



**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(национальный исследовательский университет)»**

**Кафедра 316 «Системное моделирование и автоматизированное  
проектирование»**

## **КУРСОВАЯ РАБОТА**

**по дисциплине «Нейронные сети»**

**Тема: Решение задачи нелинейной регрессии с  
использованием XGboost**

Студент \_\_\_\_\_ **Брызгин Г.К.**

Группа \_\_\_\_\_ **МЗО-435Б-21**

Проверил \_\_\_\_\_ **Нагибин С.Я.**

Оценка \_\_\_\_\_ Дата защиты «\_\_» 2024 г.

**Москва 2024**



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(национальный исследовательский университет)»

Кафедра 316 «Системное моделирование и автоматизированное  
проектирование»

## ЗАДАНИЕ

на курсовую работу по дисциплине

«Нейронные сети»

Студент МЗО-435Б-21 Брызгин Георгий Константинович  
(№ группы, Ф. И. О.)

Тема Решение задачи нелинейной регрессии с использованием XGboost

### Рекомендуемая литература

1. С.Рашка Python и машинное обучение. Москва: ДМК Пресс, 2017.
2. П.Флах Машинное обучение: Наука и искусство построения алгоритмов, которые извлекают знания из данных. Москва: ДМК Пресс, 2015.
3. К.Элбон Машинное обучение с использованием Python. Сборник рецептов. СПб: БХВ-Петербург, 2019.
4. Ф.Шолле Глубокое обучение на Python. СПб: Питер, 2021.

Задание выдано «\_\_\_\_\_» октябрь 2024 г.

Проверил \_\_\_\_\_ Нагибин С.Я.  
(Ф. И. О., подпись)

Студент \_\_\_\_\_  
(подпись)

## СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ .....</b>	<b>4</b>
<b>1.Регрессия .....</b>	<b>5</b>
<b>1.1 Линейная регрессия .....</b>	<b>5</b>
<b>1.2 Нелинейная регрессия .....</b>	<b>5</b>
<b>2.Метрики регрессии .....</b>	<b>7</b>
<b>2.1 MSE.....</b>	<b>7</b>
<b>2.2 MAE.....</b>	<b>7</b>
<b>2.3 MAPE.....</b>	<b>8</b>
<b>2.4 SMAPE.....</b>	<b>8</b>
<b>XGBoost.....</b>	<b>9</b>
<b>3. Реализация решения задачи нелинейной регрессии с использованием XGBoost.....</b>	<b>12</b>
<b>3.1 Скачивание и EDA .....</b>	<b>12</b>
<b>3.2 Подготовка данных .....</b>	<b>17</b>
<b>3.3 Установка XGBoost и реализация модели нелинейной регрессии..</b>	<b>17</b>
<b>ЗАКЛЮЧЕНИЕ .....</b>	<b>22</b>
<b>СПИСОК ЛИТЕРАТУРЫ.....</b>	<b>23</b>
<b>ПРИЛОЖЕНИЕ.....</b>	<b>25</b>

## ВВЕДЕНИЕ

Благодаря машинному обучению программист не обязан писать инструкции, учитывающие все возможные проблемы и содержащие все решения. Вместо этого в компьютер (или отдельную программу) закладывают алгоритм самостоятельного нахождения решений путём комплексного использования статистических данных, из которых выводятся закономерности и на основе которых делаются прогнозы.

Машинное обучение - это область науки, которая занимается разработкой алгоритмов и моделей, которые позволяют компьютеру обучаться на данных и выполнять задачи, не требующие явного программирования. В рамках машинного обучения существует несколько типов задач, каждый из которых предназначен для решения определенной проблемы.

Одной из основных задач является регрессия - обучение на основе данных, содержащих числовые значения, с дальнейшим предсказанием новых числовых значений на основе этой информации.

Для оптимизации же самой модели нередко используют градиентный бустинг (комбинирование слабых функций, которые строятся в ходе итеративного процесса, где на каждом шаге новая модель обучается с использованием данных об ошибках предыдущих). Существует большое количество реализаций алгоритмов градиентного бустинга (XGBoost, CatBoost и т.д.) [9], однако рассматриваться будет только XGBoost [7].

XGBoost является гибким и удобным инструментом, т.к. поддерживает параллельное обучение, раннюю остановку и т.д.

В рамках данной курсовой работы необходимо было решить задачу нелинейной регрессии на выбранном датасете при помощи реализации XGBoost.

## 1.Регрессия

### 1.1 Линейная регрессия

Это один из простейших алгоритмов машинного обучения, описывающий зависимость целевой переменной от признака в виде линейной функции  $f_{w,b}(x) = w_0x_0 + w_1x_1 + \dots w_nx_n + b$ , где  $b$  – смещение модели,  $w$ -вектор её весов, а  $x$  – вектор признаков одного обучающего образца. Линейная регрессия используется для моделирования и анализа взаимосвязи между одной зависимой переменной и одной или несколькими независимыми переменными. Она помогает предсказать значение зависимой переменной на основе значений независимых переменных, предполагая линейную зависимость между ними.[10]

### 1.2 Нелинейная регрессия

Нелинейная регрессия - это способ нахождения нелинейной модели взаимосвязи между зависимой переменной и набором независимых переменных. В отличие от традиционной линейной регрессии, которая ограничена оценкой линейных моделей, нелинейная регрессия может оценивать модели с произвольными взаимосвязями между независимыми и зависимыми переменными. Это достигается при помощи итерационных алгоритмов оценки.

Преимущества нелинейной регрессии включают более точное моделирование сложных зависимостей между переменными и гибкость использования различных типов нелинейных функций для выбора наиболее подходящей модели. Формула нелинейной регрессии может иметь разный вид в зависимости от выбранной модели. Например, для полиномиальной регрессии, рассмотренной выше, она выглядит так:  $y = \beta_0 + \beta_1x + \beta_2x^2 + \dots + \beta_nx^n + \varepsilon$ , где  $y$  – зависимая переменная,  $x$  – независимая переменная,  $\beta_0, \beta_1, \dots, \beta_n$ -коэффициенты модели,  $\varepsilon$  – ошибка. Однако все эти преимущества не делают модель крайне эффективной, не используя дополнительные алгоритмы оптимизации.[6]

К основным недостаткам нелинейной регрессии можно отнести[11]:

- Чувствительность к шуму: Нелинейные модели могут быть более чувствительными к шуму в данных. Это может привести к нестабильности и снижению точности прогнозов
- Вычислительная сложность: Обучение нелинейных моделей часто требует больших вычислительных ресурсов и времени. Это особенно актуально для методов, таких как градиентный бустинг или нейронные сети.
- Переобучение: Переобучение приводит к тому, что модель хорошо работает на тренировочных данных, но плохо обобщает на новые данные
- Выбор параметров: Настройка параметров нелинейной регрессии требует значительных вычислительных ресурсов и времени. Неправильный выбор параметров может привести к плохой производительности модели.

## 2. Метрики регрессии

Метрики позволяют количественно оценить степень соответствия предсказанных значений фактическим данным, что является критически важным для разработки и улучшения прогнозных моделей[12].

### 2.1 MSE

Одна из самых популярных метрик в задаче регрессии:

$$MSE(y^{true}, y^{pred}) = \frac{1}{n} \sum_{i=1}^N (y_i - f(x_i))^2.$$

MSE неограничен сверху, и может быть нелегко понять, насколько «хорошим» или «плохим» является то или иное его значение. Чтобы появились какие-то ориентиры, делают следующее:

Берут наилучшее константное предсказание с точки зрения MSE — среднее арифметическое меток  $\bar{y}$ . При этом чтобы не было подглядывания в test, среднее нужно вычислять по обучающей выборке

Рассматривают в качестве показателя ошибки

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - f(x_i))^2}{\sum_{i=1}^N (y_i - \bar{y})^2}.$$

У идеально решающего правила  $R^2$  равен 1, у наилучшего константного предсказания он равен 0 на обучающей выборке. MSE квадратично штрафует за большие ошибки на объектах.

### 2.2 MAE

Средняя абсолютная ошибка (MAE) в нелинейной регрессии — это метрика, которая измеряет среднюю величину ошибок между предсказанными и фактическими значениями. Она вычисляется как среднее арифметическое абсолютных разностей между этими значениями. Чем меньше значение MAE, тем точнее модель. Формула MAE:

$$MAE(y^{true}, y^{pred}) = \frac{1}{N} \sum_{i=1}^N |y_i - f(x_i)|$$

### 2.3 MAPE

Средняя абсолютная процентная ошибка (MAPE) в нелинейной регрессии — это метрика, которая измеряет среднюю величину ошибок между предсказанными и фактическими значениями в процентах. Она помогает оценить точность модели, показывая, насколько предсказания модели отклоняются от реальных данных в среднем в процентном выражении. Чем меньше значение MAPE, тем точнее модель

$$MAPE(y^{true}, y^{pred}) = \frac{1}{N} \sum_{i=1}^N \frac{|y_i - f(x_i)|}{|y_i|}$$

### 2.4 SMAPE

Симметричная средняя абсолютная процентная ошибка (SMAPE) в нелинейной регрессии - это метрика, которая измеряет точность предсказаний модели. Используется для оценки точности модели, особенно в случаях, когда важно учитывать симметрию ошибок. Чем меньше значение SMAPE, тем точнее модель

$$SMAPE(y^{true}, y^{pred}) = \frac{1}{N} \sum_{i=1}^N \frac{2|y_i - f(x_i)|}{y_i + f(x_i)}$$



## XGBoost

XGBoost - это контролируемый метод машинного обучения для классификации и регрессии, который используется в инструменте Обучение с использованием AutoML. XGBoost - это сокращение от "экстремального повышения градиента". Этот метод основывается на дереве решений и работает лучше, чем другие методы, такие как произвольное дерево и градиентное повышение. Он лучше работает со сложными большими наборами данных, используя различные методы оптимизации.[7]

XGBoost обеспечивает высокую производительность благодаря своим преимуществам над другими реализациями:

- Регуляризация - параметр регуляризации (лямбда) используется при расчете показателей подобия, чтобы снизить чувствительность к отдельным данным и избежать переобучения.
- Сокращение — параметр сложности дерева (гамма) выбирается для сравнения выигрышей. Ветвь, где коэффициент усиления меньше значения гаммы, удаляется. Это предотвращает избыточную подгонку за счет обрезки ненужных ветвей и уменьшения глубины деревьев.
- Скetch взвешенных квантилей — вместо проверки каждого возможного значения в качестве порога для разделения данных используются только взвешенные квантили. Выбор квантилей осуществляется с помощью алгоритма скетча, который оценивает распределение по нескольким системам в сети.
- Параллельное обучение — этот метод делит данные на блоки, которые можно использовать параллельно для создания деревьев или других вычислений.
- Поиск разделения с учетом разреженности — XGBoost обрабатывает разреженность данных, пробуя оба направления в разбиении и находя направление по умолчанию, вычисляя усиление.

- Доступ с учетом кэша — этот метод использует кэш-память системы для расчета показателей сходства и выходных значений. Кэш-память является более быстрой памятью для доступа по сравнению с основной памятью и повышает общую производительность модели.
- Блоки для вычислений вне ядра - этот метод работает с большими наборами данных, которые не помещаются в кэш или основную память и должны храниться на жестких дисках. Этот набор данных разбивается на блоки и сжимается. Несжатые данные в основной памяти работают быстрее, чем при чтении с жесткого диска. Другой метод, называемый сегментированием, используется, когда данные должны храниться на нескольких жестких дисках.[15]

XGBoost отлично подходит для решения задач нелинейной регрессии, т.к. он учитывает сложные нелинейные зависимости между признаками и отлично оптимизирован.

Основной функцией оптимизации градиентного бустинга представлена на рисунке 1

$$\mathcal{L}^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

Рисунок 1 – Функция оптимизации градиентного бустинга

Где  $l$  – функция потерь,  $y_i, \hat{y}_i^t$  – значение  $i$ -го элемента обучающей выборки и сумма предсказанных первых  $t$  деревьев соответственно,  $x_i$  – набор признаков  $i$  – го элемента обучающей выборки,  $f_i$  – функция, которую необходимо обучить на шаге  $t$ ,  $f_t(x_i)$  – предсказание на  $i$ -ом элементе обучающей выборки,  $\Omega(f)$ -регуляризация функции  $f$ .  $\Omega(f) = \gamma T + 12\lambda \|\omega\|_2$ , где  $T$  — количество вершин в дереве,  $\omega$  — значения в листьях, а  $\gamma$  и  $\lambda$  — параметры регуляризации.

Дальше с помощью разложения Тейлора до второго члена можем приблизить оптимизируемую функцию  $L^{(t)}$  следующим выражением:

$$L^{(t)} = \sum_{i=1}^n l\left(y_i, \hat{y}_i^{t-1} + g_i f_t(x_i) + 0.5 h_i f_t^2(x_i)\right) + \Omega(f_t), \text{ где}$$

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}}, h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{t-1})}{\partial^2 \hat{y}_i^{t-1}}$$

Т.к. необходимо минимизировать ошибку на обучающей выборке – ищем минимум  $L^{(t)}$  для каждого  $t$ .

Минимум этого выражения относительно  $f_t(x_i)$  находится в точке  $f_t(x_i) = \frac{-g_i}{h_i}$ . Каждое отдельное дерево ансамбля  $f_t(x_i)$  обучается стандартным алгоритмом.

### 3. Реализация решения задачи нелинейной регрессии с использованием XGBoost

#### 3.1 Скачивание и EDA

Первым этапом был подбор данных, корректных для решения задачи. На листинге 1 продемонстрирован алгоритм скачивания и отображения содержимого этого алгоритма. Выбраны были два датасета – качество красного вина и белого. Соединим их в один датасет, ведь столбцы их идентичны

##### Листинг 1 – Скачивание и отображение набора данных

```
!kaggle datasets download uciml/red-wine-quality-cortez-et-al-2009
!unzip /content/red-wine-quality-cortez-et-al-2009.zip -d /content
!rm /content/red-wine-quality-cortez-et-al-2009.zip
!kaggle datasets download piyushagni5/white-wine-quality
!unzip /content/white-wine-quality.zip -d /content
!rm /content/white-wine-quality.zip
data = pd.read_csv('/content/winequality-red.csv')
data.head()
```

Далее на рисунках 2 - 8 будет представлена основная информация о данном наборе данных

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Рисунок 2 – Содержимое датасета

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   fixed acidity          1599 non-null   float64
1   volatile acidity       1599 non-null   float64
2   citric acid            1599 non-null   float64
3   residual sugar         1599 non-null   float64
4   chlorides              1599 non-null   float64
5   free sulfur dioxide    1599 non-null   float64
6   total sulfur dioxide   1599 non-null   float64
7   density               1599 non-null   float64
8   pH                    1599 non-null   float64
9   sulphates              1599 non-null   float64
10  alcohol                1599 non-null   float64
11  quality                1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
None

```

Рисунок 3 – Информация о датасете

	fixed acidity	volatile acidity	citric acid	residual sugar	\
count	1599.000000	1599.000000	1599.000000	1599.000000	
mean	8.319637	0.527821	0.270976	2.538806	
std	1.741096	0.179060	0.194801	1.409928	
min	4.600000	0.120000	0.000000	0.900000	
25%	7.100000	0.390000	0.090000	1.900000	
50%	7.900000	0.520000	0.260000	2.200000	
75%	9.200000	0.640000	0.420000	2.600000	
max	15.900000	1.580000	1.000000	15.500000	

	chlorides	free sulfur dioxide	total sulfur dioxide	density	\
count	1599.000000	1599.000000	1599.000000	1599.000000	
mean	0.087467	15.874922	46.467792	0.996747	
std	0.047065	10.460157	32.895324	0.001887	
min	0.012000	1.000000	6.000000	0.990070	
25%	0.070000	7.000000	22.000000	0.995600	
50%	0.079000	14.000000	38.000000	0.996750	
75%	0.090000	21.000000	62.000000	0.997835	
max	0.611000	72.000000	289.000000	1.003690	

	pH	sulphates	alcohol	quality
count	1599.000000	1599.000000	1599.000000	1599.000000
mean	3.311113	0.658149	10.422983	5.636023
std	0.154386	0.169507	1.065668	0.807569
min	2.740000	0.330000	8.400000	3.000000
25%	3.210000	0.550000	9.500000	5.000000
50%	3.310000	0.620000	10.200000	6.000000
75%	3.400000	0.730000	11.100000	6.000000
max	4.010000	2.000000	14.900000	8.000000

Рисунок 4 – Описание основных статистических характеристик

```

fixed acidity      0
volatile acidity   0
citric acid        0
residual sugar     0
chlorides          0
free sulfur dioxide 0
total sulfur dioxide 0
density           0
pH                0
sulphates          0
alcohol            0
quality            0
dtype: int64

```

Рисунок 5 – Проверка на наличие пропущенных значений

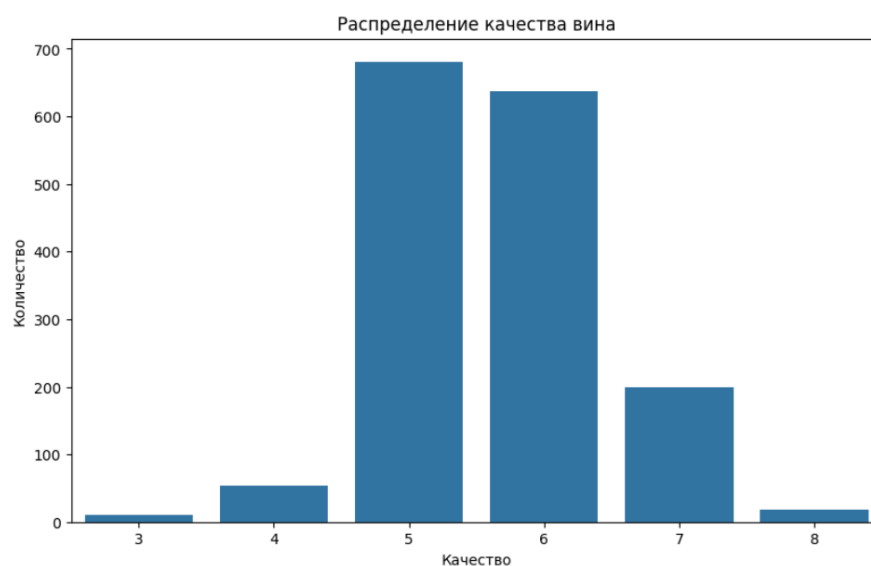


Рисунок 6 – Распределение качества вина

Показывает общее количество вина того или иного качества

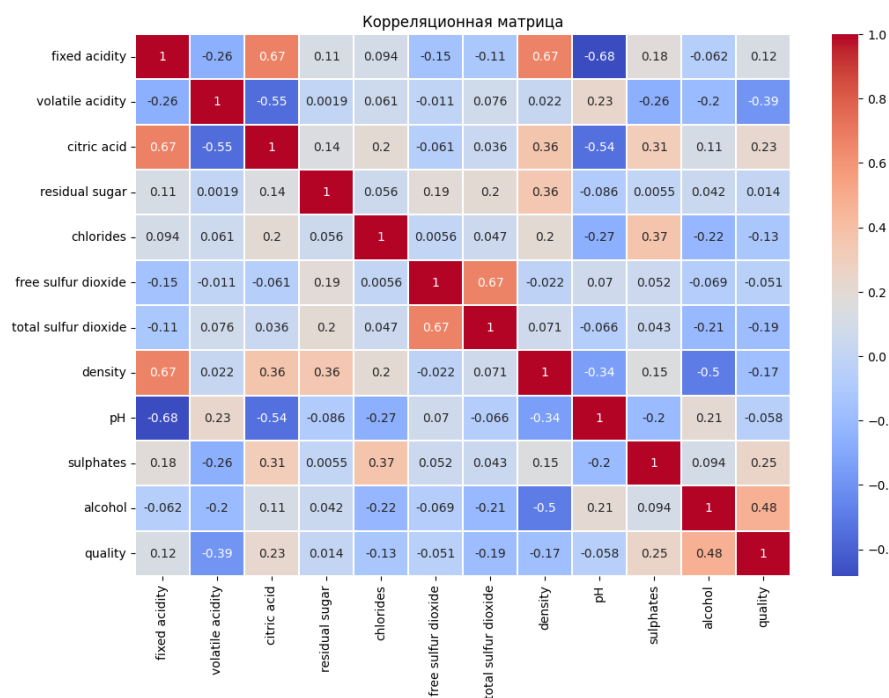


Рисунок 7 – Корреляционная матрица

Корреляционная матрица показывает взаимосвязь между несколькими переменными в наборе данных, т.е. как к примеру зависит качество алкоголя от процентного содержания спирта. Значения, которые могут быть в матрице:

- +1 – идеальная корреляция(переменные напрямую зависят друг от друга)
- -1 – идеальная обратная корреляция(переменные зависят друг от друга прям пропорционально)
- 0 – переменные вообще не зависят друг от друга

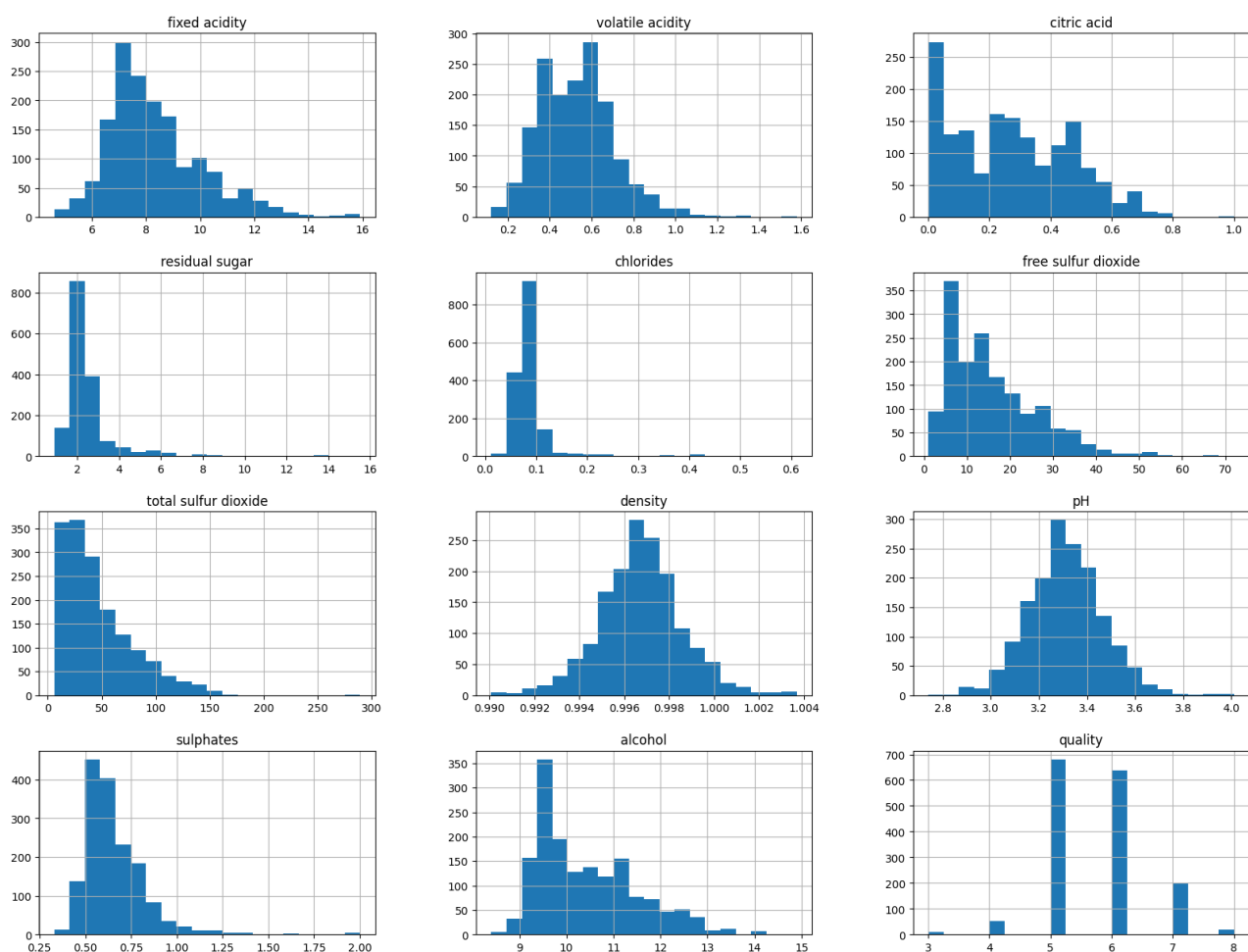


Рисунок 8 – Распределение химического состава вина

Данный график показывает в каком диапазоне лежат те или иные переменные во всем датасэте. К примеру, Citric Acid в основном достаточно в малых диапазонах, т.е. в большинстве вин его содержание мало.



После того, как был произведен первичный EDA(разведочный анализ данных), можно «предподготовить» сами данные.

### **3.2 Подготовка данных**

Сначала необходимо соединить два датасэта, выделить целевую переменную и отделить признаки(см.Листинг 2)

#### **Листинг 2 – Отделение переменных**

```
full_data = pd.concat([data, test_data], ignore_index=True)
y = full_data ['quality']
x = full_data.drop('quality', axis=1)
```

После, разделим dataset красного вина на тренировочную выборку и валидационную(см.Листинг 3).

Листинг 3 – Разделение набора данных на тренировочный и валидационный

```
x_train, x_val, y_train, y_val = train_test_split(x, y,
test_size=0.2, random_state=42)
```

Далее нормализуем данные(см.Листинг 4). Т.к. набор не нормально распределен – используем MinMaxScaler вместо StandartScaler, он преобразует признаки, масштабируя каждый признак до заданного диапазона

#### **Листинг 4 – Нормализация данных**

```
norm = MinMaxScaler(feature_range = (0, 1))
norm.fit(x_train)
x_train = norm.transform(x_train)
x_val = norm.transform(x_val)
x_test = norm.transform(x_test)
```

Теперь данные полностью предподготовлены и можно начинать построение самой модели, её обучение и оценку

### **3.3 Установка XGBoost и реализация модели нелинейной регрессии**

Для установки самой библиотеки необходимо использовать команду  
`!pip install xgboost.`

Далее «собираем» уже саму модель, настраиваем гиперпараметры и запускаем обучение(см.Листинг 5)

## Листинг 5 – Модель нелинейной регрессии, реализованная при помощи

xgboost

```
xgb_reg = xgb.XGBRegressor(  
    objective='reg:squarederror',  
    n_estimators=1000,  
    learning_rate=0.1,  
    max_depth=4,  
    min_child_weight=5,  
    reg_alpha=0.1,  
    reg_lambda=1.0,  
    random_state=42,  
    early_stopping_rounds=50,  
    eval_metric='rmse'  
)  
xgb_reg.fit(  
    x_train, y_train,  
    eval_set=[(x_val, y_val)],  
    verbose=True  
)  
y_pred_val = xgb_reg.predict(x_val)  
y_pred_test = xgb_reg.predict(x_test)
```

Теперь определим метрики, которых нет в стандартной библиотеке, а именно: MAPE и SMAPE. Код представлен на листинге 6

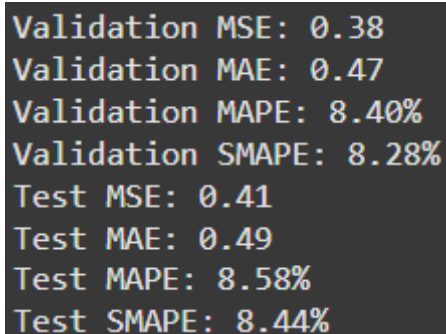
### Листинг 6 – Определение MAPE

```
def mean_absolute_percentage_error(y_true, y_pred):  
    y_true, y_pred = np.array(y_true), np.array(y_pred)  
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

### Листинг 7 – Определение SMAPE

```
def symmetric_mean_absolute_percentage_error(y_true, y_pred):  
    y_true, y_pred = np.array(y_true), np.array(y_pred)  
    return np.mean(2 * np.abs(y_true - y_pred) / (np.abs(y_true)  
+ np.abs(y_pred))) * 100
```

Далее просто смотрим результаты по метрикам(см.рис.9)



```
Validation MSE: 0.38  
Validation MAE: 0.47  
Validation MAPE: 8.40%  
Validation SMAPE: 8.28%  
Test MSE: 0.41  
Test MAE: 0.49  
Test MAPE: 8.58%  
Test SMAPE: 8.44%
```

Рисунок 9 – Результаты по метрикам

Неплохие результаты, однако, можно попробовать подобрать гиперпараметры так, чтобы результаты стали еще лучше. Сделать это можно при помощи встроенной в библиотеку sklearn функции GridSearchCV - метод поиска наилучших гиперпараметров для модели машинного обучения. Он систематически проверяет все комбинации заданных гиперпараметров и оценивает производительность модели для каждой из них, используя кросс-валидацию. На листинге 8 продемонстрирован код для поиска наилучших гиперпараметров.

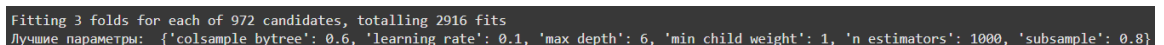
#### Листинг 8 – Поиск наилучших гиперпараметров

```
param_grid = {
    'n_estimators': [100, 500, 1000],
    'max_depth': [3, 4, 5, 6],
    'learning_rate': [0.01, 0.1, 0.2],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}

grid_search = GridSearchCV(
    estimator=xgb.XGBRegressor(),
    param_grid=param_grid,
    scoring='neg_mean_absolute_error',
    cv=3,
    verbose=1
)

grid_search.fit(x_train, y_train)
print("Лучшие параметры: ", grid_search.best_params_)
```

В результате, после запуска поиска, мы получим результат, продемонстрированный на рисунке 10.



```
Fitting 3 folds for each of 972 candidates, totalling 2916 fits
Лучшие параметры: {'colsample_bytree': 0.6, 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 1000, 'subsample': 0.8}
```

#### Рисунок 10 – Лучшие гиперпараметры

Теперь попробуем их применить, вставив в модель, и обучив её по новой. Новый результат представлен на рисунке 11.

```
Validation MSE: 0.35  
Validation MAE: 0.44  
Validation MAPE: 7.79%  
Validation SMAPE: 7.65%  
Test MSE: 0.39  
Test MAE: 0.46  
Test MAPE: 8.13%  
Test SMAPE: 7.97%
```

Рисунок 11 – Результаты обучения с лучшими гиперпараметрами

Для наглядности – на рисунке 12 продемонстрированы старые и новые результаты обучения

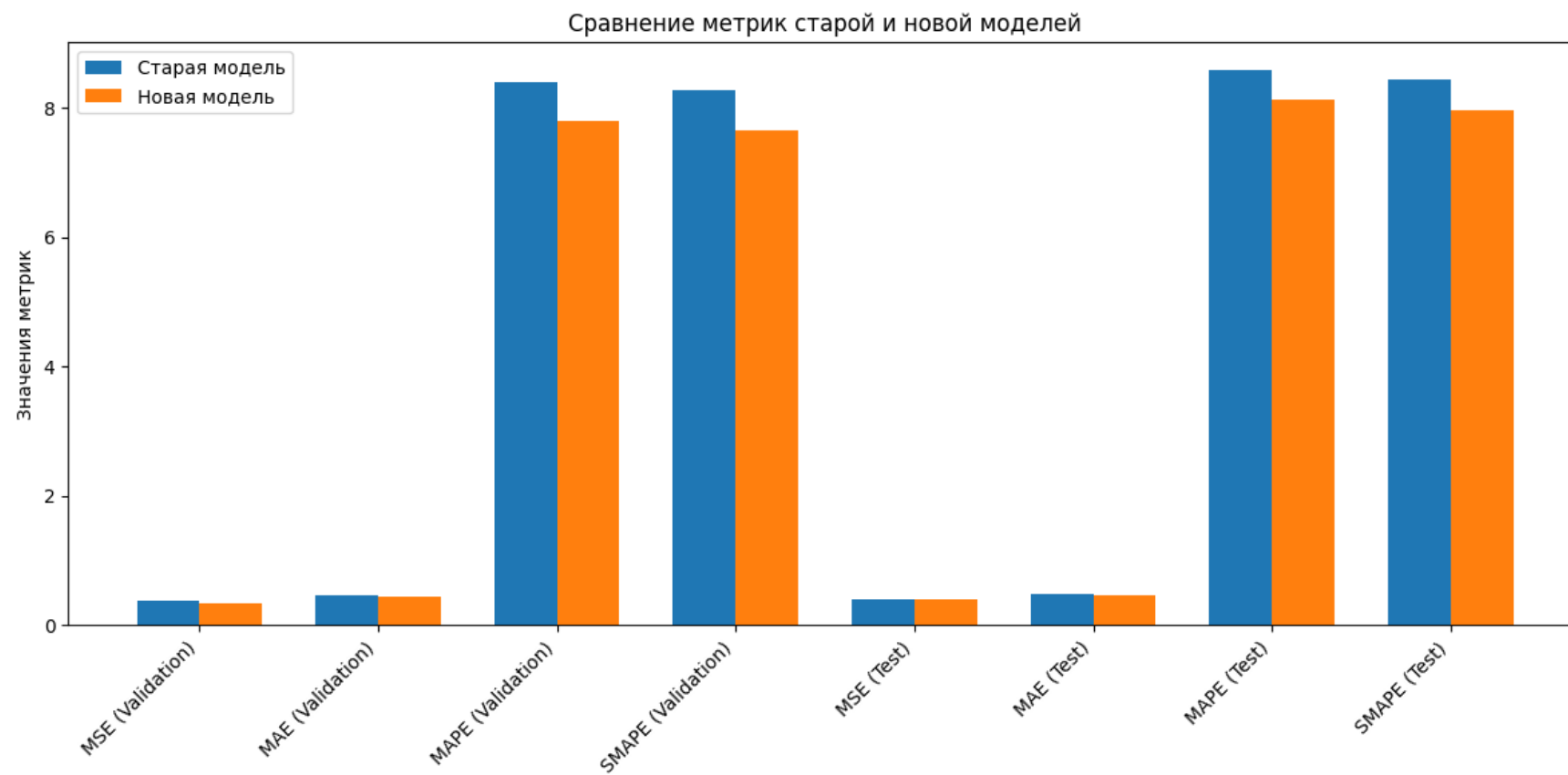


Рисунок 12 – Сравнение старых и новых результатов

Таким образом, наглядно видно, что с изменением гиперпараметров модели – качество модели возросло, т.к. ошибки стали.

## **ЗАКЛЮЧЕНИЕ**

Использование XGBoost для решения задачи нелинейной регрессии является эффективным подходом, позволяющим достигать высоких результатов при анализе и предсказании данных в задачах регрессии. Благодаря гибкости настройки, XGBoost позволяет адаптировать модель под специфику данных, что улучшает точность предсказаний. Поддержка множества гиперпараметров дала возможность оптимизировать модель для достижения лучших результатов.. Этот алгоритм эффективно обрабатывает пропущенные значения, что делает модель устойчивой к неполной информации. Это важное преимущество, особенно при работе с реальными наборами данных, где пропуски часто встречаются. Включение L1 и L2 регуляризации помогло предотвратить переобучение модели, улучшив её обобщающую способность и стабильность предсказаний. Механизмы ранней остановки, поддерживаемые данной библиотекой, позволили прекратить обучение модели, когда дальнейшие итерации не приводили к значительным улучшениям, что также способствовало предотвращению переобучения.

Высокая точность предсказаний XGBoost была подтверждена метриками, такими как MSE, MAE, MAPE и SMAPE на тренировочных, валидационных и тестовых наборах данных.

XGBoost также включает механизмы ранней остановки, что помогает предотвращать переобучение и повышает общую стабильность модели. Эти преимущества демонстрируют, что данный алгоритм является мощным и надежным инструментом для решения задач нелинейной регрессии, позволяющим достигать высоких результатов при анализе и предсказании данных.

## СПИСОК ЛИТЕРАТУРЫ

1. С.Рашка Python и машинное обучение. Москва: ДМК Пресс, 2017.
2. П.Флах Машинное обучение: Наука и искусство построения алгоритмов, которые извлекают знания из данных. Москва: ДМК Пресс, 2015
3. К.Элбон Машинное обучение с использованием Python. Сборник рецептов. СПб: БХВ-Петербург, 2019.
4. Ф.Шолле Глубокое обучение на Python. СПб: Питер, 2021.
5. Машинное обучение // Хабр URL: [https://habr.com/ru/hubs/machine\\_learning/articles/](https://habr.com/ru/hubs/machine_learning/articles/) (дата обращения: 06.11.2024).
6. Нелинейная регрессия // MachineLearning.ru URL: [http://www.machinelearning.ru/wiki/index.php?title=%D0%9D%D0%B5%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F\\_%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D1%8F](http://www.machinelearning.ru/wiki/index.php?title=%D0%9D%D0%B5%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F_%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D1%8F) (дата обращения: 07.11.2024).
7. XGBoost // MediaWiki:ITMO URL: <https://neerc.ifmo.ru/wiki/index.php?title=XGBoost> (дата обращения: 06.11.2024).
8. XGBoost в машинном обучении // BI CONSULT URL: <https://datafinder.ru/products/xgboost-v-mashinnom-obuchenii> (дата обращения: 10.11.2024).
9. Градиентный бустинг. Реализация с нуля на Python и разбор особенностей его модификаций (XGBoost, CatBoost, LightGBM) // habr URL: <https://habr.com/ru/articles/799725/> (дата обращения: 10.12.2024).
10. Линейная регрессия на Python: объясняем на пальцах // proglib URL: <https://proglib.io/p/linear-regression> (дата обращения: 10.12.2024).
11. Нелинейные методы регрессии: изучение сложности отношений // FasterCapital URL: <https://fastercapital.com/ru/content/ru/content/Нелинейные-методы-регрессии--изучение-сложности-отношений.html> (дата обращения: 12.12.2024).

12.Простыми словами про метрики в ИИ. Регрессия. MSE, RMSE, MAE, R-квадрат, MAPE // Habr URL: <https://habr.com/ru/articles/820499/> (дата обращения: 01.12.2024).

13.XGBoost Documentation // DMLC SGBOST URL: <https://xgboost.readthedocs.io/en/stable/> (дата обращения: 12.12.2024).



## ПРИЛОЖЕНИЕ

### Приложение А – Код программы

```
!pip install pandas-profiling seaborn plotly scipy termcolor
lightgbm kaggle xgboost
!kaggle datasets download uciml/red-wine-quality-cortez-et-al-
2009
!unzip /content/red-wine-quality-cortez-et-al-2009.zip -d
/content
!rm /content/red-wine-quality-cortez-et-al-2009.zip
!kaggle datasets download piyushagni5/white-wine-quality
!unzip /content/white-wine-quality.zip -d /content
!rm /content/white-wine-quality.zip
!rm /content/winequality.names
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
from sklearn.model_selection import train_test_split,
GridSearchCV, cross_val_score
from sklearn.metrics import mean_squared_error,
mean_absolute_error
import xgboost as xgb
data = pd.read_csv('/content/winequality-white.csv', sep=";")
test_data = pd.read_csv('/content/winequality-red.csv')
full_data = pd.concat([data, test_data], ignore_index=True)
y = full_data['quality']
x = full_data.drop('quality', axis=1)
x_train, x_temp, y_train, y_temp = train_test_split(x, y,
test_size=0.3, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp,
test_size=0.5, random_state=42)
norm = MinMaxScaler(feature_range = (0, 1))
norm.fit(x_train)
x_train = norm.transform(x_train)
x_val = norm.transform(x_val)
x_test = norm.transform(x_test)
#{'colsample_bytree': 0.6, 'learning_rate': 0.1, 'max_depth': 6,
'min_child_weight': 1, 'n_estimators': 1000, 'subsample': 0.8}
xgb_reg = xgb.XGBRegressor(
    objective='reg:squarederror',
    n_estimators=1000,
    learning_rate=0.1,
    max_depth=4,
    min_child_weight=5,
    reg_alpha=0.1,
    reg_lambda=1.0,
    random_state=42,
```

```

        early_stopping_rounds=50,
        eval_metric='rmse'

    )
    xgb_reg.fit(
        x_train, y_train,
        eval_set=[(x_val, y_val)],
        verbose=True
    )
    y_pred_val = xgb_reg.predict(x_val)
    y_pred_test = xgb_reg.predict(x_test)

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def symmetric_mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(2 * np.abs(y_true - y_pred) / (np.abs(y_true)
+ np.abs(y_pred))) * 100

old_mse_val = mean_squared_error(y_val, y_pred_val)
old_mae_val = mean_absolute_error(y_val, y_pred_val)
old_mape_val = mean_absolute_percentage_error(y_val, y_pred_val)
old_smape_val = symmetric_mean_absolute_percentage_error(y_val,
y_pred_val)

print(f"Validation MSE: {old_mse_val:.2f}")
print(f"Validation MAE: {old_mae_val:.2f}")
print(f"Validation MAPE: {old_mape_val:.2f}%")
print(f"Validation SMAPE: {old_smape_val:.2f}%")
old_mse_test = mean_squared_error(y_test, y_pred_test)
old_mae_test = mean_absolute_error(y_test, y_pred_test)
old_mape_test = mean_absolute_percentage_error(y_test,
y_pred_test)
old_smape_test =
symmetric_mean_absolute_percentage_error(y_test, y_pred_test)

print(f"Test MSE: {old_mse_test:.2f}")
print(f"Test MAE: {old_mae_test:.2f}")
print(f"Test MAPE: {old_mape_test:.2f}%")
print(f"Test SMAPE: {old_smape_test:.2f}%")

mse_val = mean_squared_error(y_val, y_pred_val)
mae_val = mean_absolute_error(y_val, y_pred_val)
mape_val = mean_absolute_percentage_error(y_val, y_pred_val)
smape_val = symmetric_mean_absolute_percentage_error(y_val,
y_pred_val)
print(f"Validation MSE: {mse_val:.2f}")
print(f"Validation MAE: {mae_val:.2f}")
print(f"Validation MAPE: {mape_val:.2f}%")

```

```

print(f"Validation SMAPE: {smape_val:.2f}%")
mse_test = mean_squared_error(y_test, y_pred_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
mape_test = mean_absolute_percentage_error(y_test, y_pred_test)
smape_test = symmetric_mean_absolute_percentage_error(y_test,
y_pred_test)

print(f"Test MSE: {mse_test:.2f}")
print(f"Test MAE: {mae_test:.2f}")
print(f"Test MAPE: {mape_test:.2f}%")
print(f"Test SMAPE: {smape_test:.2f}%")
param_grid = {
    'n_estimators': [100, 500, 1000],
    'max_depth': [3, 4, 5, 6],
    'learning_rate': [0.01, 0.1, 0.2],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}
grid_search = GridSearchCV(
    estimator=xgb.XGBRegressor(),
    param_grid=param_grid,
    scoring='neg_mean_absolute_error',
    cv=3,
    verbose=1
)
grid_search.fit(x_train, y_train)
print("Лучшие параметры: ", grid_search.best_params_)
new_results = [mse_val, mae_val, mape_val, smape_val, mse_test,
mae_test, mape_test, smape_test]
old_results = [old_mse_val, old_mae_val, old_mape_val,
old_smape_val, old_mse_test, old_mae_test, old_mape_test,
old_smape_test]
metrics = ['MSE (Validation)', 'MAE (Validation)', 'MAPE
(Validation)', 'SMAPE (Validation)',
           'MSE (Test)', 'MAE (Test)', 'MAPE (Test)', 'SMAPE
(Test)']

x, width = np.arange(len(metrics)), 0.35
fig, ax = plt.subplots(figsize=(12, 6))
rects1 = ax.bar(x - width/2, old_results, width, label='Старая
модель')
rects2 = ax.bar(x + width/2, new_results, width, label='Новая
модель')

ax.set_ylabel('Значения метрик')
ax.set_title('Сравнение метрик старой и новой моделей')
ax.set_xticks(x)
ax.set_xticklabels(metrics, rotation=45, ha='right')
ax.legend()
fig.tight_layout()
plt.show()

```