



**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»

Слушатель

Киселев Дмитрий Владимирович

Москва, 2023

Содержание

Введение.....	3
1. Аналитическая часть.....	5
1.1. Постановка задачи	5
1.2. Описание используемых методов	6
1.3. Разведочный анализ данных	14
2. Практическая часть	16
2.1. Разбиение и предобработка данных.....	16
2.2. Разработка и обучение моделей	18
2.3. Тестирование моделей.....	18
2.4. Разработка нейронной сети для прогнозирования соотношения матрица-наполнитель	20
2.5. Разработка приложения.....	25
2.6. Создание удаленного репозитория.....	29
3. Заключение	30
4. Библиографический список	31

Введение

Тема данной работы: Прогнозирование конечных свойств новых материалов (композиционных материалов).

Композиционный материал - многокомпонентный материал, изготовленный (человеком или природой) из двух или более компонентов с существенно различными физическими и/или химическими свойствами, которые, в сочетании, приводят к появлению нового материала с характеристиками, отличными от характеристик отдельных компонентов и не являющимися простой их суперпозицией. В составе композита принято выделять матрицу/матрицы и наполнитель/наполнители, последние выполняют функцию армирования (по аналогии с арматурой в таком композиционном строительном материале, как железобетон). В качестве наполнителей композитов как правило выступают углеродные или стеклянные волокна, а роль матрицы играет полимер. Сочетание разных компонентов позволяет улучшить характеристики материала и делает его одновременно лёгким и прочным. При этом отдельные компоненты остаются таковыми в структуре композитов, что отличает их от смесей и затвердевших растворов. Варьируя состав матрицы и наполнителя, их соотношение, ориентацию наполнителя, получают широкий спектр материалов с требуемым набором свойств. Многие композиты превосходят традиционные материалы и сплавы по своим механическим свойствам и в то же время они легче. Использование композитов обычно позволяет уменьшить массу конструкции при сохранении или улучшении её механических характеристик.

По структуре композиты делятся на несколько основных классов: волокнистые, дисперсно-упрочнённые, упрочнённые частицами и нанокомпозиты. Наиболее распространённые: композиционные материалы с металлической матрицей (железобетон); полимерные композиционные материалы (стеклопла-

стики, углепластики, органопластики); композиционные материалы на основе керамики (зубные протезы).

Композиционные материалы применяются:

- в авиационной, ракетной и космической отрасли;
- в автомобильной промышленности;
- в бытовой технике;
- в горнодобывающей промышленности;
- в медицинской промышленности;
- в металлургической промышленности;
- в сельскохозяйственном машиностроении;
- в строительстве;
- в судостроительной промышленности;
- в химической промышленности;
- в электротехнической промышленности;
- в ядерной промышленности;
- и много где ещё.

В связи с вышеперечисленным возникает необходимость исследования настоящей темы работы. Имея характеристики компонентов, рассчитать свойства композиционного материала невозможно, достигается опытным путём. Но на результатах достижений моделируются новые возможные в будущем достижения. Машинное обучение помогает в формировании новых моделей и прогнозировании возможных результатов, а кроме того, может подсказать направление дальнейшего развития.

1. Аналитическая часть

1.1. Постановка задачи

На входе имеются данные о начальных свойствах компонентов композиционных материалов (количество связующего, наполнителя, температурный режим отверждения и т.д.). На выходе необходимо спрогнозировать ряд конечных свойств получаемых композиционных материалов.

Необходимо обучить алгоритм машинного обучения, который будет определять значения:

- Модуль упругости при растяжении, ГПа
- Прочность при растяжении, МПа

Необходимо написать нейронную сеть, которая будет рекомендовать:

- Соотношение матрица-наполнитель

Необходимо написать приложение, которое будет выдавать прогноз полученный в задании.

Датасет состоит из двух файлов: X_br.xlsx (размерностью 10 признаков с 1023 значениями) и X_npr.xlsx (размерностью 3 признака с 1040 значениями). Файлы требуется объединить по индексу с типом INNER. После объединения получился датасет размерностью 13 признаков с 1023 значениями. Пропуски отсутствуют. Обнаружены выбросы при построении boxplot ящик с усами, количеством: 25 выбросов в 24 строках при использовании метода 3-ёх сигм; 93 выброса в 87 строках при использовании межквартильного размаха. Решил удалить выбросы найденные при использовании метода 3-ёх сигм, дабы не потерять возможно значимые данные. Можно было вообще не удалять, видно что датасет и так предварительно очищенный.

1.2. Описание используемых методов

Задача регрессии в машинном обучении - это задача предсказания какой-то численной характеристики объекта предметной области по определенному набору его параметров (атрибутов). Задачи регрессии на практике встречаются довольно часто. Например, предсказание цены объекта недвижимости - классическая регрессионная задача. В таких проблемах атрибутами выступают разные характеристики квартир или домов - площадь, этажность, расположение, расстояние до центра города, количество комнат, год постройки. В разных наборах данных собрана разная информация и, соответственно, модели тоже должны быть разные. Другой пример - предсказание цены акций или других финансовых активов. Или предсказание температуры завтрашним днем.

1.2.1 Метод ближайших соседей

Метод ближайших соседей k-NN – метрический алгоритм для автоматической классификации объектов или регрессии. В случае использования метода для классификации объект присваивается тому классу, который является наиболее распространённым среди соседей данного элемента, классы которых уже известны. В случае использования метода для регрессии, объекту присваивается среднее значение по ближайшим к нему объектам, значения которых уже известны. Алгоритм может быть применен к выборкам с большим количеством атрибутов.

1.2.2 Метод опорных векторов SVR

Метод опорных векторов (**Support Vector Machines — SVM**) – это набор контролируемых методов обучения, используемых для классификации, регрессии и обнаружения выбросов.

Модель метода опорных векторов – отображение данных точками в пространстве, так что между наблюдениями отдельных категорий имеется разрыв, и он максимален.

Каждый объект данных представляется как вектор (точка) в r -мерном пространстве. Он создаёт линию или гиперплоскость, которая разделяет данные на классы.

Достоинства метода: для классификации достаточно небольшого набора данных. Эффективен при большом количестве гиперпараметров. Способен обрабатывать случаи, когда гиперпараметров больше, чем количество наблюдений. Существует возможность гибко настраивать разделяющую функцию.

Недостатки метода: неустойчивость к шуму; для больших наборов данных требуется долгое время обучения.

1.2.3 Метод Решающих деревьев (Decision tree)

Решающие деревья (Decision Trees) - еще один непараметрический метод, применяемый и для классификации, и для регрессии. Решающие правила автоматически генерируются в процессе обучения на обучающем множестве путем обобщения обучающих примеров. Поэтому их называют индуктивными правилами, а сам процесс обучения — индукцией решающих деревьев.

Дерево состоит из элементов двух типов: узлов (node) и листьев (leaf).

В узлах находятся решающие правила и производится проверка соответствия примеров этому правилу. В результате проверки множество примеров, попавших в узел, разбивается на два подмножества: удовлетворяющие правилу и не удовлетворяющие ему. Затем к каждому подмножеству вновь применяется правило и процедура рекурсивно повторяется пока не будет достигнуто некоторое условие остановки алгоритма. В последнем узле проверка и разбиение не производится и он объявляется листом.

В листе содержится не правило, а подмножество объектов, удовлетворяющих всем правилам ветви, которая заканчивается данным листом. Для классификации — это класс, ассоциируемый с узлом, а для регрессии — соответствующий листу интервал целевой переменной.

При формировании правила для разбиения в очередном узле дерева необходимо выбрать атрибут, по которому это будет сделано. Общее правило для классификации — выбранный атрибут должен разбить множество наблюдений в узле так, чтобы результирующие подмножества содержали примеры с одинаковыми метками класса, а количество объектов из других классов в каждом из этих множеств было как можно меньше.

Для регрессии критерием является дисперсия вокруг среднего. Минимизируя дисперсию вокруг среднего, мы ищем признаки, разбивающие выборку таким образом, что значения целевого признака в каждом листе примерно равны.

Преимущество деревьев решений в том, что они легко интерпретируемы, понятны человеку. Они могут использоваться для извлечения правил на естественном языке. Высокая точность работы, нетребовательность к подготовке данных.

Недостаток деревьев решений - склонность переобучаться. Переобучение в случае дерева решений — это точное распознавание примеров, участвующих в обучении и полная несостоятельность на новых данных. В худшем случае, дерево будет большой глубины и сложной структуры, а в каждом листе будет только один объект.

1.2.4 Метод Случайного леса (Random forest)

Метод Случайного леса (Random forest) — это множество решающих деревьев. Универсальный алгоритм машинного обучения с учителем, представи-

тель ансамблевых методов. Усреднение предсказаний отдельных деревьев. Для определения входных данных каждому дереву используется метод случайных подпространств. Базовые алгоритмы обучаются на различных подмножествах признаков, которые выделяются случайным образом.

Достоинства метода: не переобучается; не требует предобработки входных данных; эффективно обрабатывает пропущенные данные, данные с большим числом классов и признаков; имеет высокую точность предсказания и внутреннюю оценку обобщающей способности модели, а также высокую параллелизуемость и масштабируемость.

Недостатки метода: построение занимает много времени; сложно интерпретируемый; не обладает возможностью экстраполяции; может недо обучаться; трудоёмко прогнозируемый; иногда работает хуже, чем линейные методы.

1.2.5 Линейная регрессия

Линейная регрессия — это метод анализа данных, который предсказывает ценность неизвестных данных с помощью другого связанного и известного значения данных. Он математически моделирует неизвестную или зависимую переменную и известную или независимую переменную в виде линейного уравнения. Это один из самых простых и эффективных инструментов статистического моделирования. Она определяет зависимость переменных с помощью линии наилучшего соответствия. Например, предположим, что у вас есть данные о ваших расходах и доходах за прошлый год. Методы линейной регрессии анализируют эти данные и определяют, что ваши расходы составляют половину вашего дохода. Затем они рассчитывают неизвестные будущие расходы, сокращая вдвое будущий известный доход.

Достоинства метода: быстрый и простой в реализации; легко интерпретируемый; имеет меньшую сложность по сравнению с другими алгоритмами.

Недостатки метода: моделирует только прямые линейные зависимости; требует прямую связь между зависимыми и независимыми переменными; выбросы оказывают огромное влияние, а границы линейны.

1.2.6 Гребневая регрессия RidgeCV

Метод регрессии лассо (LASSO, Least Absolute Shrinkage and Selection Operator) — это вариация линейной регрессии, специально адаптированная для данных, которые имеют сильную корреляцию признаков друг с другом. LASSO использует сжатие коэффициентов (shrinkage) и этим пытается уменьшить сложность данных, искривляя пространство, на котором они лежат. В этом процессе лассо автоматически помогает устранить или исказить сильно коррелированные и избыточные функции в методе с низкой дисперсией. Регрессия лассо использует регуляризацию L1, то есть взвешивает ошибки по их абсолютному значению.

Гребневая регрессия или ридж-регрессия — так же вариация линейной регрессии, очень похожая на регрессию LASSO. Она так же применяет сжатие и так же хорошо работает для данных, которые демонстрируют сильную мультиколлинеарность. Самое большое различие между ними в том, что гребневая регрессия использует регуляризацию L2, которая взвешивает ошибки по их квадрату, чтобы сильнее наказывать за более значительные ошибки.

Регуляризация позволяет интерпретировать модели. Если коэффициент стал 0 (для Lasso) или близким к 0 (для Ridge), значит данный входной признак не является значимым.

1.2.7 Повышение градиента дерева (GradientBoostingRegressor)

Градиентный бустинг (Gradient Boosting) — это ансамбль деревьев решений, обученный с использованием градиентного бустинга. В основе данного алгоритма лежит итеративное обучение деревьев решений с целью минимизиро-

вать функцию потерь. Основная идея градиентного бустинга: строятся последовательно несколько базовых классификаторов, каждый из которых как можно лучше компенсирует недостатки предыдущих. Финальный классификатор является линейной композицией этих базовых классификаторов.

Повышение градиента опирается на логику, в которой последующие предикторы учатся на ошибках предыдущих предикторов. Таким образом, к концу мы получаем лучшие прогнозы и ускоряем обучение.

Достоинства метода: новые алгоритмы учатся на ошибках предыдущих; требуется меньше итераций, чтобы приблизиться к фактическим прогнозам; наблюдения выбираются на основе ошибки; простой в настройке темпа обучения и применения; легко интерпретируем.

Недостатки метода: необходимо тщательно выбирать критерии остановки, иначе это может привести к переобучению; наблюдения с наибольшей ошибкой появляются чаще; слабее и менее гибко чем нейронные сети.

1.2.8 Стохастический градиентный спуск (SGD) SGDRegressor

Стохастический градиентный спуск (SGDRegressor) — это простой, но очень эффективный подход к подгонке линейных классификаторов и регрессоров под выпуклые функции потерь. Этот подход подразумевает корректировку весов нейронной сети, используя аппроксимацию градиента функционала, вычисленную только на одном случайном обучающем примере из выборки. Градиент потерь оценивается для каждой выборки за раз, и модель обновляется по пути с уменьшением графика силы (он же скорость обучения). Регуляризатор — это штраф, добавленный к функции потерь, который сжимает параметры модели до нулевого вектора, используя либо квадрат евклидовой нормы L2, либо абсолютную норму L1, либо их комбинацию (эластичная сеть). Если обновление параметра пересекает значение 0,0 из-за регуляризатора, обновле-

ние усекается до 0,0, чтобы можно было изучить разреженные модели и добиться выбора онлайн-функций.

Достоинства метода: эффективен; прост в реализации; имеет множество возможностей для настройки кода; способен обучаться на избыточно больших выборках.

Недостатки метода: требует ряд гиперпараметров; чувствителен к масштабированию функций; может не сходиться или сходиться слишком медленно; функционал многоэкстремален; процесс может "застрять" в одном из локальных минимумов; возможно переобучение.

1.2.9 Регрессор, который делает прогнозы, используя простые правила `DummyRegressor(*, strategy='mean', constant=None, quantile=None)`

Этот регрессор полезен в качестве простой основы для сравнения с другими (реальными) регрессорами. Обычно не используется его для решения реальных задач.

1.2.10 AdaBoostRegressor

Регрессор AdaBoost — это метаоценщик, который начинает с подгонки регрессора к исходному набору данных, а затем подбирает дополнительные копии регрессора к тому же набору данных, но где веса экземпляров корректируются в соответствии с ошибкой текущего прогноза. Таким образом, последующие регрессоры больше фокусируются на сложных случаях. Это повышающий регрессор с постоянно лучшей производительностью.

Основной принцип AdaBoost состоит в том, чтобы подогнать последовательность слабых учеников (т. е. Моделей, которые лишь немного лучше, чем случайное предположение, например, небольшие деревья решений) на многократно изменяемых версиях данных. Прогнозы от всех из них затем объединяются посредством взвешенного большинства голосов (или суммы) для получе-

ния окончательного прогноза. Модификации данных на каждой так называемой итерации повышения состоят в применении весов к каждой из обучающих выборок. Первоначально все веса установлены на значении $1/(\text{количество всех весов})$, так что первый шаг просто обучает слабого обучаемого на исходных данных. Для каждой последующей итерации веса выборки индивидуально изменяются, и алгоритм обучения повторно применяется к повторно взвешенным данным. На данном этапе те обучающие примеры, которые были неправильно предсказаны усиленной моделью, созданной на предыдущем шаге, имеют увеличенные веса, тогда как веса уменьшаются для тех, которые были предсказаны правильно. По мере продолжения итераций, примеры, которые трудно предсказать, получают все большее влияние. Таким образом, каждый последующий слабый ученик вынужден концентрироваться на примерах, которые упускают предыдущие в последовательности.

1.2.11 Нейронная сеть

Нейронная сеть — это множество нейронов, соединенных между собой связями. Структура нейронной сети пришла в мир программирования из биологии. Вычислительная единица нейронной сети — нейрон или персептрон.

У каждого нейрона есть определённое количество входов, куда поступают сигналы, которые суммируются с учётом значимости (веса) каждого входа.

Смещение — это дополнительный вход для нейрона, который равен 1 и имеет собственный вес соединения.

У нейрона есть функция активации, которая определяет выходное значение нейрона. Она используется для того, чтобы ввести нелинейность в нейронную сеть. Примеры активационных функций: `relu`, `linear`, `sigmoid`, `softmax`, `tanh`.

У полносвязной нейросети выход каждого нейрона подается на вход всем нейронам следующего слоя.

В нейросети есть:

- входной слой — его размер соответствует входным параметрам;
- скрытые слои — их количество и размерность определяем специалист;
- выходной слой — его размер соответствует выходным параметрам.

Прямое распространение — это процесс передачи входных значений в нейронную сеть и получения выходных данных, которые называются прогнозируемым значением.

Прогнозируемое значение сравниваем с фактическим с помощью функции потерь. В методе обратного распространения ошибки градиенты (производные значений ошибок) вычисляются по значениям весов в направлении, обратном прямому распространению сигналов. Значение градиента вычитают из значения веса, чтобы уменьшить значение ошибки. Таким образом происходит процесс обучения. Обновляются веса каждого соединения, чтобы функция потерь минимизировалась.

Для обновления весов в модели используются различные оптимизаторы.

Количество эпох показывает, сколько раз выполнялся проход для всех примеров обучения.

Нейронные сети применяются для решения задач регрессии, классификации, распознавания объектов, речи и звуков, компьютерного зрения и других. На настоящий момент это самый мощный, гибкий и широко применяемый инструмент в машинном обучении.

1.3. Разведочный анализ данных

Цель разведочного анализа — получение первоначальных представлений о характерах распределений переменных исходного набора данных, формирование оценки качества исходных данных (наличие пропусков, выбросов), выявление характера взаимосвязи между переменными с целью последующего вы-

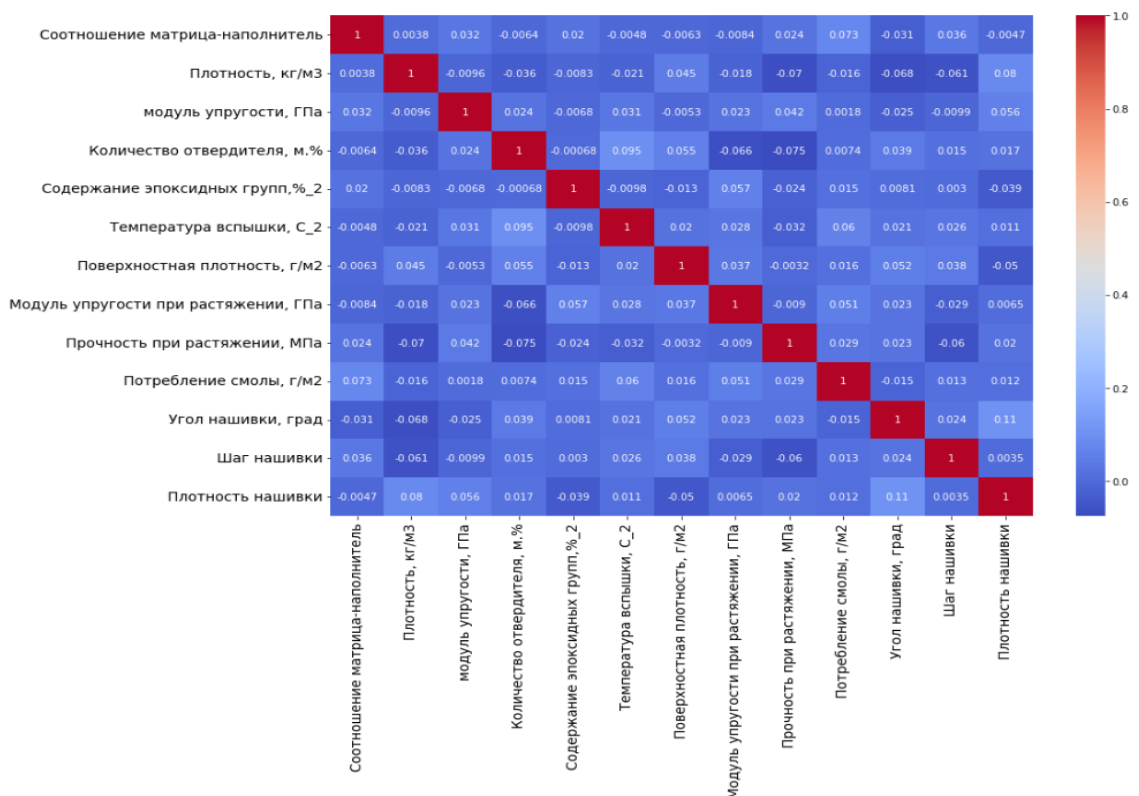
движения гипотез о наиболее подходящих для решения задачи моделях машинного обучения.

Но прежде чем передать данные в работу моделей машинного обучения, необходимо обработать и очистить их. Необработанные данные могут содержать искажения и пропущенные значения – это ненадёжно, поскольку способно привести к крайне неверным результатам по итогам моделирования. Но безосновательно удалять что-либо тоже неправильно. Именно поэтому сначала набор данных надо изучить. Метод `info()`, в котором видны пропущенные значения и тип данных.

Обязательно смотрим описательную статистику — метод `describe()`.

Проверяем на количество уникальных значений — метод `nunique()`.

Смотрим корреляцию между признаками — метод `corr()`, в купе к нему для наглядности рисуем тепловую карту из которой видны зависимости между признаками и тем они сильнее как значение стремится к -1 или к 1, и тем слабее как значение стремится к 0.



В нашем случае корреляция около 0, а значит взаимосвязи крайне слабые.

2. Практическая часть

2.1. Разбиение и предобработка данных

Распределения у признаков очень близки к нормальному распределению. Выбросов немного, посчитаем и удалим их (решил использовать метод 3-ёх сигм). kde (оценка плотности ядра) показала, что в нашем датасете, признаки находятся в разных диапазонах, поэтому необходимо нормализовать признаки в датасете. Проведём нормализацию или стандартизацию данных (для целевой переменной «Модуль упругости при растяжении, ГПа» использовал нормализованный датасет, а для целевой переменной «Прочность при растяжении, МПа» использовал стандартизированный датасет).

Нормализация – это процедура предобработки входной информации (обучающих, тестовых и валидационных выборок, а также реальных данных), при которой значения признаков во входном векторе приводятся к некоторому заданному диапазону, например, $[0 \dots 1]$ или $[-1 \dots 1]$.

$$X_{i_norm} = (X_i - \min(X)) / (\max(X) - \min(X))$$

X_{i_norm} - нормализованный элемент признака

X_i - непреобразованный элемент признака

$\min(X)$, $\max(X)$ – наименьший, наибольший элемент признака

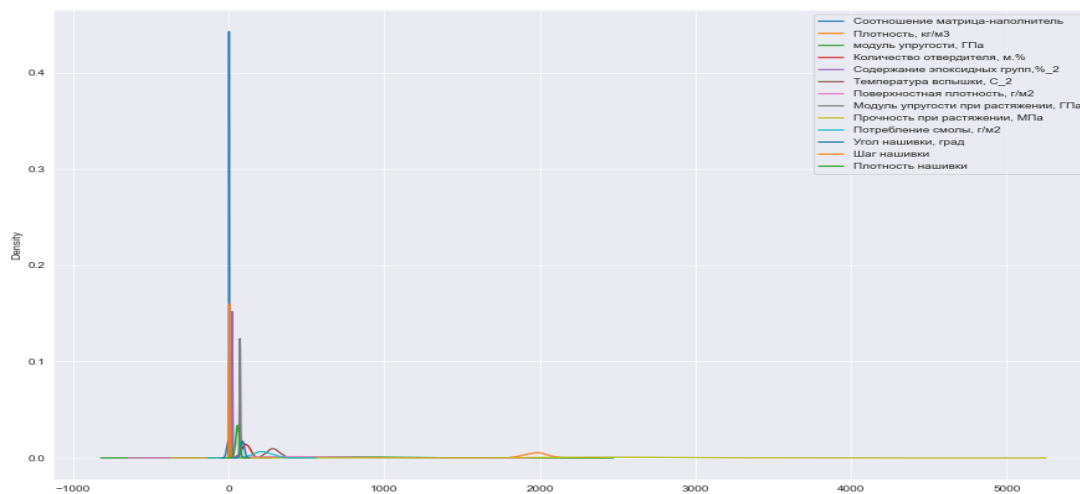
Стандартизация - это приведения к определённому формату и представлению, которые обеспечивают их корректное применение в многомерном анализе, совместных исследованиях, сложных технологиях аналитической обработки. Приведение к математическому ожиданию (среднему значению) равному 0 и стандартному отклонению равному 1.

$$z = (x - \mu) / \sigma$$

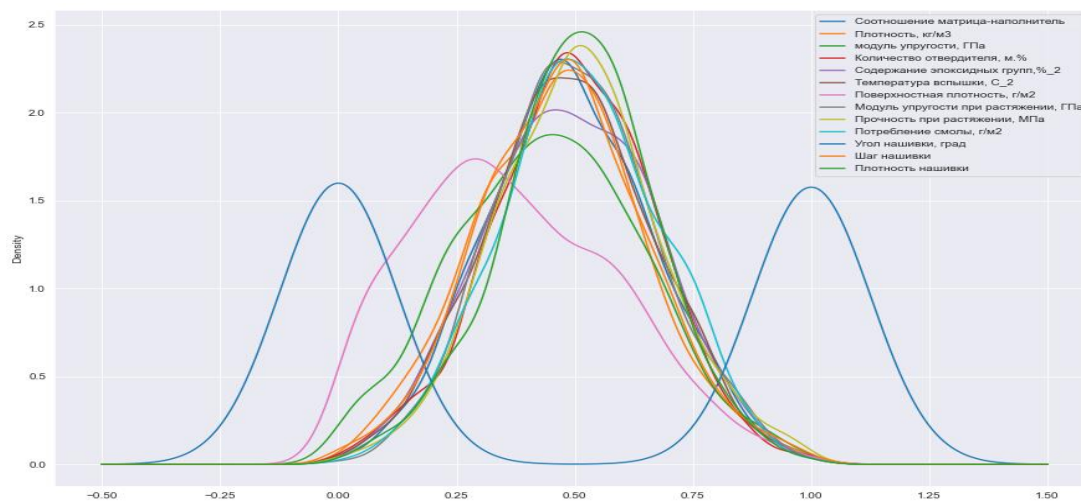
x - значение точки данных; σ - стандартное отклонение;

μ - среднее значение (математическое ожидание).

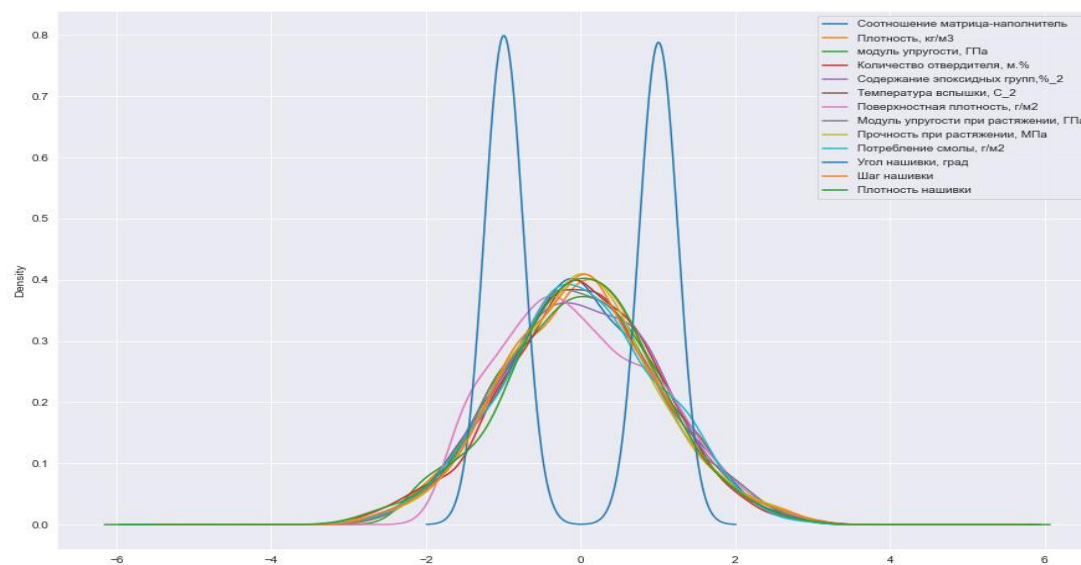
График до нормализации:



Нормализованные данные:



Стандартизированные данные:



2.2. Разработка и обучение моделей

Разработка и обучение моделей машинного обучения осуществлялась для двух выходных параметров:

- Модуль упругости при растяжении, ГПа (будем использовать нормализованный датасет)
- Прочность при растяжении, МПа (будем использовать стандартизированный датасет)

Для решения применим все методы, описанные выше.

Разобьём датасет на тренировочную и тестовую выборку в соотношении 70% и 30% соответственно.

Поиск гиперпараметров по сетке реализует класс GridSearchCV из библиотеки sklearn. Он получает модель и набор гиперпараметров, поочередно передает их в модель, выполняет обучение и определяет лучшие комбинации гиперпараметры. Перекрестная проверка уже встроена в этот класс. Применим её в каждом методе.

2.3. Тестирование моделей

Для модуля упругости при растяжении, ГПа:

	MSE:	MAE:	R2:
KNeighborsRegressor(n_neighbors=30)	0.026989	0.132739	-0.020474
SVR(kernel="poly", gamma="auto")	0.026254	0.130561	0.007326
DecisionTreeRegressor(criterion="absolute_error", splitter="random", max_depth=1, random_state=1)	0.026285	0.130917	0.006147
RandomForestRegressor(n_estimators=150, criterion="absolute_error", random_state=1)	0.027026	0.132052	-0.021895
LinearRegression(fit_intercept=True, copy_X=True, n_jobs=-1, positive=True)	0.026172	0.130441	0.010415
RidgeCV(fit_intercept=True, gcv_mode="eigen")	0.026343	0.130956	0.003947
GradientBoostingRegressor(n_estimators=50, max_depth=3, random_state=1)	0.028132	0.136365	-0.063705
SGDRegressor(penalty="l1", shuffle=False, learning_rate="adaptive", random_state=1)	0.026533	0.132072	-0.003246
DummyRegressor(strategy="median", quantile=0.1)	0.026454	0.130975	-0.000254
AdaBoostRegressor(n_estimators=25, learning_rate=0.1, random_state=1)	0.026307	0.130463	0.005294

На алгоритме линейной регрессии модель выдала лучший результат.



Для прочности при растяжении, МПа:

	MSE:	MAE:	R2:
KNeighborsRegressor(n_neighbors=15)	1.113348	0.829424	-0.069217
SVR(kernel="linear", gamma="scale")	1.059066	0.828067	-0.017088
DecisionTreeRegressor(criterion="squared_error", splitter="best", max_depth=1, random_state=1)	1.067445	0.814039	-0.025134
RandomForestRegressor(n_estimators=50, criterion="absolute_error", random_state=1)	1.050733	0.813191	-0.009085
LinearRegression(fit_intercept=False, copy_X=True, n_jobs=-1, positive=True)	1.066604	0.823477	-0.024326
RidgeCV(fit_intercept=False, gcv_mode="auto")	1.065749	0.822930	-0.023506
GradientBoostingRegressor(n_estimators=50, max_depth=3, random_state=1)	1.084678	0.819900	-0.041684
SGDRegressor(penalty="l1", shuffle=True, learning_rate="invscaling", random_state=1)	1.066826	0.823175	-0.024540
DummyRegressor(strategy="mean", quantile=0.1)	1.041286	0.802264	-0.000012
AdaBoostRegressor(n_estimators=25, learning_rate=0.3, random_state=1)	1.051999	0.812497	-0.010301

Здесь все результаты плохие, наименее худший результат был получен на регрессоре, который вообще не рекомендуется к использованию в реальных задачах.



В этой работе я использую следующие метрики:

– MSE (Mean Squared Error) измеряет среднее значение квадратов ошибок, то есть среднюю квадратическую разницу между оценочными значениями и фактическим значением

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_{true} - y_{pred})^2 ;$$

– MAE (Mean Absolute Error) - средняя абсолютная ошибка так же принимает значения в тех же единицах, что и целевая переменная

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

$\underbrace{\hspace{1.5cm}}_{\text{test set}}$
 $\underbrace{\hspace{1.5cm}}_{\text{predicted value}}$
 $\underbrace{\hspace{1.5cm}}_{\text{actual value}}$

– R2 или коэффициент детерминации измеряет долю дисперсии, объясненную моделью, в общей дисперсии целевой переменной. Если он близок к единице, то модель хорошо объясняет данные, если же он близок к нулю, то прогнозы сопоставимы по качеству с константным предсказанием (горизонтальная линия объясняет данные так же хорошо, как и полученная модель). Если он принимает отрицательные значения, то это говорит о том, что горизонтальная линия объясняет данные лучше, чем полученная модель.

$$R^2 = 1 - \frac{D[y|x]}{D[y]} = 1 - \frac{\sigma^2}{\sigma_y^2},$$

где $D[y] = \sigma_y^2$ — дисперсия случайной величины y , а $D[y|x] = \sigma^2$ — условная (по факторам x) дисперсия зависимой переменной (дисперсия ошибки модели).

2.4. Разработка нейронной сети для прогнозирования соотношения матрица-наполнитель

Написал несколько нейронных сетей пробуя найти лучшую, меняя количество нейронов, слоёв, функций активации (использовал различные вариации ФА, такие как sigmoid, softmax, tanh, relu, linear). Оставил из них 7, которые, так сказать, из всех прочих лучше себя показали:

1.

```
model1 = Sequential()
#model1.add(Input(shape=X_train.shape[1]))
model1.add(Dense(128, activation='sigmoid', input_shape=[X_train.shape[1]]))
model1.add(Dense(128, activation='sigmoid'))
model1.add(Dense(1, activation='relu'))
#выведем полученную модель на экран
model1.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	1664
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 1)	129

```
=====
Total params: 18,305
Trainable params: 18,305
Non-trainable params: 0
```

2.

```
model2 = Sequential()
model2.add(Input(shape=X_train.shape[1]))
model2.add(Dense(8, activation='sigmoid'))
model2.add(Dense(8, activation='sigmoid'))
model2.add(Dense(1, activation='linear'))
#выведем полученную модель на экран
model2.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 8)	104
dense_4 (Dense)	(None, 8)	72
dense_5 (Dense)	(None, 1)	9

```
=====
Total params: 185
Trainable params: 185
Non-trainable params: 0
```

3.

```
model3 = Sequential()
model3.add(Input(shape=X_train.shape[1]))
#добавим смещение, чтобы модель стала более гибкой
model3.add(Dense(128, activation='softmax', use_bias=True))
model3.add(Dense(64, activation='softmax', use_bias=True))
model3.add(Dense(32, activation='softmax', use_bias=True))
model3.add(Dense(1, activation='sigmoid', use_bias=True))
#выведем полученную модель на экран
model3.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 128)	1664
dense_7 (Dense)	(None, 64)	8256
dense_8 (Dense)	(None, 32)	2080
dense_9 (Dense)	(None, 1)	33

```
=====
Total params: 12,033
Trainable params: 12,033
Non-trainable params: 0
```

4.

```
model14 = Sequential()
model14.add(Input(shape=X_train.shape[1]))
model14.add(Dense(1028, activation='sigmoid'))
model14.add(Dense(512, activation='sigmoid'))
model14.add(Dense(256, activation='sigmoid'))
model14.add(Dense(128, activation='sigmoid'))
model14.add(Dense(64, activation='sigmoid'))
model14.add(Dense(1, activation='relu'))
#выведем полученную модель на экран
model14.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 1028)	13364
dense_11 (Dense)	(None, 512)	526848
dense_12 (Dense)	(None, 256)	131328
dense_13 (Dense)	(None, 128)	32896
dense_14 (Dense)	(None, 64)	8256
dense_15 (Dense)	(None, 1)	65
Total params: 712,757		
Trainable params: 712,757		
Non-trainable params: 0		

5.

```
model15 = Sequential()
model15.add(Input(shape=X_train.shape[1]))
model15.add(Dense(2048, activation='tanh'))
model15.add(Dense(256, activation='tanh'))
model15.add(Dense(64, activation='tanh'))
model15.add(Dense(1, activation='linear'))
#выведем полученную модель на экран
model15.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_16 (Dense)	(None, 2048)	26624
dense_17 (Dense)	(None, 256)	524544
dense_18 (Dense)	(None, 64)	16448
dense_19 (Dense)	(None, 1)	65
Total params: 567,681		
Trainable params: 567,681		
Non-trainable params: 0		

6.

```
model16 = Sequential()
model16.add(Input(shape=X_train.shape[1]))
model16.add(Dense(512, activation='sigmoid'))
model16.add(Dense(64, activation='sigmoid'))
model16.add(Dense(8, activation='sigmoid'))
model16.add(Dense(1, activation='linear'))
#выведем полученную модель на экран
model16.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_20 (Dense)	(None, 512)	6656
dense_21 (Dense)	(None, 64)	32832
dense_22 (Dense)	(None, 8)	520
dense_23 (Dense)	(None, 1)	9
Total params: 40,017		
Trainable params: 40,017		
Non-trainable params: 0		

7.

```
model7 = Sequential()
model7.add(Input(shape=X_train.shape[1]))
model7.add(Dense(128, activation='sigmoid'))
model7.add(Dense(64, activation='sigmoid'))
model7.add(Dense(32, activation='sigmoid'))
model7.add(Dense(16, activation='sigmoid'))
model7.add(Dense(8, activation='sigmoid'))
model7.add(Dense(1, activation='linear'))
#выведем полученную модель на экран
model7.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(None, 128)	1664
dense_25 (Dense)	(None, 64)	8256
dense_26 (Dense)	(None, 32)	2080
dense_27 (Dense)	(None, 16)	528
dense_28 (Dense)	(None, 8)	136
dense_29 (Dense)	(None, 1)	9

```
-----
Total params: 12,673
Trainable params: 12,673
Non-trainable params: 0
```

Во всех нейросетях использовал:

```
model7.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_squared_error'])
```

#функция ранней остановки, patience=5 - если в течение 5-ти эпох потери на валидации (val_loss) не будут уменьшаться
stop7 = tf.keras.callbacks.EarlyStopping(monitor='val_loss', verbose=1, patience=5)

```
history7_train = model7.fit(X_train, y_train,
                             batch_size=70,
                             epochs=30,
                             verbose=1,
                             callbacks=[stop7],
                             validation_data=(X_val, y_val))
```

Разбивку сделал следующим образом:

```
X = df_ss.drop('Соотношение матрица-наполнитель', axis=1)
y = df_ss['Соотношение матрица-наполнитель']
X_tr, X_test, y_tr, y_test = train_test_split(X, y, test_size=0.15, random_state=1)
```

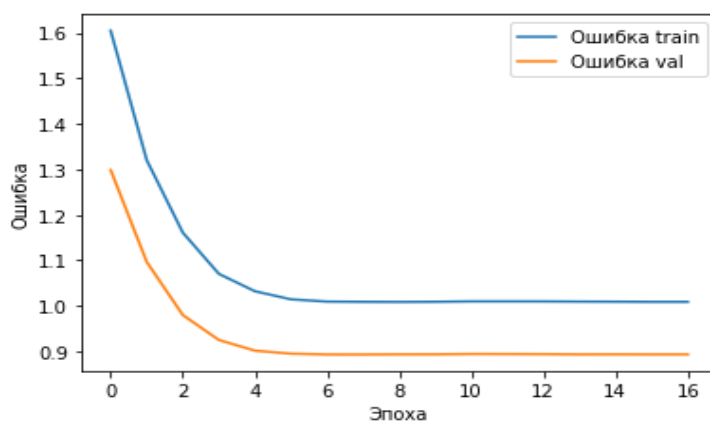
```
X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, test_size=0.15, random_state=1)
```

Для того, чтобы обучить модель, потренировать, протестировать на валидационных данных и посмотреть результат на тестовой выборке.


```
Epoch 1/30
11/11 [=====] - 2s 30ms/step - loss: 1.0264 - mean_squared_error: 1.6047 - val_loss: 0.8985 - val_mean_squared_error: 1.2987
Epoch 2/30
11/11 [=====] - 0s 9ms/step - loss: 0.9177 - mean_squared_error: 1.3203 - val_loss: 0.8250 - val_mean_squared_error: 1.0967
Epoch 3/30
11/11 [=====] - 0s 8ms/step - loss: 0.8547 - mean_squared_error: 1.1611 - val_loss: 0.7818 - val_mean_squared_error: 0.9801
Epoch 4/30
11/11 [=====] - 0s 10ms/step - loss: 0.8187 - mean_squared_error: 1.0705 - val_loss: 0.7615 - val_mean_squared_error: 0.9252
Epoch 5/30
11/11 [=====] - 0s 6ms/step - loss: 0.8048 - mean_squared_error: 1.0323 - val_loss: 0.7516 - val_mean_squared_error: 0.9017
Epoch 6/30
11/11 [=====] - 0s 7ms/step - loss: 0.7983 - mean_squared_error: 1.0145 - val_loss: 0.7485 - val_mean_squared_error: 0.8952
Epoch 7/30
11/11 [=====] - 0s 6ms/step - loss: 0.7972 - mean_squared_error: 1.0096 - val_loss: 0.7487 - val_mean_squared_error: 0.8935
Epoch 8/30
11/11 [=====] - 0s 7ms/step - loss: 0.7976 - mean_squared_error: 1.0091 - val_loss: 0.7489 - val_mean_squared_error: 0.8934
Epoch 9/30
11/11 [=====] - 0s 7ms/step - loss: 0.7971 - mean_squared_error: 1.0088 - val_loss: 0.7484 - val_mean_squared_error: 0.8937
Epoch 10/30
11/11 [=====] - 0s 7ms/step - loss: 0.7972 - mean_squared_error: 1.0092 - val_loss: 0.7484 - val_mean_squared_error: 0.8937
Epoch 11/30
11/11 [=====] - 0s 7ms/step - loss: 0.7975 - mean_squared_error: 1.0100 - val_loss: 0.7483 - val_mean_squared_error: 0.8945
Epoch 12/30
11/11 [=====] - 0s 7ms/step - loss: 0.7973 - mean_squared_error: 1.0101 - val_loss: 0.7483 - val_mean_squared_error: 0.8944
Epoch 13/30
11/11 [=====] - 0s 7ms/step - loss: 0.7972 - mean_squared_error: 1.0100 - val_loss: 0.7483 - val_mean_squared_error: 0.8941
Epoch 14/30
11/11 [=====] - 0s 7ms/step - loss: 0.7973 - mean_squared_error: 1.0096 - val_loss: 0.7486 - val_mean_squared_error: 0.8936
Epoch 15/30
11/11 [=====] - 0s 7ms/step - loss: 0.7972 - mean_squared_error: 1.0094 - val_loss: 0.7485 - val_mean_squared_error: 0.8937
Epoch 16/30
11/11 [=====] - 0s 7ms/step - loss: 0.7971 - mean_squared_error: 1.0090 - val_loss: 0.7486 - val_mean_squared_error: 0.8936
Epoch 17/30
11/11 [=====] - 0s 7ms/step - loss: 0.7972 - mean_squared_error: 1.0090 - val_loss: 0.7486 - val_mean_squared_error: 0.8936
Epoch 17: early stopping
```

Как видно на 17 эпохе обучение остановилось, ввиду того, что ошибка перестала уменьшаться в течении 5-ти эпох, поэтому сработала функция ранней остановки, данные 7-й нейросети.

Так изменялась ошибка.



Сравнение метрик.

	0	1		0
history1_test	0.848025	1.050867	r2_score1	-0.001128
history2_test	0.860793	1.068046	r2_score2	-0.017495
history3_test	0.905366	1.186593	r2_score3	-0.130430
history4_test	0.848025	1.050867	r2_score4	-0.001128
history5_test	0.919941	1.220210	r2_score5	-0.162457
history6_test	0.849433	1.052788	r2_score6	-0.002959
history7_test	0.847440	1.050008	r2_score7	-0.000310

Как видно 7-я нейросеть сработала лучше чем остальные. Но не лучше методов машинного обучения, т.к. предсказания её, так же далеки от истинных значений. Но могло бы быть и хуже.



2.5. Разработка приложения

Разработаны три приложения предсказывающие значения каждой из наших трёх целевых переменных. Приложения разрабатывались на языке программирования Python с помощью фреймворка Flask в редакторе кода для кроссплатформенной разработки веб и облачных приложений Visual Studio Code.

Для модуля упругости при растяжении код выглядит следующим образом:

```

1  from flask import Flask, request, render_template
2  import pickle
3
4  app1 = Flask(__name__, template_folder='templates1')
5
6  @app1.route('/', methods=['POST', 'GET'])
7
8  def main1():
9      if request.method == 'GET':
10         return render_template('main1.html')
11     if request.method == 'POST':
12         loaded_model1 = pickle.load(open('model_lin_reg.pkl', 'rb'))
13         f1 = float(request.form['feature1'])
14         f2 = float(request.form['feature2'])
15         f3 = float(request.form['feature3'])
16         f4 = float(request.form['feature4'])
17         f5 = float(request.form['feature5'])
18         f6 = float(request.form['feature6'])
19         f7 = float(request.form['feature7'])
20         f8 = float(request.form['feature8'])
21         f9 = float(request.form['feature9'])
22         f10 = float(request.form['feature10'])
23         f11 = float(request.form['feature11'])
24         f12 = float(request.form['feature12'])
25         y_pred = loaded_model1.predict([[f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12]])
26         return render_template('main1.html', result=y_pred)
27
28  app1.run()

```

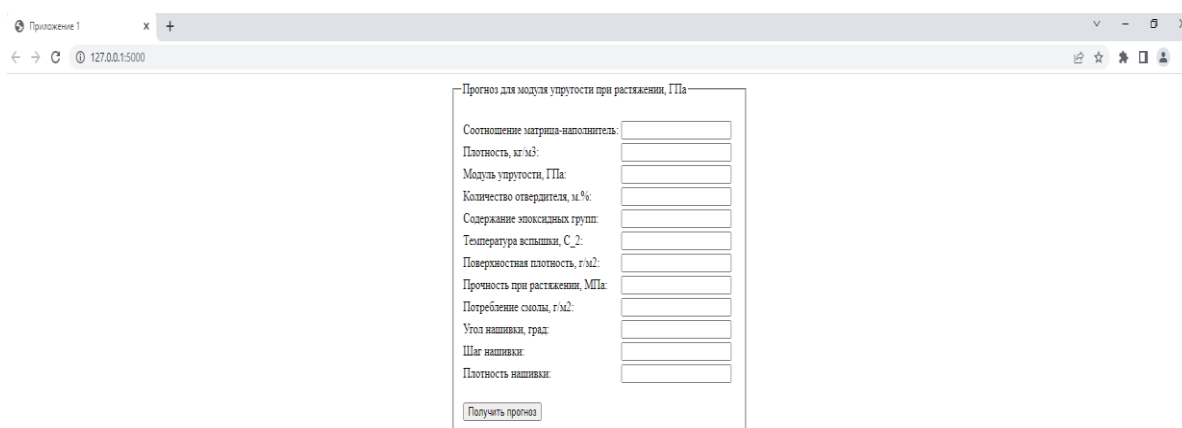
Осуществляем запуск, получаем следующее сообщение:

```

* Serving Flask app "app1" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

```

Переходим по ссылке и осуществляется переход в браузер:



Приложение 1 x +

← → 127.0.0.1:5000

Прогноз для модуля упругости при растяжении, ГПа

Соотношение матрица-наполнитель:

Плотность, кг/м³:

Модуль упругости, ГПа:

Количество отверстий, м.%:

Содержание эпоксидных групп:

Температура вспыхива, С.2:

Поверхностная плотность, г/м²:

Прочность при растяжении, МПа:

Потребление смолы, г/м²:

Угол нашивки, град:

Шаг нашивки:

Плотность нашивки:

Вводим значения во все поля, например:

Прогноз для модуля упругости при растяжении, ГПа

Соотношение матрица-наполнитель:	<input type="text" value="1,1"/>
Плотность, кг/м3:	<input type="text" value="2.2"/>
Модуль упругости, ГПа:	<input type="text" value="3,3"/>
Количество отвердителя, м. %:	<input type="text" value="4.4"/>
Содержание эпоксидных групп:	<input type="text" value="5,5"/>
Температура вспышки, С_2:	<input type="text" value="6.6"/>
Поверхностная плотность, г/м2:	<input type="text" value="7,7"/>
Прочность при растяжении, МПа:	<input type="text" value="8.8"/>
Потребление смолы, г/м2:	<input type="text" value="9,9"/>
Угол нашивки, град:	<input type="text" value="10.10"/>
Шаг нашивки:	<input type="text" value="11,11"/>
Плотность нашивки:	<input type="text" value="12.12"/>

Нажимаем кнопку отправить, получаем результат:

Модуль упругости при растяжении, ГПа

[2.15897088]

Можем дальше вводить данные и получать результаты.

Для прочности при растяжении код был усовершенствован:

```

8  def main2():
9      if request.method == 'GET':
10         return render_template('main2.html')
11     if request.method == 'POST':
12         loaded_model2 = pickle.load(open('model_DummyRegressor.pkl', 'rb'))
13         features = []
14         for i in range(1, 13):
15             f = float(request.form[f'feature{i}'])
16             features.append(f)
17         y_pred = loaded_model2.predict([[features]])
18         return render_template('main2.html', result=y_pred)
19
20 app2.run()
```

Продельываем ту же самую процедуру и получаем:

Прогноз для прочности при растяжении, МПа

Соотношение матрица-наполнитель:	<input type="text" value="12,12"/>
Плотность, кг/м3:	<input type="text" value="11.11"/>
Модуль упругости, ГПа:	<input type="text" value="10,01"/>
Количество отвердителя, м. %:	<input type="text" value="9.9"/>
Содержание эпоксидных групп:	<input type="text" value="8,8"/>
Температура вспышки, С_2:	<input type="text" value="7.7"/>
Поверхностная плотность, г/м2:	<input type="text" value="6,6"/>
Модуль упругости при растяжении, ГПа:	<input type="text" value="5.5"/>
Потребление смолы, г/м2:	<input type="text" value="4,4"/>
Угол нашивки, град:	<input type="text" value="3.3"/>
Шаг нашивки:	<input type="text" value="2,2"/>
Плотность нашивки:	<input type="text" value="1.1"/>

Прочность при растяжении, МПа

[0.00418093]

Для модели нейронной сети в коде убрал лишние скобки, т.к. высказивала ошибка размерности, не считая переименований и загружаемой модели:

```
def main3():
    if request.method == 'GET':
        return render_template('main3.html')
    if request.method == 'POST':
        loaded_model3 = pickle.load(open('model_neur_netw.pkl', 'rb'))
        features = []
        for i in range(1, 13):
            f = float(request.form[f'feature{i}'])
            features.append(f)
        y_pred = loaded_model3.predict([features])
        return render_template('main3.html', result=y_pred)

app3.run()

y_pred = loaded_model3.predict([features])
```

Прогноз нейросети для соотношения матрица-наполнитель

Плотность, кг/м3:	<input type="text" value="10.01"/>
Модуль упругости, ГПа:	<input type="text" value="20.02"/>
Количество отвердителя, м. %:	<input type="text" value="30.03"/>
Содержание эпоксидных групп:	<input type="text" value="40.04"/>
Температура вспышки, С_2:	<input type="text" value="50.05"/>
Поверхностная плотность, г/м2:	<input type="text" value="60.06"/>
Модуль упругости при растяжении, ГПа:	<input type="text" value="70.07"/>
Прочность при растяжении, МПа:	<input type="text" value="80.08"/>
Потребление смолы, г/м2:	<input type="text" value="90.09"/>
Угол нашивки, град:	<input type="text" value="100.001"/>
Шаг нашивки:	<input type="text" value="110.011"/>
Плотность нашивки:	<input type="text" value="120.012"/>

Соотношение матрица-наполнитель

[[0.00019304]]

2.6. Создание удаленного репозитория

Репозиторий, созданный в моём профиле на GitHub, находится по адресу:
<https://github.com/dkiselev82/educational-project>

3. Заключение

Проведённая работа позволяет сделать некоторые выводы. Распределение данных в датасете очень близко к нормальному, коэффициенты корреляции признаков стремятся к нулю. Используемые при обучении моделей методы не позволили получить достоверных предсказаний. Модели регрессии не показали высокой точности в предсказании целевых переменных. «Лучше» всего, показали себя методы машинного обучения: для модуля упругости при растяжении, ГПа – метод линейной регрессии; для прочности при растяжении, МПа – DummyRegressor.

Также невозможно определить из признаков материалов соотношение «матрица – наполнитель». Полученный результат не указывает на то, что предсказание признаков композитных материалов невозможно, возможно дело в недостатке данных и других возможных признаков, подходов, возможно пересмотра инструментов для предсказания целевых переменных.

Поэтому необходимы дополнительные данные, получение новых признаков, консультации экспертов в предметной области, новые исследования, эффективная работа команды, состоящей из различных экспертов и дополнительные опыты с данными материалами. Немного улучшить результат можно путём увеличения тренировочной выборки и уменьшения тестовой выборки, но значительных улучшений не произойдёт.

Предсказание целевых переменных в данном датасете не дало желаемых результатов. Самое главное из-за очень слабой корреляции между признаками. Поэтому проведя огромную работу с данными и применив довольно большое количество методов к ним, выявление лучших настраиваемых гиперпараметров, хорошего результата всё равно достигнуто не было. Задачу решить так, чтобы получить хороший результат на имеющихся данных не возможно ни одним из использованных методов.

4. Библиографический список

1. Материалы курса.
2. Силен Дэви, Мейсман Арно, Али Мохамед. Основы Data Science и Big Data. Python и наука о данных. – СПб.: Питер, 2017. 337 стр. – Режим доступа: <https://djvu.online/file/8ADC5FWnNyQ1v>
3. Andre Ye. 5 алгоритмов регрессии в машинном обучении, о которых вам следует знать: – Режим доступа: <https://habr.com/ru/company/vk/blog/513842/>
4. Краткий обзор алгоритма машинного обучения Метод Опорных Векторов (SVM) – Режим доступа: <https://habr.com/ru/post/428503/>
5. Документация по библиотеке pandas: – Режим доступа: https://pandas.pydata.org/docs/user_guide/index.html#user-guide
6. Документация по библиотеке numpy: – Режим доступа: <https://numpy.org/doc/1.22/user/index.html#user>
7. Документация по библиотеке matplotlib: – Режим доступа: <https://matplotlib.org/stable/users/index.html>
8. Документация по библиотеке seaborn: – Режим доступа: <https://seaborn.pydata.org/tutorial.html>
9. Документация по библиотеке sklearn: – Режим доступа https://scikit-learn.org/stable/user_guide.html
10. Документация по библиотеке keras: – Режим доступа: <https://keras.io/api/>
11. Документация по библиотеке Module: tf.keras: – Режим доступа: https://www.tensorflow.org/api_docs/python/tf/keras
12. Руководство по быстрому старту в flask: – Режим доступа: <https://flask-russian-docs.readthedocs.io/ru/latest/quickstart.html>