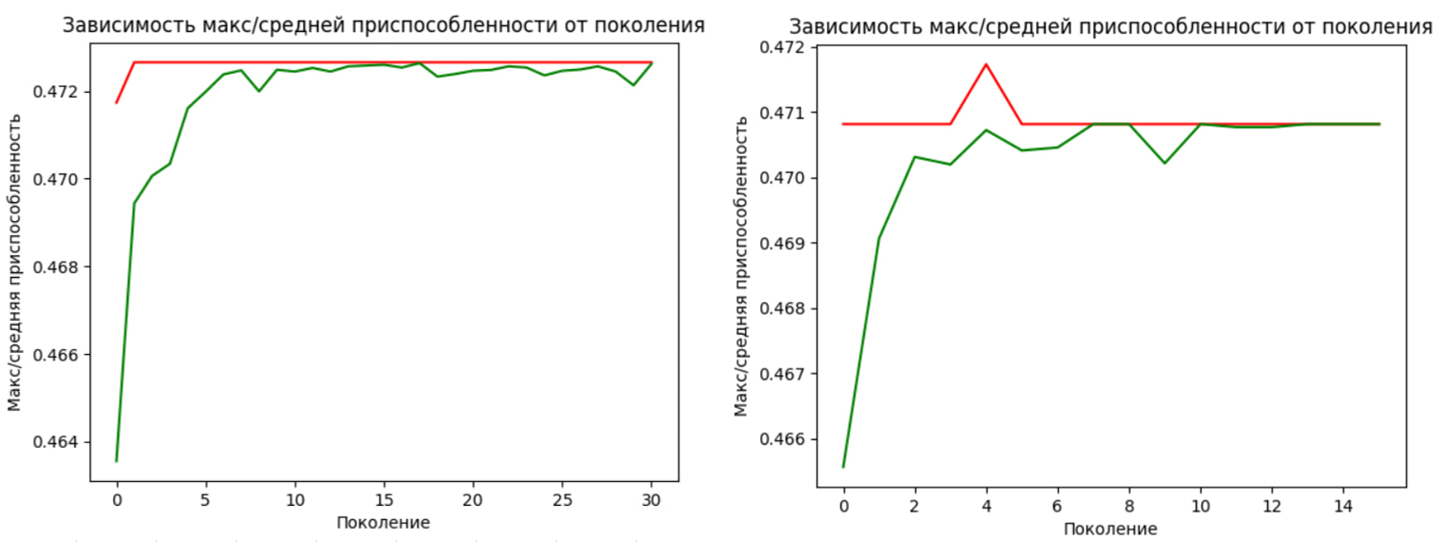


Рисунок 54 Обучение моделей логистической регрессии

На графиках видно, как уже после 2-3 поколений средняя приспособленность находится рядом с максимальной приспособленностью. На левом графике представлена модель со следующими параметрами:

* Оператор отбора – турнирный отбор;
* Оператор скрещивания – одноточечное скрещивание;
* Оператор мутации – новое значение в одном из генов;
* Вероятность скрещивания = 0,9;
* Вероятность мутации = 0,1;
* Количество индивидуумов в популяции = 50;
* Максимальное количество популяций = 30.

На правом графике модель имеет такие же параметры, кроме значения количества индивидуумов в популяции, которое равно 30, и вероятности скрещивания, которое равно 0,8. Метрики качества модели слева (f1\_score равна 0,4726, balances\_accuracy равна 0,675) чуть лучше, чем у модели справа (f1\_score равна 0,4754, balances\_accuracy равна 0,66).

По результатам запусков нескольких моделей с разными параметрами можно сделать общий вывод, что оптимальными параметрами в случае обучения логистической регрессии являются следующие значения: оператор отбора – турнирный отбор; оператор скрещивания – одноточечное скрещивание; оператор мутации – новое значение в одном из генов; вероятность скрещивания = 0,9; вероятность мутации = 0,1; количество индивидуумов в популяции = 50; максимальное количество популяций = 20.

Время работы модели логистической регрессии примерно равно 63 секундам на поколение из 30 индивидуумов. Было установлено, что изменение операторов и вероятностей отбора, скрещивания, мутации на скорость обучение не влияют.

Второй моделью стал RandomForestClassifier(). Для модели случайного леса подбирались следующие гиперпараметры:

* n\_estimators – количество деревьев в модели;
* criterion – функция для измерения качества разделения;
* max\_depth – максимальная глубина деревьев;
* min\_samples\_split – минимальное количество выборок, необходимое для разделения внутреннего узла.

На рисунке 55 представлен график обучения модели случайного леса со следующими параметрами генетического алгоритма:

* Оператор отбора – турнирный отбор;
* Оператор скрещивания – двухточечное скрещивание;
* Оператор мутации – новое значение в одном из генов;
* Вероятность скрещивания = 0,85;
* Вероятность мутации = 0,15;
* Количество индивидуумов в популяции = 30;
* Максимальное количество популяций = 20.

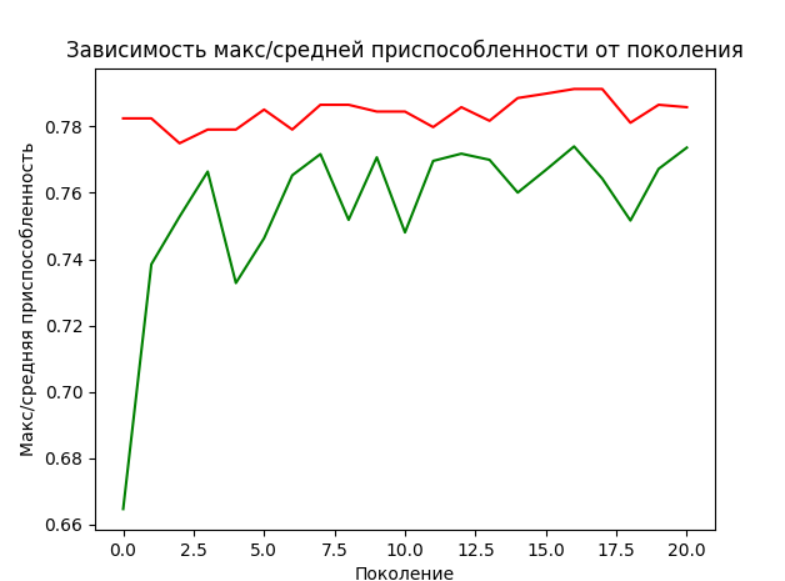


Рисунок 55 График обучения случайного леса №1

Сильные колебания значения средней приспособленности объясняются вероятность мутации, которая в данной модели равна 0,15. Можно увидеть, как и максимальная приспособленность, и средняя приспособленность растут с каждой популяцией. Данная модель получилась оптимальной. Значение f1\_score равно 0,785, balances\_accuracy равно 0,824.

При вероятности мутации 0,1 модель обучалась следующим образом (рис. 56):

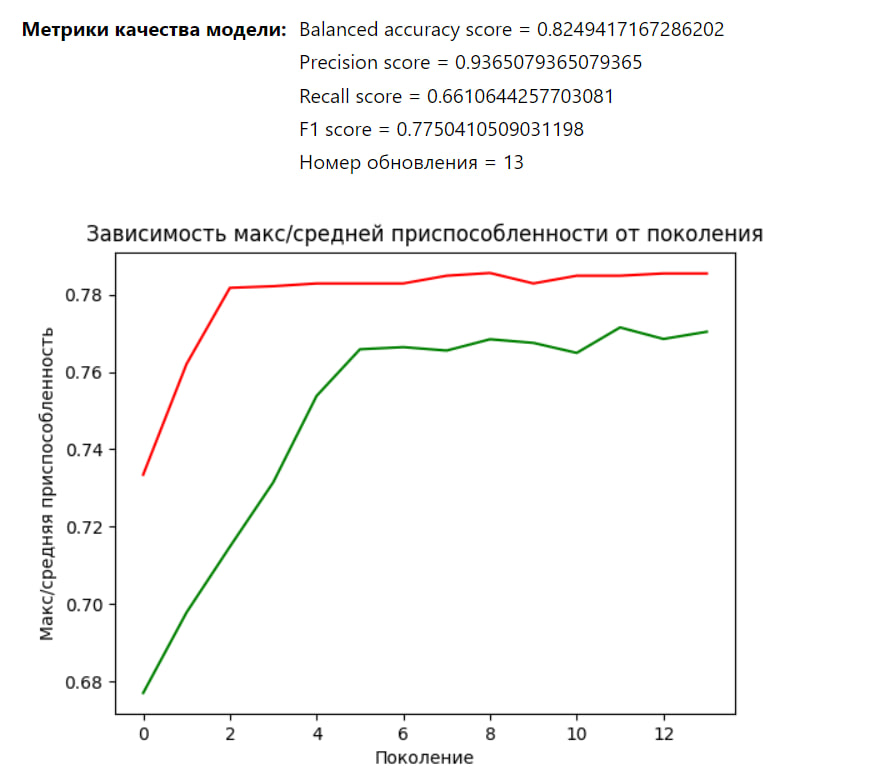


Рисунок 56 График обучения случайного леса №2

Изменение операторов отбора, скрещивания и мутации ухудшали качество модели, а значения вероятностей скрещивания и мутации в диапазоне 0,85-0,9 и 0,1-0,15 соответственно оказались, как и в случае логистической регрессии, оптимальными.

На рисунке 57 представлены две неудачные модели случайного леса.

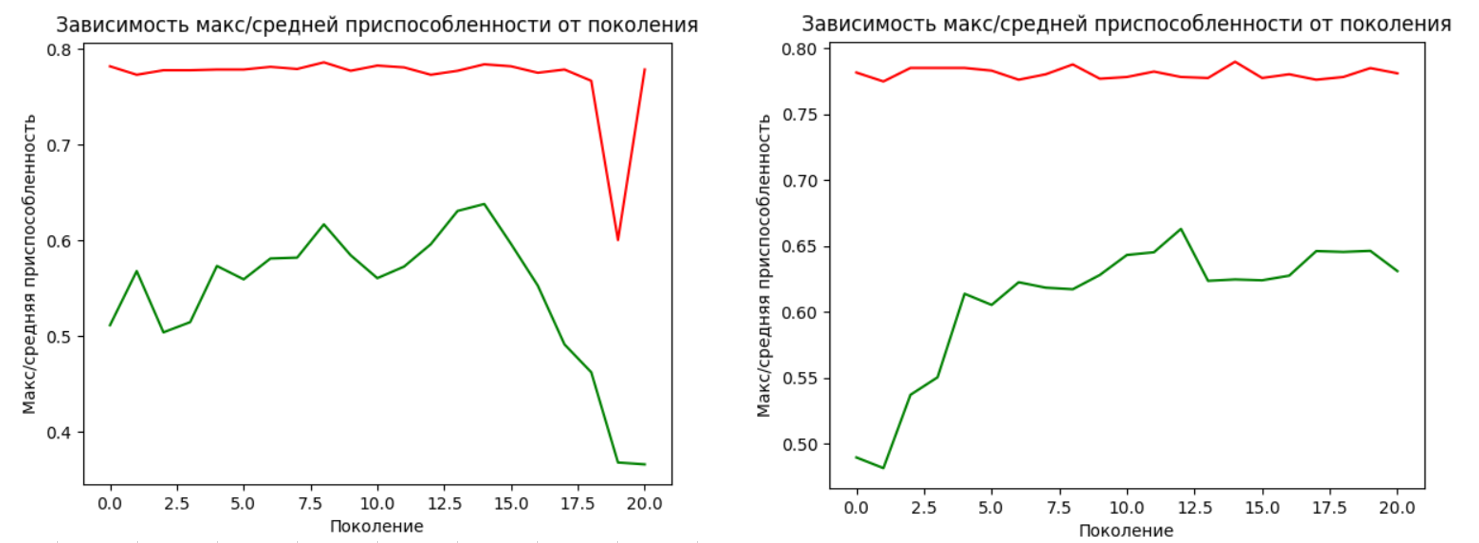


Рисунок 57 Неудачные модели случайного леса

Как мы видим и в первом, и во втором случае средняя приспособленность не приближается к значению максимальной приспособленности, а в первом случае она даже сильно падает после 13 поколения. В данных моделях количество индивидуумов равно 15, а в модели слева к этому оператор отбора – случайный отбор.

Время работы модели случайного леса примерно равно 102 секундам на поколение из 30 индивидуумов.

Третьей моделью стал GradientBoostingClassifier(). Для модели градиентного бустинга подбирались такие же гиперпараметры, как для модели случайного леса: n\_estimators, criterion, max\_depth, min\_samples\_split.

На рисунке 58 представлен график обучения оптимальной модели градиентного бустинга со следующими параметрами генетического алгоритма:

* Оператор отбора – турнирный отбор;
* Оператор скрещивания – двухточечное скрещивание;
* Оператор мутации – новое значение в одном из генов;
* Вероятность скрещивания = 0,85;
* Вероятность мутации = 0,15;
* Количество индивидуумов в популяции = 50;

Максимальное количество популяций = 20.

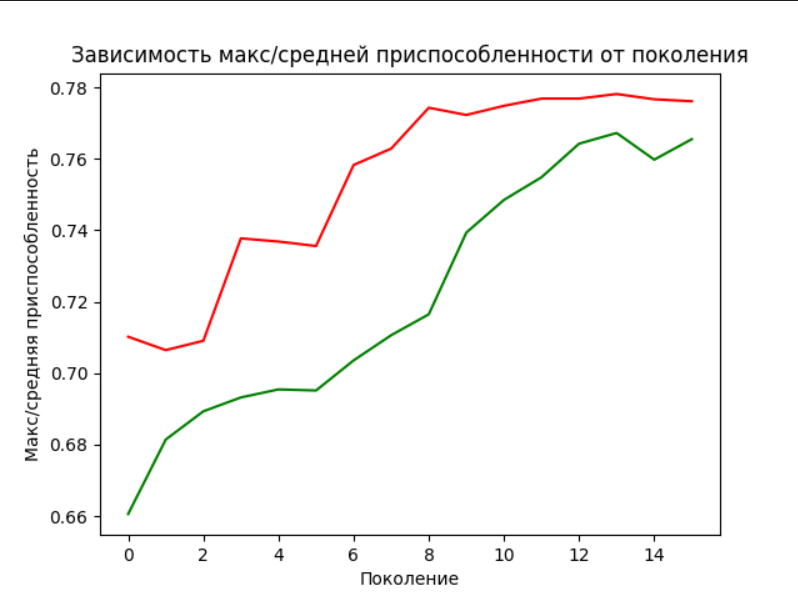


Рисунок 58 График обучения градиентного бустинга №1

Метрики качества этой модели: f1\_score = 0,7813, balances\_accuracy = 0,831. При этом время работы модели примерно равно 190 секундам на поколение из 50 индивидуумов. Модели из 30 индивидуумов обучались за 139 секунд на поколение.

Как и в случаях логистической регрессии и случайного леса изменение операторов отбора, скрещивания и мутации значительно ухудшали обучение модели. На рисунке 59 представлены неудачные модели градиентного бустинга.

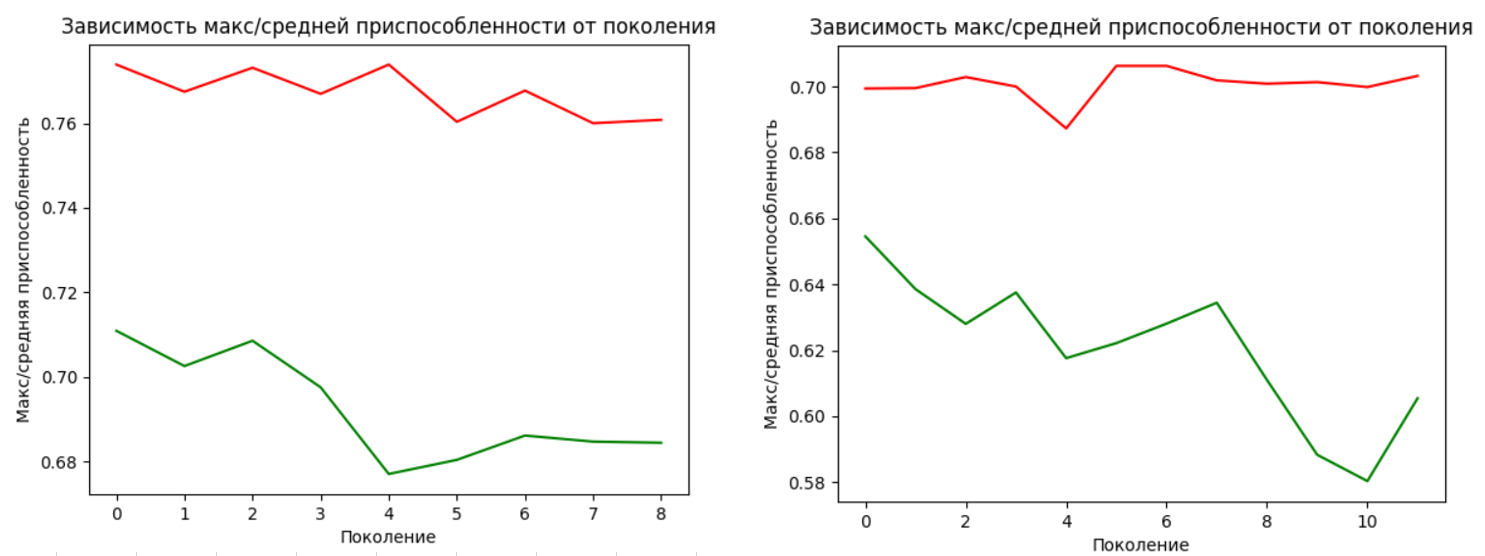


Рисунок 59 Неудачные модели градиентного бустинга

Выводы:

1. Лучшая вероятность скрещивания 0,9;
2. Лучшая вероятность мутации 0,1;
3. Логистическая регрессия показывает наихудшие результаты обучения;
4. Случайный лес и градиентный бустинг показывают одинаковые результаты обучения;
5. Логистическая регрессия работает быстрее, чем другие модели, но при этой метрики качества хуже;
6. Изменение операторов отбора, скрещивания и мутации незначительно влияет на время работы программы, но значительно влияет на качество обученной модели. Также стоит учитывать скорость работы процессора в разные моменты времени;
7. Турнирный отбор для всех моделей является оптимальным оператором отбора;
8. Количество индивидуумов в популяции влияет на скорость приближения кривой средней приспособленности к кривой максимальной приспособленности.

Таким образом, оптимальной моделью машинного обучения в задаче кредитного скоринга является модель случайного леса, так как при наименьшем времени работы она дает наилучшие метрики качества. Модель логистической регрессии, несмотря на наименьшую скорость обучения, дает очень низкий результат по метрикам качества (f1\_score ~ 47%), а модель градиентного бустинга при одинаковых метриках качества обучается на 38% медленнее, чем модель случайного леса.

# **Заключение**

В результате выполнения выпускной квалификационной работы было разработано программное обеспечение визуализации работы генетических алгоритмов, подобраны гиперпараметры для трех моделей машинного обучения, а также решена задача кредитного скоринга.

В разработанном программном обеспечении был реализован следующий функционал:

* Выбор настроек генетического алгоритма;
* Запуск и остановка генетического алгоритма;
* Пошаговая демонстрация работы алгоритма (вывод индивидуумов каждой популяции и отображение лучших и худших);
* Построение графиков работы генетического алгоритма;
* Вывод информации о работе модели кредитного скоринга.

В процессе разработки приложения были пройдены все этапы проектирования программного обеспечения, от исследования предметной области до эксплуатации разработанного программного обеспечения. Разработанное программное обеспечение полностью удовлетворяет всем требованиям.

При разработке приложения использовались языки программирования: Python, JavaScript, HTML и СSS. Архитектурой приложения стала клиент-серверная архитектура. При разработке программного обеспечения для визуализации работы генетических алгоритмов были использованы инструменты WebSocket, модели машинного обучения библиотеки sklearn, библиотека DEAP для работы с генетическими алгоритмами.

В процессе разработки программного обеспечения визуализации работы генетических алгоритмов применялись знания по анализу предметной области, проектированию веб-сервисов, машинному обучению и созданию web-интерфейса.

# **Список литературы**

1. Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы. – 2010.
2. Вирсански Э. Генетические алгоритмы на Python //М.: ДМК Пресс. – 2020.
3. Долженко А. М. и др. Сравнение скорости сходимости различных видов генетических алгоритмов //Современные тенденции развития и перспективы внедрения инновационных технологий в машиностроении, образовании и экономике. – 2016. – №. 1. – С. 176-178.
4. Паклин Н. Непрерывные генетические алгоритмы-математический аппарат. BaseGroup Labs //Сайт в Интернете http://www. basegroup. ru/genetic/real\_coded\_ga. htm. – Т. 7.
5. Панченко Т. В. Генетические алгоритмы. – 2007.
6. Завгородний С. Д. и др. Генетические алгоритмы и их применение. – 2017.
7. Галушин П. В., Семенкин Е. С. Асимптотический вероятностный генетический алгоритм //Сибирский аэрокосмический журнал. – 2009. – №. 4. – С. 37-42.
8. Брестер К. Ю., Семенкин Е. С. Адаптивный генетический алгоритм многокритериальной оптимизации. – 2013.
9. Сопов Е. А., Сопов С. А. Вероятностный генетический алгоритм решения сложных задач многокритериальной оптимизации с адаптивной мутацией и прогнозом множества Парето //Вестник Самарского государственного аэрокосмического университета им. академика СП Королёва (национального исследовательского университета). – 2011. – №. 6. – С. 273-282.
10. Семенов Г. Е., Кейно П. П. Применение математических моделей на основе генетических алгоритмов в задачах планирования сложных технических объектов //Прикладная информатика. – 2019. – Т. 14. – №. 2. – С. 56-62.
11. Емельянов А. А. и др. Комплексное имитационное моделирование с применением генетических алгоритмов //Прикладная информатика. – 2017. – Т. 12. – №. 6. – С. 89-100.
12. Коротеев М.В. Учебное пособие по дисциплине «Анализ данных и машинное обучение» - 2018.
13. Мюллер А., Гвидо С. Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными //Санкт-Петербург: ООО «Альфа-книга. – 2017.
14. Рашка С., Мирджалили В. Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2 //Санкт-Петербург: ООО «Диалектика. – 2020.
15. Coelho L. P., Richert W. Building machine learning systems with Python. – Packt Publishing Ltd, 2015.
16. Bolstad W. M., Curran J. M. Introduction to Bayesian statistics. – John Wiley & Sons, 2016.
17. Bari A., Chaouchi M., Jung T. Predictive analytics for dummies. – John Wiley & Sons, 2016.
18. Макфарланд Д. С. JavaScript и jQuery. – 2012.
19. Флэнаган Д. JavaScript. Карманный справочник. Сделайте веб-страницы интерактивными!/Перевод АГ Сысонюк //Москва.: Издательский дом" Вильямс. – 2015. – С. 320.
20. Фримен Э., Робсон Э. Изучаем программирование на JavaScript //Санкт-Петербург: Питер. – 2015.
21. Duckett J. Web design with HTML, CSS, JavaScript and jQuery set. – IN : Wiley, 2014. – Т. 1.
22. Шмитт К. CSS. Рецепты программирования, 3-е изд. – БХВ-Петербург, 2011.
23. Кантор И. Современный учебник JavaScript //Современный учебник Java Script [Электронный ресурс]/И. Кантор.–2014.–Режим доступа: http://learn. javascript. ru.–Дата доступа. – 2014. – Т. 12.
24. Мержевич В. Справочник по HTML //URL: http://htmlbook. ru.
25. JavaScript Я. Электронный учебник //URL: https://learn. javascript. ru/json (дата обращения 19.04. 2017).
26. Release P. T. 2.6. 3. docs@ python. org //Python Software Foundation. – 2009.
27. Сузи Р. А. Язык программирования Python //М.: Бином. Лаборатория знаний. – 2006.
28. Kramer O., Kramer O. Scikit-learn //Machine learning for evolution strategies. – 2016. – С. 45-53.
29. Porcu V., Porcu V. Matplotlib //Python for Data Mining Quick Syntax Reference. – 2018. – С. 201-234.
30. Fortin F. A. et al. DEAP: Evolutionary algorithms made easy //The Journal of Machine Learning Research. – 2012. – Т. 13. – №. 1. – С. 2171-2175.
31. Coelli T. A guide to DEAP version 2.1: a data envelopment analysis (computer) program //Centre for Efficiency and Productivity Analysis, University of New England, Australia. – 1996. – Т. 96. – №. 08. – С. 1-49.
32. Бабенко М., Левин М. Введение в теорию алгоритмов и структур данных. – Litres, 2021.
33. Кулаичев А. П. Методы и средства комплексного анализа данных. – 2006.
34. Маккинни У. Python и анализ данных. – Litres, 2022.
35. Чашкин Ю. Р. Математическая статистика. Анализ и обработка данных: Учебное пособие //Ростов-на-Дону: Феникс. – 2010. – Т. 236.

# **Приложение 1. Файл js.js**

const socket = new WebSocket('ws://localhost:5600');

let conn\_close = false;

socket.addEventListener('open', function (event) {

console.log('Connection Established');

});

let lstGen = [];

let gen = '0';

let max = 0;

let avg = 0;

let std = 0;

let history\_std = [];

let count = 0;

//выполняет то, что надо, когда состояние месса от сервера(сокета)

socket.addEventListener('message', function (event) {

var arr = event.data.split('|');

if (arr[0]=='stats') {

max = Number(arr[1]).toFixed(7);

avg = Number(arr[2]).toFixed(7);

std = Number(arr[3]).toFixed(7);

} else if (arr[0]=='report'){

count+=1;

document.getElementById('report').innerHTML = "";

document.getElementById('report').insertAdjacentHTML('beforeend', `<h4 style="margin: 5px;">Метрики качества модели:</h4>`);

document.getElementById('report').insertAdjacentHTML('beforeend', `<div class="report2" id="report2" style="width: 70%;">

<p>Balanced accuracy score = ${arr[1]}</p>

<p>Precision score = ${arr[2]}</p>

<p>Recall score = ${arr[3]}</p>

<p>F1 score = ${arr[4]}</p>

</div>`);

document.getElementById('report').insertAdjacentHTML('beforeend', `<img src="../gr${arr[5]}.png">`);

} else {

arr[1] = arr[1].replace(/[^a-zа-яё0-9.\s]/gi, '').split(' ');

arr[2] = Number(arr[2].replace(/[^0-9.]/g, '')).toFixed(7);

if (arr[0] == gen){

if (arr[3]==1){

lstGen.push([`<div class="fl-col" style="border-color: svet">

<div class="fl-col-text">

<p>Gen: ${arr[0]}</p>

<p>Penalty: ${arr[1][0]}</p>

<p>C: ${arr[1][1]}</p>

<p>MaxIter: ${arr[1][2]}</p>

<p>FitnessF = ${arr[2]}</p>

</div>

</div>`, arr[2]]);

} else {

lstGen.push([`<div class="fl-col" style="border-color: svet">

<div class="fl-col-text">

<p>Gen: ${arr[0]}</p>

<p>N\_estimators: ${arr[1][0]}</p>

<p>Criterion: ${arr[1][1]}</p>

<p>Max\_depth: ${arr[1][2]}</p>

<p>MinSamples\_split: ${arr[1][3]}</p>

<p>FitnessF = ${arr[2]}</p>

</div>

</div>`, arr[2]]);

};

} else {

document.getElementById('wrapper').insertAdjacentHTML('beforeEnd', `<div class="f1-row" id="${gen}"></div>`);

for (let ind of lstGen) {

if (ind[1] >= (avg)) {

ind[0] = ind[0].replace("svet", "#66BF81");

document.getElementById(gen).insertAdjacentHTML('beforeEnd', ind[0]);

} else if (ind[1] <= (avg)) {

ind[0] = ind[0].replace("svet", "#B55757");

document.getElementById(gen).insertAdjacentHTML('beforeEnd', ind[0]);

} else {

ind[0] = ind[0].replace("svet", "#85acce");

document.getElementById(gen).insertAdjacentHTML('beforeEnd', ind[0]);

};

};

gen = arr[0];

lstGen = [];

if (arr[3]==1){

lstGen.push([`<div class="fl-col" style="border-color: svet">

<div class="fl-col-text">

<p>Gen: ${arr[0]}</p>

<p>Penalty: ${arr[1][0]}</p>

<p>C: ${arr[1][1]}</p>

<p>MaxIter: ${arr[1][2]}</p>

<p>FitnessF = ${arr[2]}</p>

</div>

</div>`, arr[2]]);

} else {

lstGen.push([`<div class="fl-col" style="border-color: svet">

<div class="fl-col-text">

<p>Gen: ${arr[0]}</p>

<p>N\_estimators: ${arr[1][0]}</p>

<p>Criterion: ${arr[1][1]}</p>

<p>Max\_depth: ${arr[1][2]}</p>

<p>MinSamples\_split: ${arr[1][3]}</p>

<p>FitnessF = ${arr[2]}</p>

</div>

</div>`, arr[2]]);

};

history\_std.push(avg);

if (history\_std.length==3){

if (Math.abs(Number(history\_std[0])-Number(history\_std[1]))<0.0001 && Math.abs(Number(history\_std[1])-Number(history\_std[2]))<0.0001){

stop();

} else {

history\_std.shift();

};

};

};

};

});

function go(){

document.getElementById('wrapper').innerHTML = "";

document.getElementById('report').innerHTML = "";

lstGen = [];

gen = '0';

max = 0;

avg = 0;

let sel = document.getElementById("sel-select").value;

let mate = document.getElementById("mate-select").value;

let PC = document.getElementById("PC").value;

let mutate = document.getElementById("mutate-select").value;

let PM = document.getElementById("PM").value;

let sizeP = document.getElementById("sizeP").value;

let sizeI = document.getElementById("sizeI").value;

let model = document.getElementById("model").value;

socket.send('start '+JSON.stringify([new String(sel), new String(mate), new String(PC),

new String(mutate), new String(PM), new String(sizeP), new String(sizeI), new String(model)]));

}

const contactServer = () => {

if (conn\_close == true){

conn\_close = false;

location.reload();

go();

};

go();

};

const stop = () => {

conn\_close = true;

socket.close();

};

# **Приложение 2. Файл server.py**

import asyncio

import alg, alg2, alg3

from websockets.server import serve

import ast

import time

async def echo(websocket):

async for message in websocket:

m = message.split()

if m[0] == 'start':

lst = ast.literal\_eval(m[1])

lst[2] = float(lst[2]) #PC

lst[4] = float(lst[4]) #PM

lst[5] = int(lst[5])

lst[6] = int(lst[6])

if lst[7] == '1':

t0 = time.time()

await alg.GA(websocket, sel=lst[0], mate=lst[1], PC=lst[2], muta=lst[3], PM=lst[4], sizeP=lst[5], sizeI=lst[6])

print(f'Время работы программы: {time.time()-t0}')

elif lst[7] == '2':

t0 = time.time()

await alg2.GA(websocket, sel=lst[0], mate=lst[1], PC=lst[2], muta=lst[3], PM=lst[4], sizeP=lst[5], sizeI=lst[6])

print(f'Время работы программы: {time.time()-t0}')

else:

t0 = time.time()

await alg3.GA(websocket, sel=lst[0], mate=lst[1], PC=lst[2], muta=lst[3], PM=lst[4], sizeP=lst[5], sizeI=lst[6])

print(f'Время работы программы: {time.time()-t0}')

async def main():

async with serve(echo, "localhost", 5600):

await asyncio.Future()

asyncio.run(main())

# **Приложение 3. Файл alg.py**

import pandas as pd

import numpy as np

import random

import os

import time

import matplotlib.pyplot as plt

import algorithms

import asyncio

from decimal import \*

from deap import base

from deap import creator

from deap import tools

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn import metrics

from sklearn.metrics import precision\_score, recall\_score, auc, accuracy\_score, roc\_auc\_score,f1\_score,log\_loss, classification\_report, roc\_curve

import warnings

warnings.filterwarnings("ignore");

np.set\_printoptions(precision=8)

def maxFitness(individual, X\_train, X\_test, y\_train, y\_test):

l1\_ratio=[0.5 if individual[0]=='elasticnet' else None]

lr = LogisticRegression(penalty=individual[0], C=individual[1], max\_iter=individual[2],

solver='saga', l1\_ratio=l1\_ratio[0])

y\_pred = lr.fit(X\_train, y\_train).predict(X\_test)

m = metrics.f1\_score(y\_test, y\_pred)

return (m),

def mutate(individual, muta):

if muta == '1':

i = random.randint(1,3)

if i == 1:

individual[0]=random.choice(['l1', 'l2', 'elasticnet', 'none'])

elif i == 2:

individual[1] = random.uniform(0, 4)

else:

individual[2] = random.randint(10, 700)

elif muta == '2':

individual[0]=random.choice(['l1', 'l2', 'elasticnet', 'none'])

individual[1] = random.uniform(0, 4)

individual[2] = random.randint(10, 700)

elif muta == '3':

#individual[1] = int(individual[1]\*random.gauss(1, 0.5))

individual[2] = int(individual[2]\*random.gauss(1, 0.5))

return (individual),

async def GA(socket, sel='1', mate='1', PC=0.9, muta='1', PM=0.1, sizeP=5, sizeI=10):

df = pd.read\_csv("C:/Users/Kseniya/Desktop/diplom/PROJECT/server/dataNEW.csv")

df = df.drop('Unnamed: 0', axis=1)

X = df.drop('BAD', axis=1)

y = df['BAD']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, stratify=y, shuffle=True, random\_state=123)

# константы генетического алгоритма

POPULATION\_SIZE = sizeI # количество индивидуумов в популяции

P\_CROSSOVER = PC # вероятность скрещивания

P\_MUTATION = PM # вероятность мутации индивидуума

MAX\_GENERATIONS = sizeP # максимальное количество поколений

creator.create("FitnessMax", base.Fitness, weights=(1.0,))

creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()

toolbox.register("penalty", random.choice, ['l1', 'l2', 'elasticnet', 'none'])

toolbox.register("C", random.uniform, 0, 4)

toolbox.register("maxIter", random.randint, 10, 700)

toolbox.register("individualCreator", tools.initCycle, creator.Individual,

(toolbox.penalty, toolbox.C, toolbox.maxIter), 1)

toolbox.register("populationCreator", tools.initRepeat, list, toolbox.individualCreator)

population = toolbox.populationCreator(n=POPULATION\_SIZE)

toolbox.register("evaluate", maxFitness, X\_train=X\_train, X\_test=X\_test, y\_train=y\_train, y\_test=y\_test)

if sel == '1':

toolbox.register("select", tools.selTournament, tournsize=3)

elif sel == '2':

toolbox.register("select", tools.selRoulette)

elif sel == '3':

toolbox.register("select", tools.selRandom)

elif sel == '4':

toolbox.register("select", tools.selBest)

if mate == '1':

toolbox.register("mate", tools.cxOnePoint)

elif mate == '2':

toolbox.register("mate", tools.cxTwoPoint)

elif mate == '3':

toolbox.register("mate", tools.cxUniform, indpb=0.5)

toolbox.register("mutate", mutate, muta=muta)

stats = tools.Statistics(lambda ind: ind.fitness.values)

stats.register("max", np.max)

stats.register("avg", np.mean)

stats.register("std", np.std)

HALL\_OF\_FAME\_SIZE = 10

hof = tools.HallOfFame(HALL\_OF\_FAME\_SIZE)

population, logbook = await algorithms.eaSimple(population, toolbox,

cxpb=P\_CROSSOVER,

mutpb=P\_MUTATION,

ngen=MAX\_GENERATIONS,

stats=stats,

halloffame=hof,

verbose=True,

socket=socket,

code=1,

model=LogisticRegression,

X\_train=X\_train,

X\_test=X\_test,

y\_train=y\_train,

y\_test=y\_test)

await asyncio.sleep(2)