

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**  
**по курсу**

«Data Science Pro»

**Тема:** «Прогнозирование конечных свойств новых материалов  
(композиционных материалов)»

Слушатель

Хазов Илья Александрович

Москва, 2025

## Содержание

<b>Введение .....</b>	<b>4</b>
<b>1. Аналитическая часть.....</b>	<b>6</b>
<b>1.1 Постановка задачи .....</b>	<b>6</b>
<b>1.2 Описание используемых методов .....</b>	<b>8</b>
<b>1.2.1. Линейная регрессия .....</b>	<b>9</b>
<b>1.2.2. Лассо-регрессия (Lasso).....</b>	<b>10</b>
<b>1.2.3. Гребневая регрессия (Ridge regression).....</b>	<b>11</b>
<b>1.2.4. Градиентный бустинг .....</b>	<b>12</b>
<b>1.2.5. Метод опорных векторов (SVR) .....</b>	<b>13</b>
<b>1.2.6. Метод К-ближайших соседей (KNN) .....</b>	<b>13</b>
<b>1.2.7. Случайный лес (Random Forest) .....</b>	<b>14</b>
<b>1.2.8. Дерево решений (Decision Tree).....</b>	<b>15</b>
<b>1.2.9. Многослойный перцептрон (MLPRegressor).....</b>	<b>16</b>
<b>1.2.10. Нейронная сеть .....</b>	<b>17</b>
<b>1.3. Разведочный анализ данных.....</b>	<b>18</b>
<b>2. Практическая часть.....</b>	<b>23</b>
<b>2.1. Препроцессинг данных .....</b>	<b>23</b>
<b>2.2. Выбор и обучение моделей .....</b>	<b>24</b>
<b>2.2.1. Прогноз параметра модуля упругости при растяжении .....</b>	<b>25</b>
<b>2.2.2. Прогноз параметра прочности при растяжении.....</b>	<b>28</b>
<b>2.3. Прогноз параметра соотношение «матрица-наполнитель» .....</b>	<b>29</b>
<b>2.3.1. Применение MLPRegressor (библиотека sklearn).....</b>	<b>29</b>
<b>2.3.2. Применение нейронной сети tensorflow.keras .....</b>	<b>31</b>

<b>2.4. Оценка точности моделей на тренировочном и тестовом выборках ....</b>	<b>35</b>
<b>Заключение .....</b>	<b>39</b>
<b>Библиографический список .....</b>	<b>41</b>

## Введение

Композиты представляют собой гетерогенные системы, объединяющие две или более фазы с выраженной границей раздела. Основу материала составляет матрица (связующее), выполняющая функцию распределения нагрузок, и армирующий компонент (наполнитель), придающий специфические свойства. Взаимодействие на интерфейсе фаз создаёт синергию, недостижимую для исходных компонентов в изолированном состоянии.

Классическим примером служит железобетон, где цементная матрица сопротивляется сжатию, а стальные прутья компенсируют низкую устойчивость к растяжению. В инновационных разработках акцент смещается на сочетание полимерных связующих с волокнистыми наполнителями — стеклянными, карбоновыми или базальтовыми. Последние приобретают особую актуальность в контексте устойчивого развития.

Сырьём для производства базальтовых волокон служит магматическая порода, формирующаяся при кристаллизации лавы. Широкая распространённость базальта и его термостойкость обусловили применение в композитах для авиакосмической и строительной отраслей. Базальтопластики демонстрируют превосходство над металлическими аналогами по коррозионной стойкости, удельному весу и диэлектрическим свойствам, что определяет их как ключевой объект исследования в работе.

Расширение номенклатуры композитов сталкивается с фундаментальной проблемой: свойства конечного материала нелинейно зависят от параметров компонентов. Традиционный подход, основанный на цикле экспериментальных испытаний, требует значительных ресурсов. Альтернативой выступает компьютерное моделирование, включающее машинное обучение для оптимизации состава матрица-наполнитель.

В ходе проведённого исследования были разработаны несколько моделей, обеспечивающих предсказание параметров модулей упругости и прочности при

растяжении. Дополнительно созданы две нейронные сети, предназначенные для подбора оптимального соотношения компонентов «матрица – наполнитель». Одна из моделей нейронной сети легла в основу веб-приложения с удобным интерфейсом, реализованного на фреймворке Flask.

## 1. Аналитическая часть

### 1.1 Постановка задачи

В качестве сырых данных были предоставлены два файла:

1) «X\_bp.xlsx» — характеристики (параметры) базальтопластика с размерностью 1023 строки на 10 столбцов (после удаления лишнего столбца индексов);

2) «X\_nup.xlsx» — характеристики нашивки углепластика с размерностью 1040 строк на 3 столбца (после удаления лишнего столбца индексов).

```
# датасеты загружены через облако google drive
X_bp = pd.read_excel('/content/drive/MyDrive/DatasetsVKR/X_bp.xlsx') # датасет с характеристиками базальтопластика
X_bp.drop('Unnamed: 0', axis = 1, inplace=True) # избавляемся от лишнего столбца индексов
print(X_bp.shape) # выводим размерность 1го датасета
X_bp.head()

(1023, 10)

X_nup = pd.read_excel('/content/drive/MyDrive/DatasetsVKR/X_nup.xlsx') # датасет с характеристиками углепластика
X_nup.drop('Unnamed: 0', axis = 1, inplace=True) # избавляемся от лишнего столбца индексов
print(X_nup.shape) # выводим размерность 2го датасета
X_nup.head()

(1040, 3)
```

Рисунок 1 – Выгрузка исходных файлов .xlsx с удалением столбца индексов

Для дальнейшей работы с нашими данными необходимо объединить две таблицы в единый датасет типом INNER по индексу. После объединения данных часть строк из файла X\_nup была исключена. Дальнейший анализ проводится на итоговом датасете, включающем 13 признаков и 1023 наблюдения.

```
df_X = X_bp.merge(X_nup, left_index = True, right_index = True, how = 'inner')
print(f'Итоговый датасет имеет размерность {df_X.shape}, на первоначальном этапе обработки данных было отброшено {X_nup.shape[0]-df_X.shape[0]} строк из 2го датасета')
df_X.head().T

Итоговый датасет имеет размерность (1023, 13), на первоначальном этапе обработки данных было отброшено 17 строк из 2го датасета
```

Рисунок 2 – Объединение таблиц по типу INNER

В ходе исследовательской работы целевыми параметрами, которые будут предсказываться моими моделями, являются модуль упругости при растяжении (Гпа), прочность при растяжении (Мпа) и соотношение матрица-наполнитель.

После создания единого датасета была замечено, что параметр «Угол нашивки, град» имеет всего 2 уникальных значения, поэтому данная характеристика анализируется в качестве категориального признака.

Все остальные признаки в нашем датасете имеют вещественный тип (float64), являются непрерывными и количественными. При этом пропуски в данных отсутствуют.

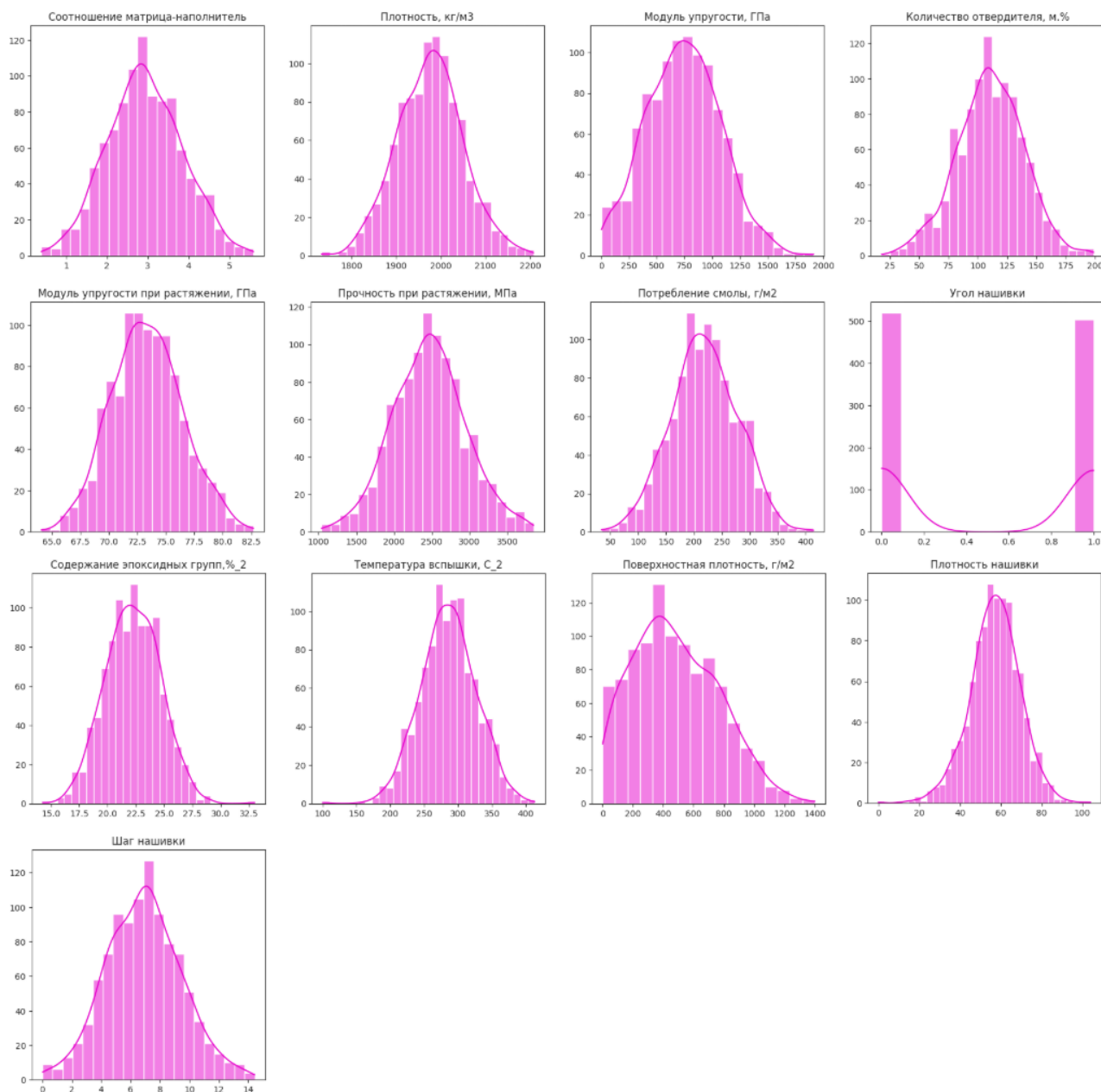


Рисунок 3 – Гистограммы распределения переменных

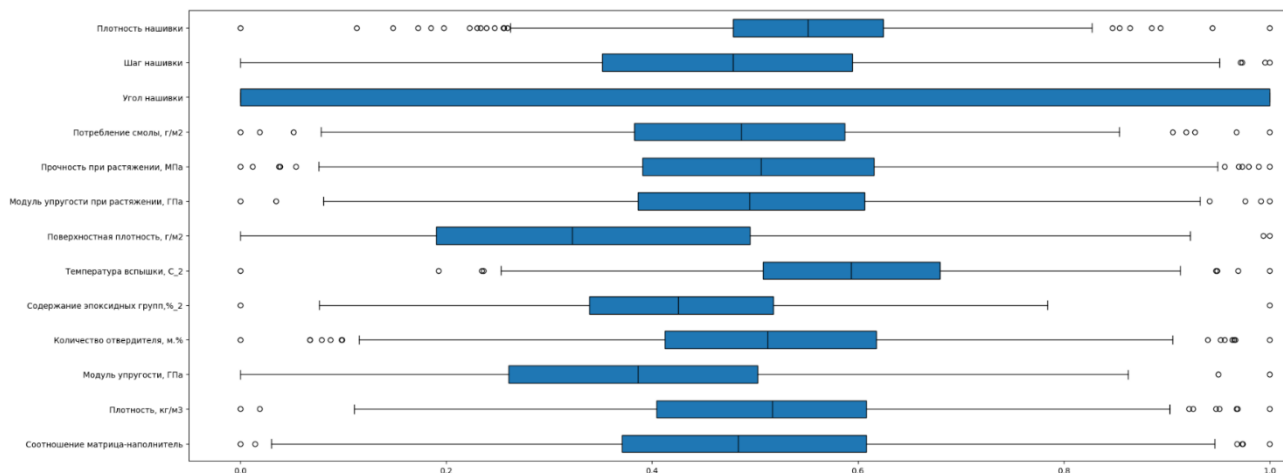


Рисунок 4 – Диаграммы «ящик с усами»

В соответствии с гистограммами распределения переменных и диаграммами «ящик с усами», представленными на рисунках 3 и 4, все признаки, за исключением ранее упомянутого «угла нашивки», имеют нормальное распределение и принимают неотрицательные значения, а также имеют значительное количество выбросов.

## 1.2 Описание используемых методов

Данная задача относится к обучению с учителем и представляет собой задачу регрессии. Основная цель алгоритмов обучения с учителем — минимизация функции потерь, что позволяет повысить точность предсказаний. В ходе исследования для решения задачи были использованы следующие методы:

- 1) линейная регрессия;
- 2) лассо-регрессия (Lasso);
- 3) метод гребневой регрессии (Ridge regression);
- 4) градиентный бустинг;
- 5) метод опорных векторов (SVM);
- 6) К-ближайших соседей (KNN);
- 7) случайный лес (Random Forest);



- 8) дерево решений (Decision Tree);
- 9) многослойный перцептрон (MLP);
- 10) нейронная сеть.

Кратко опишу принцип работы каждого из использованных методов.

### 1.2.1. Линейная регрессия

Линейная регрессия применяется для моделирования зависимости между одной входной и одной выходной переменной. В этом случае используется уравнение регрессии (1), которое задаёт прямую линию:

$$y = ax + b, \quad (1)$$

Значения коэффициентов  $a$  и  $b$ , называемых параметрами модели, определяются таким образом, чтобы минимизировать сумму квадратов разностей между фактическими значениями и предсказанными моделью. На практике их расчёт выполняется методом наименьших квадратов.

Если модель учитывает несколько входных переменных, применяется множественная линейная регрессия, описываемая уравнением (2):

$$Y = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n, \quad (2)$$

Здесь  $n$  — число входных признаков, а найденные коэффициенты формируют гиперплоскость, которая наилучшим образом приближает исходные данные. Оптимизация параметров также выполняется путём минимизации суммы квадратов отклонений наблюдений от модели.

Линейная регрессия — один из базовых методов машинного обучения. Её преимущество заключается в простоте реализации и интерпретации, что делает её отличной отправной точкой при анализе данных. Однако она накладывает жёсткие ограничения, так как предполагает линейную зависимость между переменными, что не всегда соответствует реальной картине.

Линейная регрессия отличается высокой скоростью работы и простотой реализации, что делает её удобной для первоначального анализа данных. Её модели легко интерпретировать, так как коэффициенты напрямую отражают

влияние входных переменных на результат. Кроме того, данный метод обладает низкой вычислительной сложностью по сравнению с более продвинутыми алгоритмами.

При этом линейная регрессия способна описывать только прямые зависимости, что делает её неприменимой для задач с нелинейными взаимосвязями. Также метод требует наличия чёткой связи между входными и выходными переменными, а выбросы в данных могут значительно исказить результаты, поскольку модель чувствительна к ним. Дополнительно стоит отметить, что границы, устанавливаемые линейной регрессией, всегда остаются линейными, что ограничивает её возможности в сложных многомерных задачах.

### **1.2.2. Лассо-регрессия (Lasso)**

Метод регрессии лассо (LASSO, Least Absolute Shrinkage and Selection Operator) представляет собой модификацию линейной регрессии, разработанную для работы с данными, в которых признаки обладают высокой корреляцией. Основная идея метода заключается в сжатии коэффициентов (shrinkage), что позволяет уменьшить сложность модели, изменяя пространство, в котором находятся данные. В процессе обучения LASSO автоматически устраняет или уменьшает влияние избыточных и сильно коррелированных признаков, обеспечивая более стабильные и интерпретируемые результаты. Отличительной чертой метода является использование регуляризации L1, которая штрафует модель за большие коэффициенты, заставляя некоторые из них становиться равными нулю, что фактически выполняет отбор наиболее значимых переменных.

Основным преимуществом LASSO-регрессии является её способность к автоматическому отбору признаков, что делает модель более интерпретируемой и снижает риск переобучения. Кроме того, она хорошо работает с разреженными данными, поскольку может исключать незначимые переменные. Однако метод имеет и недостатки: при наличии высокой мультиколлинеарности он может

случайно занулить один из значимых признаков, а также слабо справляется с ситуациями, когда все переменные важны, но их влияние распределено равномерно. Кроме того, для выбора оптимального коэффициента регуляризации требуется дополнительная настройка, что увеличивает вычислительные затраты.

### 1.2.3. Гребневая регрессия (Ridge regression)

Гребневая регрессия, или ридж-регрессия, представляет собой модификацию линейной регрессии, аналогичную методу LASSO. Она также использует сжатие коэффициентов и особенно эффективна при наличии сильной мультиколлинеарности в данных. Ключевое различие заключается в применяемом типе регуляризации: ридж-регрессия использует L2-норму, которая штрафует модель за большие коэффициенты, но не зануляет их полностью, а лишь делает ближе к нулю. Это позволяет сохранить все признаки в модели, но уменьшает их влияние, сглаживая возможные выбросы и снижая переобучение.

Регуляризация делает модель более устойчивой, а её коэффициенты — интерпретируемыми: чем ближе значение параметра к нулю, тем менее значим соответствующий признак.

Главным преимуществом ридж-регрессии является её способность стабилизировать модель при наличии коррелированных признаков, предотвращая чрезмерное влияние отдельных переменных. В отличие от LASSO, она не исключает признаки, что особенно полезно в случаях, когда все переменные потенциально важны.

Однако этот же аспект может быть и недостатком: если в данных присутствуют действительно незначимые признаки, ридж-регрессия не сможет их полностью удалить, что может снизить интерпретируемость модели. Кроме того, как и в случае с LASSO, требуется тщательная настройка коэффициента регуляризации для достижения оптимального баланса между переобучением и обобщающей способностью модели.

#### 1.2.4. Градиентный бустинг

Градиентный бустинг (Gradient Boosting) представляет собой ансамблевый метод, в котором модели строятся последовательно, а каждая новая модель стремится исправить ошибки предыдущей. В отличие от случайного леса, который будет описан далее, где деревья строятся независимо, в градиентном бустинге каждое новое дерево корректирует прогнозы, основываясь на остатках предыдущего. Важным элементом метода является функция потерь (loss function), которая измеряет качество предсказаний. Используя градиентный спуск, модель обновляет параметры, минимизируя значение функции потерь. При этом скорость обучения (learning rate) играет ключевую роль, определяя, насколько сильно каждое новое дерево будет корректировать предсказания.

Градиентный бустинг, основанный на деревьях решений, особенно эффективен для работы с табличными, разнородными данными и способен находить сложные нелинейные зависимости. Это один из самых мощных алгоритмов машинного обучения, который активно применяется в промышленных задачах и на соревнованиях по анализу данных. Однако он уступает нейросетям при обработке изображений, звука и других однородных данных.

Преимущество метода заключается в том, что он адаптивно исправляет ошибки предыдущих моделей, быстрее сходится к оптимальному решению и хорошо интерпретируется. Кроме того, он позволяет гибко настраивать темп обучения и параметры модели.

В то же время градиентный бустинг требует тщательного выбора критериев остановки, иначе возможен риск переобучения. Также он чувствителен к выбросам, так как строит модели на основе самых сложных наблюдений, и требует значительных вычислительных ресурсов, особенно при работе с большими данными.

### 1.2.5. Метод опорных векторов (SVR)

Метод опорных векторов (Support Vector Machine, SVM) — один из наиболее мощных и широко используемых алгоритмов машинного обучения, особенно в задачах классификации. Он строит гиперплоскость или несколько гиперплоскостей в многомерном пространстве, разделяющих объекты оптимальным образом. Ключевая идея метода заключается в выборе такой гиперплоскости, которая максимально удалена от ближайших объектов каждого класса. Эти критически важные точки называются опорными векторами и определяют границы разделения данных.

Если данные линейно неразделимы в исходном пространстве, применяется метод отображения в пространство большей размерности с использованием ядерных функций. Основные типы ядер включают линейное, полиномиальное и радиально-базисное (гауссовское, RBF). Выбор ядра и параметров модели существенно влияет на качество классификации и способность алгоритма обобщать данные.

Основное преимущество SVM — высокая эффективность при работе с небольшими, но сложными выборками. Он особенно хорош в задачах с четко выраженными границами классов и часто превосходит другие методы в условиях ограниченного количества данных.

Однако метод чувствителен к выбросам, которые могут значительно повлиять на положение разделяющей гиперплоскости. Также SVM плохо интерпретируем: сложно понять, какие признаки оказывают наибольшее влияние на предсказания модели, особенно при использовании нелинейных ядер.

### 1.2.6. Метод К-ближайших соседей (KNN)

Метод k-ближайших соседей (k-Nearest Neighbors, KNN) — это алгоритм машинного обучения, который работает по принципу запоминания обучающих данных и их последующего сравнения с новыми объектами. В отличие от

методов, строящих явную модель, KNN просто ищет среди обучающих примеров  $k$  ближайших к новому объекту и определяет его класс или значение на основе их большинства. В задачах классификации алгоритм голосует за наиболее часто встречающуюся метку, а в задачах регрессии — усредняет значения целевой переменной.

Ключевым аспектом работы KNN является выбор метрики расстояния (например, евклидовой, манхэттенской или косинусного сходства) и количества соседей  $k$ . От этих параметров зависит качество предсказаний: слишком малое  $k$  может сделать модель чувствительной к шуму, а слишком большое — привести к чрезмерному усреднению.

Главные достоинства метода — простота в реализации, интуитивно понятные результаты и низкая чувствительность к выбросам. Он не требует сложных вычислений на этапе обучения и может использоваться для различных типов данных.

Тем не менее, метод плохо масштабируется, так как при увеличении объема данных вычисления становятся затратными. Кроме того, он не создает обобщенной модели и требует хранения всех обучающих примеров, а также сильно зависит от выбора метрики расстояния, что может усложнять настройку.

### **1.2.7. Случайный лес (Random Forest)**

Случайный лес (Random Forest) — это мощный ансамблевый метод машинного обучения, который строит множество деревьев решений и объединяет их предсказания для улучшения общей точности модели. В отличие от одиночного дерева решений, случайный лес снижает вероятность переобучения за счет усреднения предсказаний нескольких деревьев, что делает его более устойчивым к шуму в данных.

Метод использует два ключевых принципа: бутстрепинг (bootstrap) и метод случайных подпространств (random subspace method). Бутстрепинг заключается в обучении каждого дерева на случайной подвыборке данных, а метод случайных

подпространств предполагает, что на каждом шаге построения дерева отбирается случайное подмножество признаков. Это помогает повысить диверсификацию моделей и снизить корреляцию между деревьями, что в конечном итоге повышает стабильность и точность предсказаний.

Случайный лес обладает рядом преимуществ: высокая точность предсказаний, устойчивость к переобучению, малочувствительность к выбросам и способность работать как с непрерывными, так и с категориальными данными. Кроме того, метод хорошо масштабируется и может эффективно использоваться на многопоточных вычислениях.

Главный недостаток этого метода — высокая вычислительная сложность, так как построение большого количества деревьев требует значительных ресурсов. Также модель теряет интерпретируемость, поскольку предсказания основываются на совокупности множества деревьев, что затрудняет анализ влияния отдельных признаков.

### **1.2.8. Дерево решений (Decision Tree)**

Деревья решений (Decision Trees) — это один из наиболее популярных и интуитивно понятных методов машинного обучения, который применяется как для задач классификации, так и для регрессии. Они представляют собой древовидную структуру, где каждый узел содержит правило, а листья определяют конечное решение. Процесс обучения дерева заключается в последовательном разбиении данных на основе определенного критерия (например, прироста информации или уменьшения дисперсии), что позволяет находить наиболее значимые признаки для предсказания.

Одним из главных достоинств деревьев решений является их высокая интерпретируемость — результаты можно представить в виде набора логических правил, которые легко понять и использовать. Кроме того, деревья не требуют сложной предварительной обработки данных, могут работать с признаками разных типов и справляются даже с пропущенными значениями.

Данный метод склонен к переобучению, особенно если дерево становится слишком глубоким, запоминая обучающие данные вместо выявления общих закономерностей. Это может привести к ухудшению качества предсказаний на новых данных. Для борьбы с этим используют методы обрезки и ограничение глубины дерева. Также стоит отметить, что деревья решений чувствительны к шуму в данных, а небольшие изменения в обучающей выборке могут сильно повлиять на итоговую структуру дерева.

### **1.2.9. Многослойный перцептрон (MLPRegressor)**

Многослойный перцептрон — это разновидность искусственной нейронной сети, которая используется для решения задач регрессии. Он обучается на наборе данных с входными и выходными значениями. Основу MLP составляют три и более слоев перцептронов: входной, один или несколько скрытых и выходной слой. Каждый нейрон в этих слоях связан с нейронами следующего слоя с помощью весов, которые корректируются в процессе обучения с помощью алгоритма обратного распространения ошибки.

Главное преимущество MLP — его способность моделировать сложные нелинейные зависимости между входами и выходами, что делает его универсальным инструментом для прогнозирования. Алгоритм может обобщать входные данные и не требует строгих предположений о распределении признаков.

Однако у метода есть и недостатки: процесс обучения может быть долгим и требовательным к вычислительным ресурсам, особенно при наличии большого количества скрытых слоев. Кроме того, MLP чувствителен к выбору гиперпараметров, таких как количество слоев, нейронов и скорость обучения, а также к масштабу входных данных, что делает его настройку нетривиальной задачей. Также существует риск застревания в локальных минимумах из-за невыпуклой функции потерь.



### 1.2.10. Нейронная сеть

Нейронная сеть — это мощный инструмент машинного обучения, имитирующий работу биологических нейронов. Ее главная цель — выявлять сложные зависимости в данных и строить прогнозы на их основе.

Ключевые компоненты нейронной сети:

1. Нейрон (персептрон) — вычислительный элемент сети, который принимает сигналы, суммирует их с учетом весов и передает результат через функцию активации;
2. Вес (weight) — коэффициент, определяющий значимость входного сигнала;
3. Смещение (bias) — дополнительный параметр, корректирующий результат вычислений;
4. Функция активации — нелинейное преобразование суммарного сигнала (например, ReLU, tanh);
5. Слои нейронной сети:
  - а) Входной слой — принимает исходные данные;
  - б) Скрытые слои — выполняют вычисления и выделяют закономерности;
  - в) Выходной слой — формирует итоговое предсказание;

Процесс обучения нейросети:

1. Прямое распространение (forward propagation) — входные данные проходят через сеть, и на выходе получается предсказание.
2. Функция потерь (loss function) — измеряет, насколько предсказание отличается от реального значения.
3. Обратное распространение ошибки — вычисляет градиенты ошибок и корректирует веса с помощью градиентного спуска или других оптимизаторов (Adam, SGD).
4. Обновление весов — корректировка параметров для минимизации функции потерь.

5. Эпохи (epochs) — количество полных циклов обучения на всем наборе данных.

Преимуществами нейронных сетей являются их способность моделировать сложные нелинейные зависимости, высокая гибкость архитектуры, возможность работы с различными типами данных (изображения, текст, звук, временные ряды) и способность выявлять скрытые закономерности в данных. Они широко применяются в компьютерном зрении, обработке естественного языка, медицинской диагностике и других областях, требующих сложного анализа данных.

К недостаткам нейронных сетей можно отнести высокие вычислительные затраты, необходимость большого количества данных для обучения, сложность подбора гиперпараметров и длительное время обучения. Кроме того, модели плохо интерпретируемы, что делает их менее прозрачными и сложными для объяснения, особенно в критически важных областях, таких как медицина или финансы.

### **1.3. Разведочный анализ данных**

Разведочный анализ данных направлен на получение информации о распределении переменных в исходном датасете, оценку его качества (выявление пропусков и выбросов), а также изучение взаимосвязей между признаками, что позволит сформировать обоснованные гипотезы о наиболее подходящих моделях машинного обучения для решения поставленной задачи.

В первую очередь необходимо изучить структуру данных, выявить возможные пропущенные значения, выбросы и дубликаты. В нашем датасете дубликатов и пропущенных значений не наблюдается.

```
# проверка на наличие дубликатов
dpl = df_X.duplicated().sum()
if dpl == 0:
    print('Дубликатов нет')
else:
    print(f'Количество дубликатов в датасете: {dpl}')

Дубликатов нет
```

Рисунок 5 – Проверка на наличие дубликатов

```
df_X.isnull().sum()
```

	0
Соотношение матрица-наполнитель	0
Плотность, кг/м3	0
модуль упругости, ГПа	0
Количество отвердителя, м.%	0
Содержание эпоксидных групп, %_2	0
Температура вспышки, С_2	0
Поверхностная плотность, г/м2	0
Модуль упругости при растяжении, ГПа	0
Прочность при растяжении, МПа	0
Потребление смолы, г/м2	0
Угол нашивки, град	0
Шаг нашивки	0
Плотность нашивки	0

dtype: int64

Рисунок 6 – Проверка на наличие пропусков

Для проведения разведочного анализа данных были использованы расчет статистических характеристик датасета, построение гистограмм для анализа распределения переменных, диаграммы "ящик с усами" для выявления выбросов, попарные графики рассеяния для изучения взаимосвязей между переменными, тепловая карта корреляций, а также вычисление описательной статистики для каждого признака.

Выбросы были выявлены через применение двух методов – метода трех сигм и метода межквартильного размаха.

Первый метод основан на предположении, что данные распределены нормально. Он вычисляет среднее значение и стандартное отклонение для

каждого признака. Затем значения, которые выходят за пределы трёх стандартных отклонений от среднего (в обе стороны), считаются выбросами.

Формула данного метода:

$$z = \frac{x - \mu}{\sigma}, \quad (3)$$

где:

$\mu$  — среднее значение признака,

$\sigma$  — стандартное отклонение,

Если  $|z| > 3$ , то значение считается выбросом.

Метод межквартильного размаха не предполагает нормального распределения данных, а вместо этого использует квартили. В данном методе сначала вычисляется первый квартиль ( $Q1$ ) и третий квартиль ( $Q3$ ), затем находится межквартильный размах  $IQR = Q3 - Q1$ .

Выбросами считаются значения, выходящие за границы:  $Q1 - 1,5 \times IQR$  и  $Q3 + 1,5 \times IQR$ .

После применения обоих методов на исследуемом датасете, больше выбросов было выявлено с помощью метода межквартильного размаха. По этой причине было принято решение использовать указанный метод в качестве основного (рисунок 7).

После трех итераций удалось полностью исключить выбросы из датасета (рисунок 8).

Анализ объединённого датасета показывает отсутствие выраженной зависимости между переменными, что подтверждается тепловой картой корреляционной матрицы и диаграммами рассеяния.

```

# метод 3-х сигм
def outliers_z_score(df_X):
    outliers = {}
    for column in df_X.columns:
        mean = df_X[column].mean()
        std_dev = df_X[column].std()
        z_scores = (df_X[column] - mean) / std_dev
        outliers[column] = len(z_scores[np.abs(z_scores) > 3])
    return outliers

# метод межквартильных расстояний (IQR)
def outliers_iqr(df_X):
    outliers = {}
    for column in df_X.columns:
        q1 = df_X[column].quantile(0.25)
        q3 = df_X[column].quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr
        outliers[column] = len(df_X[(df_X[column] < lower_bound) | (df_X[column] > upper_bound)])
    return outliers

outliers_z_score = outliers_z_score(df_X)
outliers_iqr = outliers_iqr(df_X)

comparison_df = pd.DataFrame({
    'Метод 3-х сигм': outliers_z_score,
    'Метод IQR': outliers_iqr
})

print("Результаты выявления выбросов:")
print(comparison_df)

```

Результаты выявления выбросов:

	Метод 3-х сигм	Метод IQR
Соотношение матрица-наполнитель	0	6
Плотность, кг/м3	3	9
Модуль упругости, ГПа	2	2
Количество отвердителя, м.%	2	14
Содержание эпоксидных групп, %_2	2	2
Температура вспышки, C_2	3	8
Поверхностная плотность, г/м2	2	2
Модуль упругости при растяжении, ГПа	0	6
Прочность при растяжении, МПа	0	11
Потребление смолы, г/м2	3	8
Угол нашивки	0	0
Шаг нашивки	0	4
Плотность нашивки	7	21

Рисунок 7 – Применение двух методов определения количества выбросов

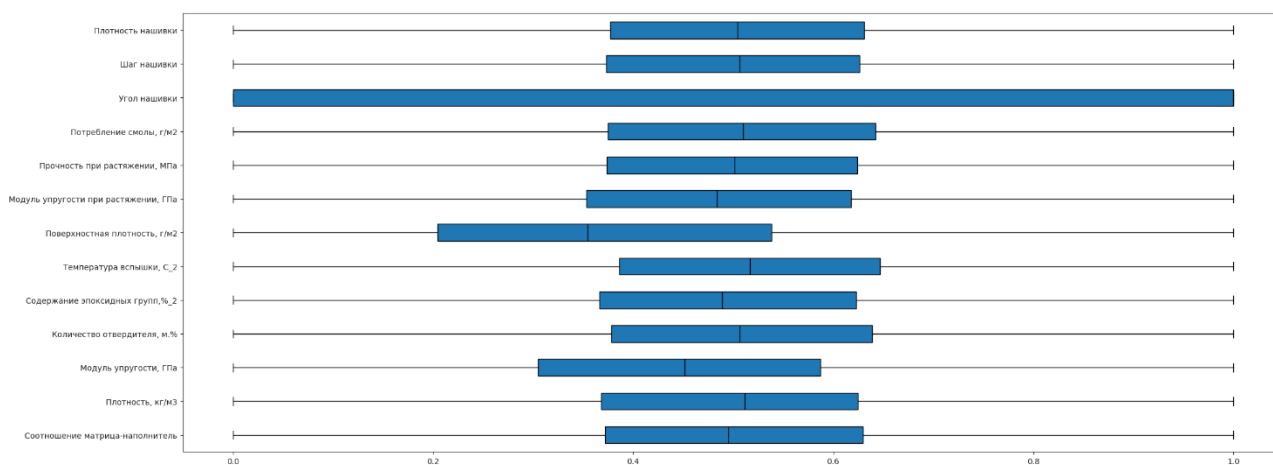


Рисунок 8 – «Ящик с усами» для признаков после удаления выбросов

Наибольшая корреляция наблюдается между плотностью нашивки и углом нашивки, однако её значение составляет всего 0,108, что указывает на отсутствие значимой связи (рисунок 9).

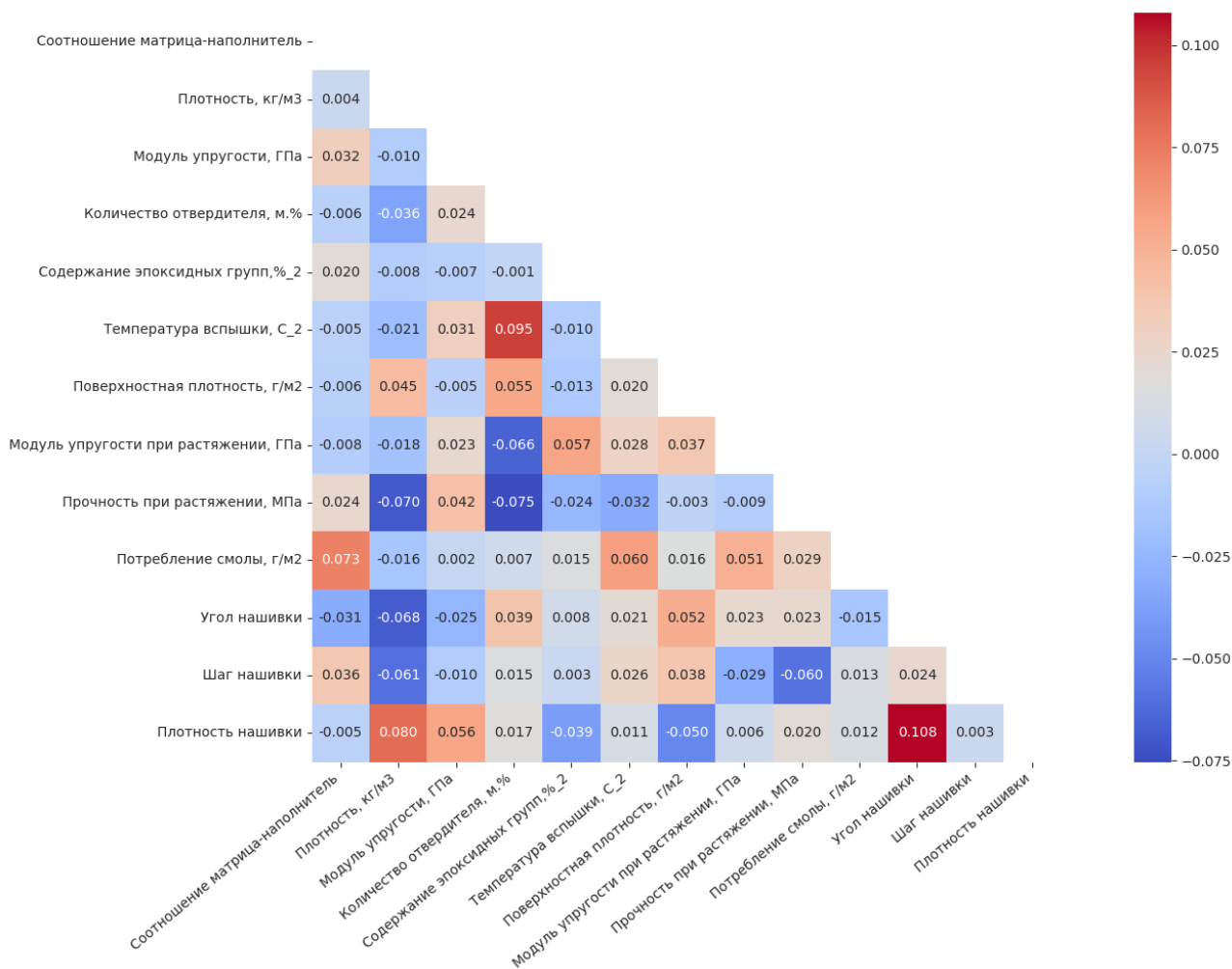


Рисунок 9 – Тепловая карта корреляционной матрицы

Анализ данных показал отсутствие линейных зависимостей между переменными, что подтверждается низкими значениями коэффициентов корреляции, то есть статистический анализ не выявил значимых взаимосвязей между параметрами.

С учетом характеристик исходных данных мы имеем три целевых признака которые выражены следующим образом: модуль упругости при растяжении и прочность при растяжении зависят от свойств матрицы, наполнителя и параметров процесса, а соотношение матрица-наполнитель определяется характеристиками матрицы, наполнителя и свойствами конечного композита. В

связи с этим необходимо построить предсказывающие модели для первых двух признаков и разработать нейронную сеть для прогнозирования последнего в соответствии с заданием.

## 2. Практическая часть

### 2.1. Препроцессинг данных

С учетом характеристик исходных данных мы имеем три целевых признака, которые в нашей задаче будут определены как выходные данные: модуль упругости при растяжении и прочность при растяжении, а также соотношение матрица-наполнитель, предсказание которого будет осуществляться с применением нейронных сетей.

Перед передачей данных в модель требуется их предварительная обработка – препроцессинг данных. Признак угла нашивки принимает дискретные значения, поэтому для его кодирования используется `OrdinalEncoder`. Числовые переменные подвергаются стандартизации с применением `StandardScaler`, который нормализует данные, приводя их к среднему значению 0 и стандартному отклонению 1, что способствует лучшей сходимости модели.

```
# для признака "Модуль упругости при растяжении"
y1_columns = ['Модуль упругости при растяжении, ГПа']
x1_columns = [col for col in df.columns if col not in y1_columns]

y1 = df.loc[:, y1_columns]
x1 = df.loc[:, x1_columns]

# для признака "Прочность при растяжении"
y2_columns = ['Прочность при растяжении, МПа']
x2_columns = [col for col in df.columns if col not in y2_columns]

y2 = df.loc[:, y2_columns]
x2 = df.loc[:, x2_columns]

# для признака "Соотношение матрица-наполнитель"
y3_columns = ['Соотношение матрица-наполнитель']
x3_columns = [col for col in df.columns if col not in y3_columns]

y3 = df.loc[:, y3_columns]
x3 = df.loc[:, x3_columns]
```

Рисунок 10 – Деление датасетов по целевому признаку для каждой из задач

```
# определяем категориальные и числовые признаки

categorical_feature = ['Угол нашивки']
num_features_x1 = list(set(x1_columns) - set(categorical_feature))
num_features_x2 = list(set(x2_columns) - set(categorical_feature))
num_features_x3 = list(set(x3_columns) - set(categorical_feature))

# создаем препроцессоры для разных задач

preproc_1 = ColumnTransformer(
    transformers=[
        ("scale_numeric", StandardScaler(), num_features_x1),
        ("encode_categorical", OrdinalEncoder(), categorical_feature)
    ]
)

preproc_2 = ColumnTransformer(
    transformers=[
        ("scale_numeric", StandardScaler(), num_features_x2),
        ("encode_categorical", OrdinalEncoder(), categorical_feature)
    ]
)

preproc_3 = ColumnTransformer(
    transformers=[
        ("scale_numeric", StandardScaler(), num_features_x3),
        ("encode_categorical", OrdinalEncoder(), categorical_feature)
    ]
)
```

Рисунок 11 – Создание препроцессоров

## 2.2. Выбор и обучение моделей

Для прогноза параметров модуля упругости при растяжении и прочности при растяжении будут использованы: метод линейной регрессии, метод лассо-регрессии, метод гребневой регрессии (Ridge regression), метод градиентного бустинга, метод опорных векторов, метод К-ближайших соседей, метод случайного леса, метод дерева решений (рисунок 12).

```
models = {
    'Dummy Regressor': DummyRegressor(strategy='mean'),
    'Linear Regression': LinearRegression(),
    'Ridge': Ridge(),
    'Lasso': Lasso(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'SVR': SVR(),
    'KNN': KNeighborsRegressor(),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Random Forest': RandomForestRegressor(random_state=42)
}
```

Рисунок 12 – Перечень используемых моделей для двух первых задач



Порядок разработки моделей для указанных двух целевых параметров следующий: разделение данных на обучающую и тестовую выборки в соотношении 70% на 30%, препроцессинг входных данных (выходные данные преобразую в массив), проверка моделей со стандартными гиперпараметрами, сравнение с результатами модели, выдающей среднее значение – Dummy Regressor со стратегией «mean» (далее – базовая модель), визуализация сравнения результатов моделей через таблицу, где зеленым выделены наибольшие результаты по коэффициенту детерминации и наименьшие результаты по метрикам ошибки RMSE (среднеквадратичная ошибка), MAE (средняя абсолютная ошибка) и MAPE (средняя абсолютная процентная ошибка).

### 2.2.1. Прогноз параметра модуля упругости при растяжении

Лучшими дефолтными моделями для модуля упругости при растяжении показали себя базовая модель и метод регрессии лассо.

```
# разделяем выборки
x1_train_initial, x1_test_initial, y1_train, y1_test = train_test_split(x1, y1, test_size=0.3, random_state=42)

# преобразуем целевой параметр в массив
y1_train = y1_train['Модуль упругости при растяжении, ГПа'].values
y1_test = y1_test['Модуль упругости при растяжении, ГПа'].values

# препроцессинг
x1_train = preproc_1.fit_transform(x1_train_initial)
x1_test = preproc_1.transform(x1_test_initial)

results_1 = evaluate_models(models, x1_train, y1_train)
styled_results_1 = style_model_results(results_1)
styled_results_1
```

	R2	RMSE	MAE	MAPE
Dummy Regressor	-0.005978	2.959776	2.391135	0.032699
Linear Regression	-0.026962	2.989545	2.427342	0.033198
Ridge	-0.026884	2.989433	2.427215	0.033197
Lasso	-0.005978	2.959776	2.391135	0.032699
Gradient Boosting	-0.134035	3.138977	2.531864	0.034632
SVR	-0.060587	3.036800	2.468571	0.033731
KNN	-0.220369	3.253134	2.637365	0.036116
Decision Tree	-1.259385	4.419100	3.621063	0.049494
Random Forest	-0.075859	3.060216	2.479546	0.033924

Рисунок 13 – Отработка моделей со стандартными гиперпараметрами

Все протестированные модели продемонстрировали неудовлетворительные результаты для наших данных. Метод дерева решений с параметрами по умолчанию отработал хуже всех.

Далее я начал перебирать различные гиперпараметры для каждой из моделей с помощью поиска по сетке с перекрестной проверкой (рисунки 14-15).

```
def grid_search(model, params, x, y):
    pd.options.display.max_colwidth = 100 # чтобы полностью отобразить оптимальные параметры при выводе

    results = pd.DataFrame()

    cv = KFold(10, shuffle=True, random_state=42)

    scoring = 'neg_root_mean_squared_error'
    searcher = GridSearchCV(model, params, cv=cv, scoring=scoring)

    searcher.fit(x, y)

    results.loc[:, 'best parameters'] = searcher.cv_results_['params']
    results.loc[:, 'RMSE'] = -searcher.cv_results_['mean_test_score']
    results.loc[:, 'rank'] = searcher.cv_results_['rank_test_score']

    return results, searcher.best_estimator_
```

Рисунок 14 – Функция для поиска оптимальных параметров моделей

```
# создаем dict с лучшими параметрами моделей
GS_best_models_1 = {}

# для обычной линейной регрессии нет возможности для перебора параметров, поэтому ее в данном случае не рассматриваем
# лучшие параметры для модели Ridge

params_1 = {
    'alpha': range(1, 10**6, 5000),
    'fit_intercept': [True, False],
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sag', 'saga']
}

search, best_model = grid_search(Ridge(), params_1, x1_train, y1_train)

# сохранение лучшей модели в словарь
GS_best_models_1[str(best_model)] = best_model

# вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```

	best parameters	RMSE
2393	{'alpha': 995001, 'fit_intercept': True, 'solver': 'saga'}	2.959781

Рисунок 15 – Пример поиска лучших гиперпараметров

На выходе получаю таблицу с параметрами (рисунок 16), обеспечивающими лучшие результаты по каждой из моделей и соответствующими метриками (коэффициент детерминации, RMSE, MAE и MAPE).

	R2	RMSE	MAE	MAPE
Ridge(alpha=995001, solver='saga')	-0.005982	2.959782	2.391148	0.032699
Lasso(alpha=1)	-0.005978	2.959776	2.391135	0.032699
GradientBoostingRegressor(learning_rate=0.05, max_depth=4, n_estimators=5, subsample=0.5)	-0.011230	2.967679	2.400278	0.032821
SVR(C=0.001)	-0.008681	2.963760	2.398581	0.032777
KNeighborsRegressor(metric='chebyshev', n_neighbors=9)	-0.077981	3.057479	2.459564	0.033687
DecisionTreeRegressor(max_depth=3, min_samples_leaf=10, min_samples_split=5)	-0.083898	3.073318	2.469612	0.033780
RandomForestRegressor(max_depth=3, min_samples_leaf=2, min_samples_split=7, n_estimators=10)	-0.036442	3.005079	2.435259	0.033320

Рисунок 16 – Метрики моделей для модуля упругости при растяжении

Таким образом, подбор гиперпараметров способен существенно повысить точность прогнозирования выбранной модели. Однако все протестированные модели продемонстрировали низкую эффективность в описании исходных данных, мне не удалось добиться даже положительных значений коэффициента детерминации.

Даже лучшая из моделей показала коэффициент детерминации, близкий к нулю, что соответствует результату хуже работы базовой модели. Линейные алгоритмы дали результат на уровне базовой модели (рисунок 17).

```
best_model_1 = Lasso(
    alpha = 1
)

best_model_1.fit(x1_train, y1_train)
y1_best = best_model_1.predict(x1_test)

base_model_1 = DummyRegressor(strategy='mean')
base_model_1.fit(x1_train, y1_train)
y1_dummy_predicted = base_model_1.predict(x1_test)

diff_stats_1 = calculate_metrics('Базовая модель', y1_test, y1_dummy_predicted)
diff_stats_1 = pd.concat([diff_stats_1, calculate_metrics('Лучшая модель (Lasso)', y1_test, y1_best)], ignore_index=False)
diff_stats_1
```

	R2	RMSE	MAE	MAPE
Базовая модель	-0.008879	3.162017	2.580193	0.035032
Лучшая модель (Lasso)	-0.008879	3.162017	2.580193	0.035032

Рисунок 17 – Вывод результатов лучшей и базовой моделей для тестовой выборки

## 2.2.2. Прогноз параметра прочности при растяжении

Для данного параметра я взял аналогичные модели в надежде получить более приемлемые результаты.

	R2	RMSE	MAE	MAPE
Dummy Regressor	-0.008783	448.544529	358.490113	0.155329
Linear Regression	-0.038337	455.122689	364.421774	0.157442
Ridge	-0.038218	455.096808	364.398516	0.157433
Lasso	-0.035833	454.589956	363.969122	0.157269
Gradient Boosting	-0.082861	464.955061	374.226187	0.161506
SVR	-0.008824	448.566744	358.793042	0.155192
KNN	-0.165694	481.601403	381.265105	0.165831
Decision Tree	-1.083707	642.290741	519.203017	0.220558
Random Forest	-0.060426	460.212136	369.131466	0.159783

Рисунок 18 – Метрики дефолтных моделей

Базовой моделью тоже взял Dummy Regressor со стратегией «mean». У линейных моделей и метода опорных векторов коэффициент детерминации оказался близким к нулю, но это все равно свидетельствует об их неэффективности по сравнению с базовой моделью. Дерево решений с параметрами по умолчанию, как и в первой задаче, продемонстрировал еще более слабые результаты по сравнению с линейными моделями.

По факту подбора оптимальных гиперпараметров лучше всех показал себя градиентный бустинг, при этом метод опорных векторов тоже показал одни из наименьших значений ошибок, близких к аналогичным значениям у линейных моделей (рисунок 19).

	R2	RMSE	MAE	MAPE
Ridge(alpha=990001, solver='sag')	-0.008786	448.545175	358.490976	0.155329
Lasso(alpha=1)	-0.035833	454.589956	363.969122	0.157269
GradientBoostingRegressor(learning_rate=0.05, max_depth=5, min_samples_split=5, n_estimators=5, subsample=0.5)	-0.005549	448.010564	357.749245	0.155125
SVR(C=0.001, kernel='poly')	-0.008242	448.422906	358.590912	0.155206
KNeighborsRegressor(metric='manhattan', n_neighbors=9, weights='distance')	-0.073862	462.713249	368.924407	0.160108
DecisionTreeRegressor(criterion='absolute_error', max_depth=3)	-0.037244	454.614407	360.615842	0.155767
RandomForestRegressor(criterion='absolute_error', max_depth=6, min_samples_leaf=5, min_samples_split=5, n_estimators=10)	-0.039761	455.404062	364.549481	0.157101

Рисунок 19 – Метрики моделей для модуля упругости при растяжении

Метрики работы наиболее успешной модели на тестовом наборе данных (рисунок 20) указывают на то, что градиентный бустинг имеет небольшое преимущество перед базовой моделью, хотя общий результат исследования остается печальным. Учитывая все факторы, у меня не вышло получить модель, которая качественно объясняет исследуемые параметры.

	R2	RMSE	MAE	MAPE
Базовая модель	-0.000057	462.664655	368.866939	0.161879
Лучшая модель (GradientBoostingRegressor)	0.000667	462.497121	368.144788	0.161549

Рисунок 20 – Результаты лучшей и базовой моделей для тестовой выборки

### 2.3. Прогноз параметра соотношение «матрица-наполнитель»

Согласно заданию выпускной квалификационной работы необходимо написать нейронную сеть, которая будет рекомендовать соотношение матрица-наполнитель. Помимо высокоуровневого API tensorflow.keras для создания и обучения нейросетей я сначала использую многослойный перцептрон (MLPRegressor) из библиотеки scikit-learn, поскольку он может себя хорошо показать в регрессионных задачах с нелинейной зависимостью между переменными.

#### 2.3.1. Применение MLPRegressor (библиотека sklearn)

В нашем случае модель MLPRegressor представляет собой глубокую нейросеть с восьмью слоями, содержащими 64, 64, 32, 32, 16, 16, 8 и 8 нейронов соответственно (рисунок 21). В качестве функции активации используется ReLU.

Оптимизация весов осуществляется с помощью алгоритма Adam, который адаптивно подстраивает скорость обучения. Максимальное число итераций обучения установлено в 1000, но благодаря механизму ранней остановки (early\_stopping=True) процесс может завершиться раньше, если качество модели на валидационной выборке перестанет улучшаться. Дополнительно применяется L2-регуляризация (alpha = 0.01), что снижает вероятность переобучения.

```
# модель многослойного перцептрона
mlp = MLPRegressor(
    hidden_layer_sizes = (64, 64, 32, 32, 16, 16, 8, 8),
    activation = 'relu',
    solver='adam',
    max_iter=1000,
    early_stopping = True,
    validation_fraction = 0.3,
    alpha = 0.01,
    random_state=42,
    verbose=True
)

mlp.fit(x3_train, y3_train)
```

Рисунок 21 – Создание архитектуры нейронной сети на основе MLPRegressor

Модель многослойного перцептрона была обучена и протестирована на имеющихся данных. Визуальный анализ результатов прогнозирования и метрики (рисунки 22-23) показывают, что предсказания модели сглажены и не улавливают колебания фактических значений, аналогично показывая результаты хуже, чем простая базовая модель. Это может свидетельствовать о недостаточной гибкости модели или о необходимости более глубокой настройки гиперпараметров.

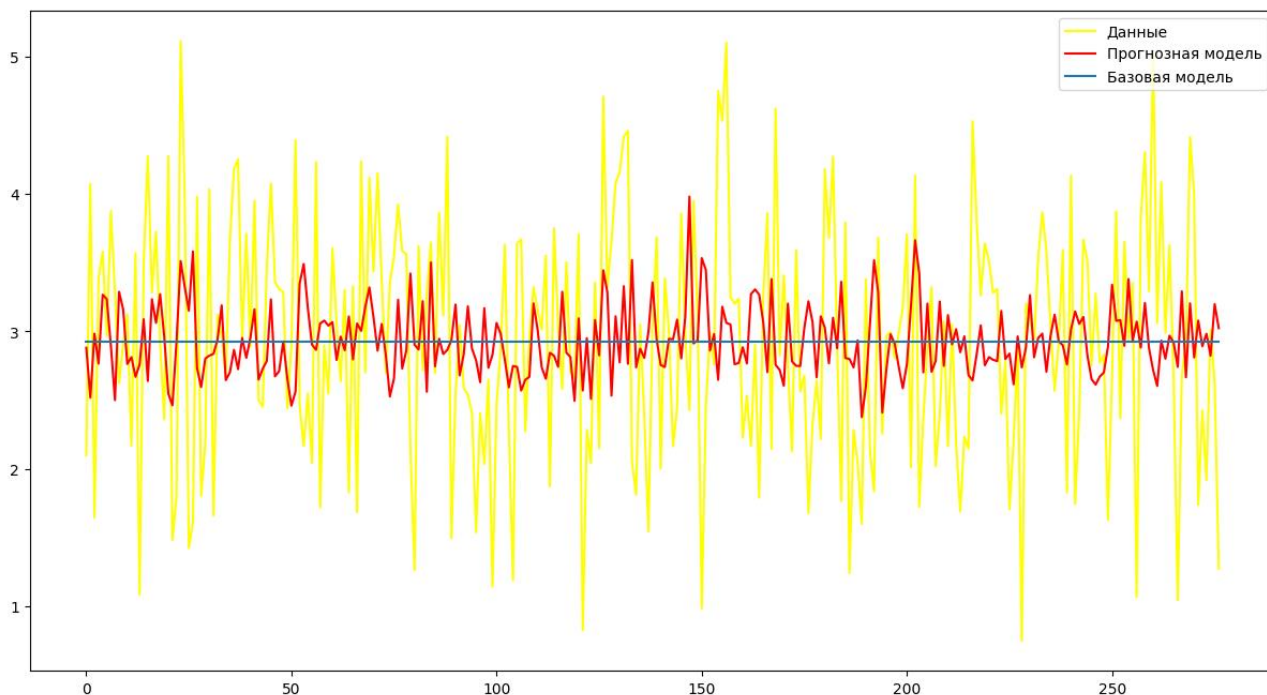


Рисунок 22 – Результат работы модели многослойного перцептрона

	R2	RMSE	MAE	MAPE
Dummy Regressor	-0.000080	0.874368	0.710610	0.303820
MLPRegressor	-0.112427	0.922174	0.762342	0.322394

Рисунок 23 – Метрики работы MLPRegressor и базовой модели

### 2.3.2. Применение нейронной сети tensorflow.keras

Входной слой содержит 12 нейронов, соответствующих количеству признаков (фичей) в данных. Архитектура аналогично включает восемь скрытых полносвязных слоев с функцией активации ReLU (рисунок 24). Выходной слой состоит из одного нейрона без функции активации, так как модель предназначена для решения задачи регрессии.



```
# создаем аналогичную архитектуру нейросети

def keras_model():
    return tf.keras.Sequential([
        keras.layers.Input(shape=(12,), name='in'),           # 12 признаков
        keras.layers.Dense(64, activation='relu', name='dense_1'),
        keras.layers.Dense(64, activation='relu', name='dense_2'),
        keras.layers.Dense(32, activation='relu', name='dense_3'),
        keras.layers.Dense(32, activation='relu', name='dense_4'),
        keras.layers.Dense(16, activation='relu', name='dense_5'),
        keras.layers.Dense(16, activation='relu', name='dense_6'),
        keras.layers.Dense(8, activation='relu', name='dense_7'),
        keras.layers.Dense(8, activation='relu', name='dense_8'),
        keras.layers.Dense(1, name='out')
    ])

```

Рисунок 24 – Архитектура нейронной сети tensorflow.keras

Обучение нейросети происходило без дополнительных гиперпараметров, количество эпох – 100, с разбивкой 70% на 30%, где меньшая часть – валидационная (рисунок 25).

```
model_NN = keras_model()

model_NN = compile_model(model_NN) # компиляция нейросети

model_NN.summary()

Model: "sequential"

```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 64)	832
dense_2 (Dense)	(None, 64)	4,160
dense_3 (Dense)	(None, 32)	2,080
dense_4 (Dense)	(None, 32)	1,056
dense_5 (Dense)	(None, 16)	528
dense_6 (Dense)	(None, 16)	272
dense_7 (Dense)	(None, 8)	136
dense_8 (Dense)	(None, 8)	72
out (Dense)	(None, 1)	9

```

Total params: 9,145 (35.72 KB)
Trainable params: 9,145 (35.72 KB)
Non-trainable params: 0 (0.00 B)

model_NN_hist = model_NN.fit(
    x3_train,
    y3_train,
    epochs = 100,
    validation_split = 0.3,
    verbose = 1)

```

Рисунок 25 – Обучение модели tensorflow.keras



Метрики loss работы нейросети отображены на графике 26. Можно заметить, что буквально с самого начала обучения потери на валидационной выборке не уменьшались, что свидетельствует о переобучении и несостоятельности модели.

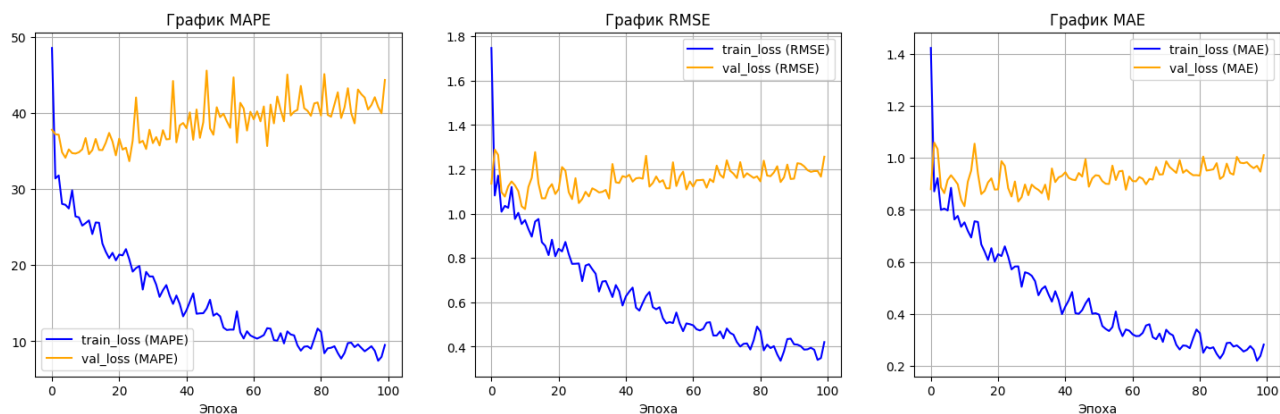


Рисунок 26 – Графики loss нейронной сети

Я постарался улучшить модель посредством внедрения нескольких методов оптимизации. В частности, был добавлен механизм обратного вызова (callback), который контролирует процесс обучения и предотвращать переобучение за счет ранней остановки или динамической корректировки гиперпараметров. Также был использован метод dropout, способствующий регуляризации модели, отключая случайные нейроны во время обучения и тем самым уменьшая вероятность переобучения.

Дополнительно был уменьшен коэффициент обучения (learning rate) с 0.01 до 0.001, что позволило модели находить оптимальные веса более эффективно, избегая слишком больших скачков. Наконец, для стабилизации процесса обучения была применена пакетная нормализация (Batch Normalization), которая выполняет нормализацию входных данных каждого слоя, ускоряя обучение и снижая влияние изменения параметров в предыдущих слоях. Новая архитектура со всеми модификациями изображена на рисунке 27.

```

from keras.layers import BatchNormalization
from keras.callbacks import EarlyStopping

def keras_model_2():
    return keras.Sequential([
        keras.layers.Input(shape=(12,), name='in'),

        keras.layers.Dense(64, name='dense_1'),
        keras.layers.BatchNormalization(),
        keras.layers.Activation('relu'),
        keras.layers.Dropout(0.05, name='dropout_1'),

        keras.layers.Dense(32, name='dense_2'),
        keras.layers.BatchNormalization(),
        keras.layers.Activation('relu'),
        keras.layers.Dropout(0.05, name='dropout_2'),

        keras.layers.Dense(16, name='dense_3'),
        keras.layers.BatchNormalization(),
        keras.layers.Activation('relu'),
        keras.layers.Dropout(0.05, name='dropout_3'),

        keras.layers.Dense(8, name='dense_4'),
        keras.layers.BatchNormalization(),
        keras.layers.Activation('relu'),
        keras.layers.Dropout(0.05, name='dropout_4'),

        keras.layers.Dense(1, name='out')
    ])
  
```

Рисунок 27 – Модифицированная архитектура нейронной сети

Благодаря изменению получилось значительно снизить потери модели и разницу результатов между тестовой и валидационной выборкой (график 28).

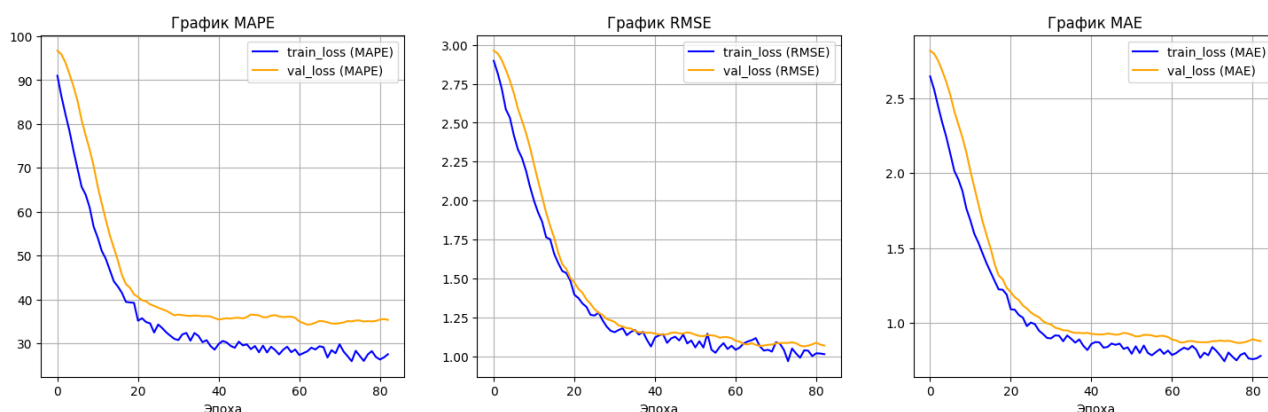


Рисунок 28 – Графики loss модифицированной нейронной сети

Сравнивая итоговые результаты вариантов нейронных сетей на tensorflow.keras можно заметить насколько сильно модифицированная

нейронная сеть лучше первоначальной нейросети (график 29). Тем не менее, обе модели нейронных сетей все так же хуже в сравнении с базовой моделью.

	R2	RMSE	MAE	MAPE
Dummy Regressor	-0.000080	0.874368	0.710610	0.303820
Обученная нейросеть	-1.100811	1.267274	1.005038	0.421054
Нейросеть модифицированная	-0.569712	1.095436	0.900044	0.327650

Рисунок 29 – Метрики работы нейронных сетей на tensorflow.keras

## 2.4. Оценка точности моделей на тренировочном и тестовом выборках

Лучшие модели с настроенными гиперпараметрами в каждой из трех задач были отработаны на тестовой выборке, результат продемонстрирован на рисунке 30.

	R2	RMSE	MAE	MAPE
Train по параметру Модуль упругости при растяжении	0.0	2.964995	2.388992	0.032669
Test по параметру Модуль упругости при растяжении	-0.008879	3.162017	2.580193	0.035032
Train по параметру Прочность при растяжении	0.056229	436.433977	348.368369	0.151018
Test по параметру Прочность при растяжении	0.007321	460.954781	367.431678	0.161232
Train по параметру Соотношение матрица-наполнитель	-0.09619	0.946166	0.721582	0.256204
Test по параметру Соотношение матрица-наполнитель	-0.557135	1.091039	0.898427	0.330539

Рисунок 30 – Сравнение метрик лучших моделей на тренировочной/тестовой выборках

Чтобы лучше сопоставить полезность лучших моделей, необходимо оценить диапазон ошибки, то есть значение максимальной ошибки (рисунок 31) и сравнить ее с разницей между медианным и нижнепороговым значениями (рисунок 32).

	max_error
Test по параметру Модуль упругости при растяжении	7.960885
Test по параметру Прочность при растяжении	1228.996390
Test по параметру Соотношение матрица-наполнитель	2.872174

Рисунок 31 – Значения максимальных ошибок лучших моделей на тестовых выборках

	min	max
Соотношение матрица-наполнитель	0.547391	5.314144
Модуль упругости при растяжении, ГПа	65.793845	81.203147
Прочность при растяжении, МПа	1250.392802	3654.434359

Рисунок 32 – Минимальные и максимальные значения целевых переменных

Таким образом, в первой задаче для прогнозирования модуля упругости при растяжении модель лассо-регрессии не смогла уловить зависимости во входных данных. Ошибка на тестовой выборке чуть больше, чем на тренировочной, но даже на обучающем наборе модель не смогла чему-то научиться. Значения переменной модуля упругости при растяжении лежит в диапазоне от 65.8 ГПа до 81.2 ГПа (амплитуда 15.4 ГПа), в то время как максимальная ошибка в 7.96 ГПа превышает половину амплитуды значений переменной, что на практике подтверждает бесполезность модели в предсказании.

Во второй задаче модель градиентного бустинга показала небольшой положительный коэффициент детерминации, что уже лучше. Ошибки на тесте и обучении близки, значит, модель нашла какие-то следы зависимости, а не просто выучила данные. Но точность всё ещё слишком низкая, и решить задачу не удалось. Максимальная ошибка в данном случае аналогично превышает половину амплитуды значений прочности при растяжении ( $1228 > 1202$ ).

Третья модель с нейронными сетями на tensorflow.keras продемонстрировала заметное ухудшение показателей на тестовых данных по сравнению с обучением. По сути модель просто запомнила данные. В данном случае максимальная ошибка так же превышает половину амплитуды значений

переменной ( $2.87 > 2.39$ ), что свидетельствует о неудовлетворительных результатах решения данной задачи.

## 2.5. Создание веб-приложения

Несмотря на плохую работу моделей, попробуй взять одну из них и на ее основе создать веб-приложение, выдающее значение «соотношения матрица-наполнитель». В качестве модели возьму вторую нейронную сеть, которая была дополнительно настроена и которая добилась лучших результатов относительно базовой нейронной сети на tensorflow.keras.

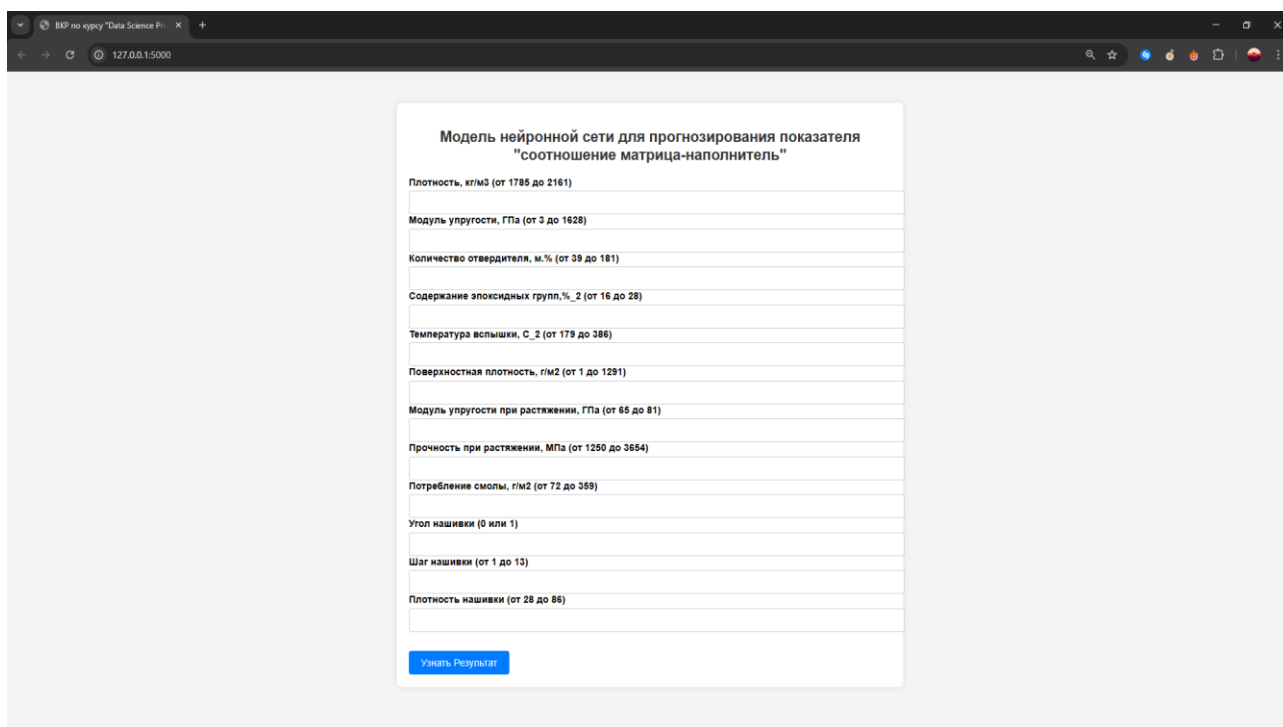
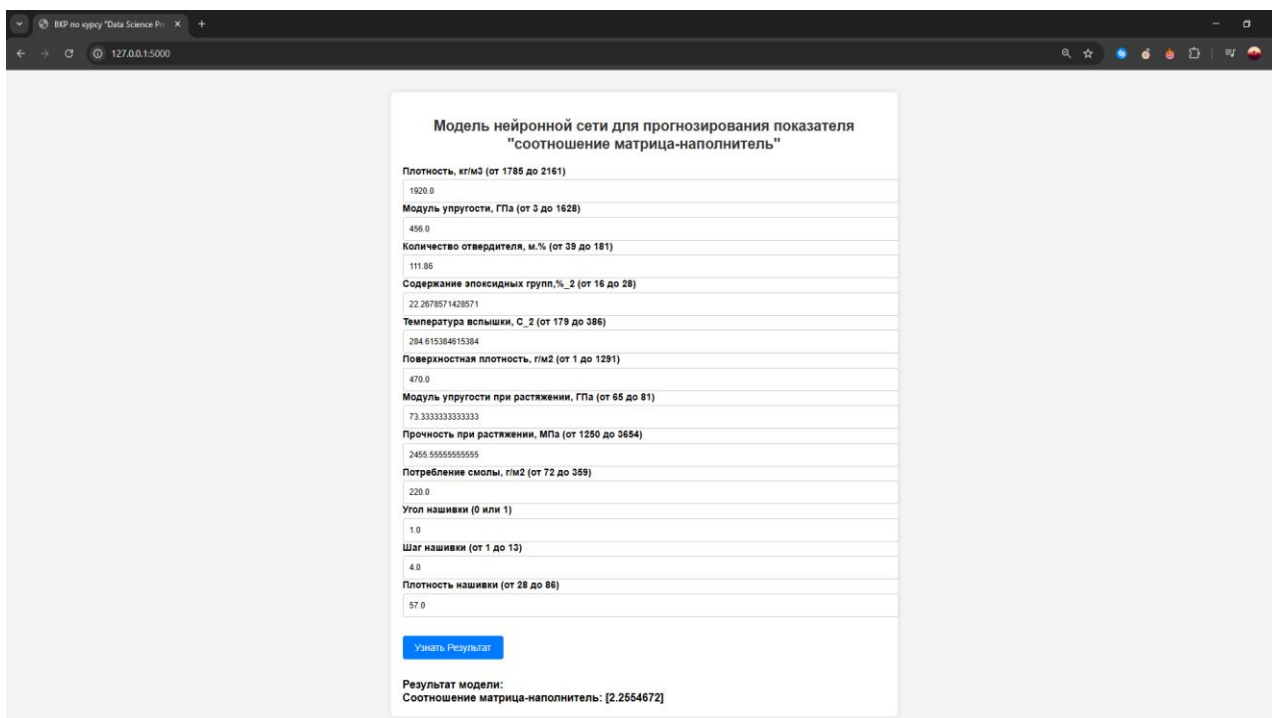


Рисунок 33 – Исходная страница веб-приложения (до ввода параметров)

По итогу у меня вышло реализовать функционал ввода входных параметров (за исключением непосредственно целевого параметра), загрузка сохраненной модели, генерация прогноза и его вывод.



Модель нейронной сети для прогнозирования показателя  
"соотношение матрица-наполнитель"

Плотность, кг/м <sup>3</sup> (от 1785 до 2161)	1920.0
Модуль упругости, ГПа (от 3 до 1628)	456.0
Количество отвердителя, м. % (от 39 до 181)	111.86
Содержание эпоксидных групп, %_2 (от 16 до 28)	22.2678571428571
Температура вспышки, C_2 (от 179 до 386)	284.615384615384
Поверхностная плотность, г/м <sup>2</sup> (от 1 до 1291)	470.0
Модуль упругости при растяжении, ГПа (от 65 до 81)	73.3333333333333
Прочность при растяжении, МПа (от 1260 до 3654)	2455.55555555555
Потребление смолы, г/м <sup>2</sup> (от 72 до 359)	220.0
Угол нашивки (0 или 1)	1.0
Шаг нашивки (от 1 до 13)	4.0
Плотность нашивки (от 28 до 86)	57.0

[Узнать Результат](#)

Результат модели:  
Соотношение матрица-наполнитель: [2.2554672]

Рисунок 34 – Пример генерации результата моделью

## 2.6. Создание репозитория на GitHub

Для агрегации результатов исследования был создан репозиторий на веб-сервисе GitHub по адресу: <https://github.com/PraymA/-DS-Pro>.

В данный репозиторий были загружены ноутбки исследования, презентация на защиту проекта, а также пояснительная записка.

## Заключение

В ходе данной работы были пройдены ключевые этапы исследования: от изучения теоретических методов анализа данных и машинного обучения до предобработки, построения аналитических решений и тестирования приложения. Работа велась с реальными производственными данными, что усложняло процесс, поскольку не существовало заранее известных решений. Проведён разведочный анализ данных, выделены признаки, подготовлены данные для моделей, но используемые алгоритмы не позволили получить достоверные прогнозы.

После нахождения оптимальных гиперпараметров метод лассо-регрессии дал наилучший результат для предсказания параметра модуля упругости при растяжении, а градиентный бустинг – для параметра модуля прочности при растяжении. Тем не менее, отсутствие значимых корреляций между признаками и слабая предсказательная способность моделей указывают на недостатки набора данных, необходимость пересмотра подходов и инструментов прогнозирования.

Среди возможных причин неудачи можно выделить отсутствие чёткой постановки задачи и дополнительной информации о физических процессах, что затрудняло выделение значимых признаков. Также стоит учитывать, что работа велась с уже обработанными данными, и применение альтернативных методов очистки могло бы привести к лучшим результатам. Недостаток знаний и опыта в области нейросетей также мог повлиять на итоговые показатели, так как оптимальный выбор архитектуры сети является сложной задачей.

Для дальнейшей работы стоит углубиться в изучение нейросетевых моделей, протестировать различные архитектуры и методы обучения. В том числе консультации с экспертами в предметной области могут дать новые инсайты, необходимые для более точного прогнозирования характеристик материалов.

Несмотря на сложности, по итогу было разработано приложение с использованием фреймворка Flask. Возможные доработки и последующие исследования могут позволить создать более качественную модель, которая будет интегрирована в готовый инструмент. В приложении реализованы функции ввода входных параметров, загрузки модели и генерации прогнозов.



## Библиографический список

1. Грас Д. Data Science. Наука о данных с нуля: Пер. с англ. - 3-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2023. - 432 с.
2. Документация по библиотеке TensorFlow: – Режим доступа: <https://www.tensorflow.org/overview> (дата обращения: 17.02.2025).
3. Силен Дэви, Мейсман Арно, Али Мохамед. Основы Data Science и Big Data. Python и наука о данных. – СПб.: Питер, 2019. – 352 с.
4. Документация по библиотеке numpy: – Режим доступа: <https://numpy.org/doc/stable/user/index.html#user> (дата обращения: 10.02.2025).
5. Документация по библиотеке seaborn: – Режим доступа: <https://seaborn.pydata.org/tutorial.html> (дата обращения: 12.02.2025).
6. Документация по библиотеке pandas: – Режим доступа: [https://pandas.pydata.org/docs/user\\_guide/index.html#user-guide](https://pandas.pydata.org/docs/user_guide/index.html#user-guide) (дата обращения: 05.02.2025).
7. Alex Maszański. Метод k-ближайших соседей (k-nearest neighbour): – Режим доступа: <https://proglib.io/p/metod-k-blizhayshih-sosedey-k-nearest-neighbour-2021-07-19> (дата обращения: 12.02.2025).
8. Документация по библиотеке scikit-learn: – Режим доступа: [https://scikit-learn.org/stable/user\\_guide.html](https://scikit-learn.org/stable/user_guide.html) (дата обращения: 11.02.2025).
9. Документация по языку программирования Python: – Режим доступа: <https://docs.python.org/3/index.html> (дата обращения: 05.02.2025).
10. Документация по библиотеке keras: – Режим доступа: <https://keras.io/api/> (дата обращения: 17.02.2025).
11. Траск Эндрю. Грокаем глубокое обучение. – СПб.: Питер, 2021. – 368 с.
12. Документация по библиотеке matplotlib: – Режим доступа: <https://matplotlib.org/stable/users/index.html> (дата обращения: 14.02.2025).

13. Руководство по быстрому старту в Flask: – Режим доступа: <https://flask-russian-docs.readthedocs.io/ru/latest/quickstart.html> (дата обращения: 17.02.2025).

14. Yury Kashnitsky. Открытый курс машинного обучения. Тема 3. Классификация, деревья решений и метод ближайших соседей: – Режим доступа: <https://habr.com/ru/company/ods/blog/322534/> (дата обращения: 12.02.2025).

15. Бизли Д. Python. Подробный справочник: учебное пособие. – Пер. с англ. – СПб.: Символ-Плюс, 2022. – 900 с.

16. Yury Kashnitsky. Открытый курс машинного обучения. Тема 5. Композиции: бэггинг, случайный лес: – Режим доступа: <https://habr.com/ru/company/ods/blog/324402/> (дата обращения: 13.02.2025).

17. Alex Maszański. Машинное обучение для начинающих: алгоритм случайного леса (Random Forest): – Режим доступа: <https://proglab.io/p/mashinnoe-obuchenie-dlya-nachinayushchih-algoritm-sluchaynogo-lesa-random-forest-2021-08-12> (дата обращения: 13.02.2025).