Задание: Прогнозирование качества вин

Отчет по проделанной работе

Выполнил: Белоконев Даниил Артемович

ТЗ:https://docs.google.com/document/d/1ZH_IT6bjXPwsW5Fmzf5zNsqLZWrSO0f7/edit?usp=sharing&ouid=117231516583900503940&rtpof=true&sd=true

Результаты проделанной работы

Скачал и распаковал данные, после чего ознакомился с их содержанием

White wines

5 rows × 13 columns

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	dens
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	

Red wines

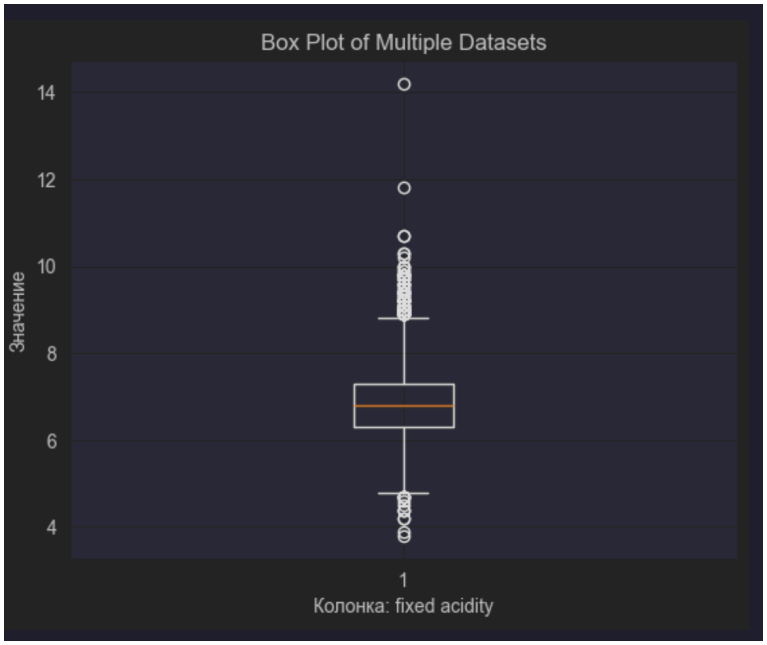
5 rows × 13 columns

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	dens
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	

Обозрел датасет на наличие выбросов (аномалий) или Null-значений. Для этого использовать стандартные инструменты:

- Pandas: для поиска Null-значений
- Matplotlib: для анализа данных на наличие выбросов с графиков распределения и boxplot-графиков
- Scipy для вычисления z score для каждого атрибута (подсчета количества выбросов в среди атрибутов) Считал, что z score выше 3 - это выброс
- Seaborn: для визуализация матрицы корреляции признаков

Boxplot-график (один из) и информация о пропущенных значениях

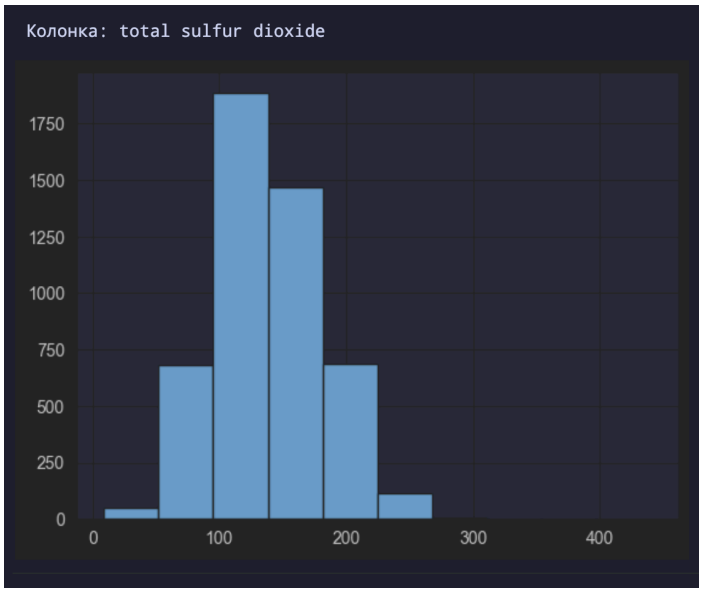



```
None
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
wine type               0
dtype: int64

\n Red wines\n<class 'pandas.core.frame.DataFrame'>\n

None
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
wine type               0
dtype: int64
```

График распределения (один из) и информация о количестве выбросов за счет показателя z score




```
Колонка: fixed acidity, количество выбросов: 46
Колонка: volatile acidity, количество выбросов: 81
Колонка: citric acid, количество выбросов: 85
Колонка: residual sugar, количество выбросов: 9
Колонка: chlorides, количество выбросов: 102
Колонка: free sulfur dioxide, количество выбросов: 32
Колонка: total sulfur dioxide, количество выбросов: 12
Колонка: density, количество выбросов: 3
Колонка: pH, количество выбросов: 32
Колонка: sulphates, количество выбросов: 48
Колонка: alcohol, количество выбросов: 0
```

Как можно заметить, данные и правда имеют выбросы, но бывают и та

Подготовил данные:

- Разделил и стандартизировал данные
- Превратил pd.DataFrame и pd.Series в tensors, которые впоследствии были объединены в dataloaders

Создал архитектуру нейросети:

```python
# Нейросеть для решения регрессионных задач
class WineRegressor(nn.Module):
    def __init__(self, input_size, hidden_sizes, dropout=0.3):
        super().__init__()
        layers = []
        sizes = [input_size] + hidden_sizes

        # Для каждого перехода sizes[i] в sizes[i+1]
        for i in range(len(sizes) - 1):
            # линейное преобразование
            layers.append(nn.Linear(sizes[i], sizes[i + 1]))
            # нормализация для ускорения и стабилизации обучения
            layers.append(nn.BatchNorm1d(sizes[i + 1]))
            # функция активации (в отличие от стандартной, такая версия снижает вероятность умирания нейронов)
            layers.append(nn.LeakyReLU(0.01))
            # регуляризация, которое борется с переобучением, не давая нейросети зацикливаться на каких-то определенных нейронах
            layers.append(nn.Dropout(dropout))

        # последний линейный слой для составления нужного выходного размера
        layers.append(nn.Linear(sizes[-1], 1))

        # сборка нейросети
        self.net = nn.Sequential(*layers)
        self._initialize_weights()

    def _initialize_weights(self):
        # инициализация для всех линейных слоев
        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.xavier_uniform_(m.weight)

                # обнуляет отклонение bias
                if m.bias is not None:
                    nn.init.zeros_(m.bias)

    def forward(self, x):
        return self.net(x).squeeze(-1)
```

Базовая нейросеть с Feedforward структурой: линейные слои + нормализация + нелинейность + регуляризация.

- **Линейные слои**: для преобразования данных и извлечения из них более богатого представления, которые помогают модели лучше понять данные
- **Нормализация** (Batch Norm): для ускорения и стабилизации обучения
- **Функция активация** (Leaky ReLU): используется данная функция активации, чтобы снизить вероятность “отмирания” нейронов
- **Регуляризация** (Dropout): борется с переобучением, не давая нейросети зацикливаться на определенных нейронах

- Последний слой: преобразует выходные данные нейросети под нужный размер (в нашем случае требуется непрерывное число, тк решаем задачу регрессии)
- Инициализация весов (Xavier): устанавливает начальные веса так, чтобы дисперсия выходов градиентов сохранялась одинаковой во всех слоях

Данная нейросеть обучается по стандартному циклу градиентного спуска:

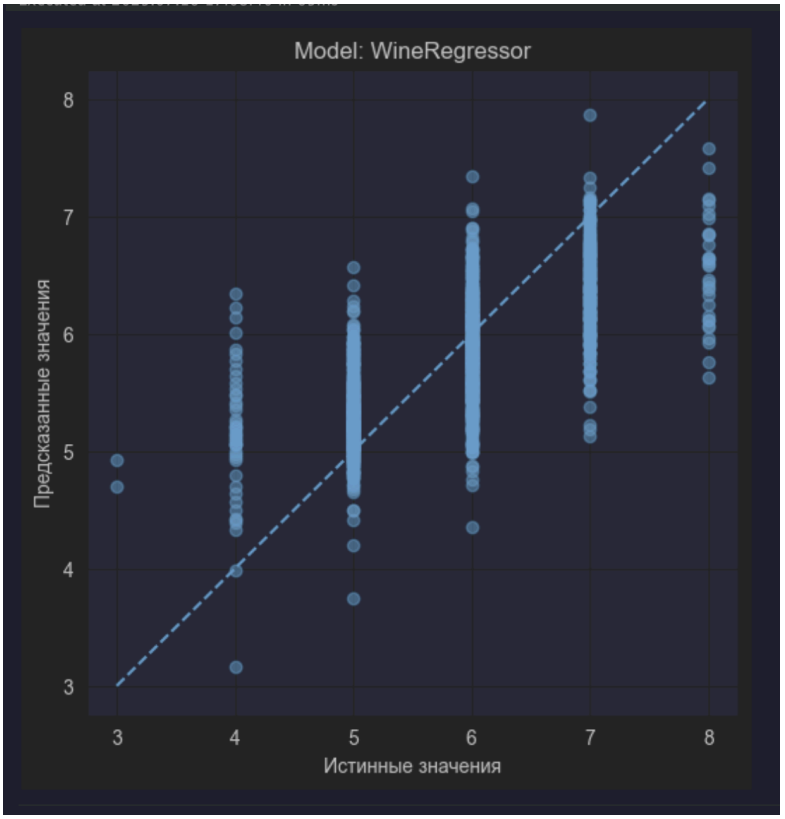
1) Обнуляем градиенты параметров нейросети для того, чтобы градиенты не складывались и не искажали обновление весов
2) Forward pass - прогон входных данных через нейросеть
3) Вычисляем функцию потерь (в нашем случае используем MSELoss)
4) backpropagation - обратное распространения ошибки (вычисляем градиенты ошибки по каждому параметру нейросети, данные градиенты сохраняются внутри каждого параметра)
5) Исходя из вычисленных градиентов и значения learning rate, делаем шаг градиентного спуска - обновляем веса нейросети (У нас в качестве оптимизатора используется Adam)
6) Данный цикл повторяется для каждого батча и эпохи

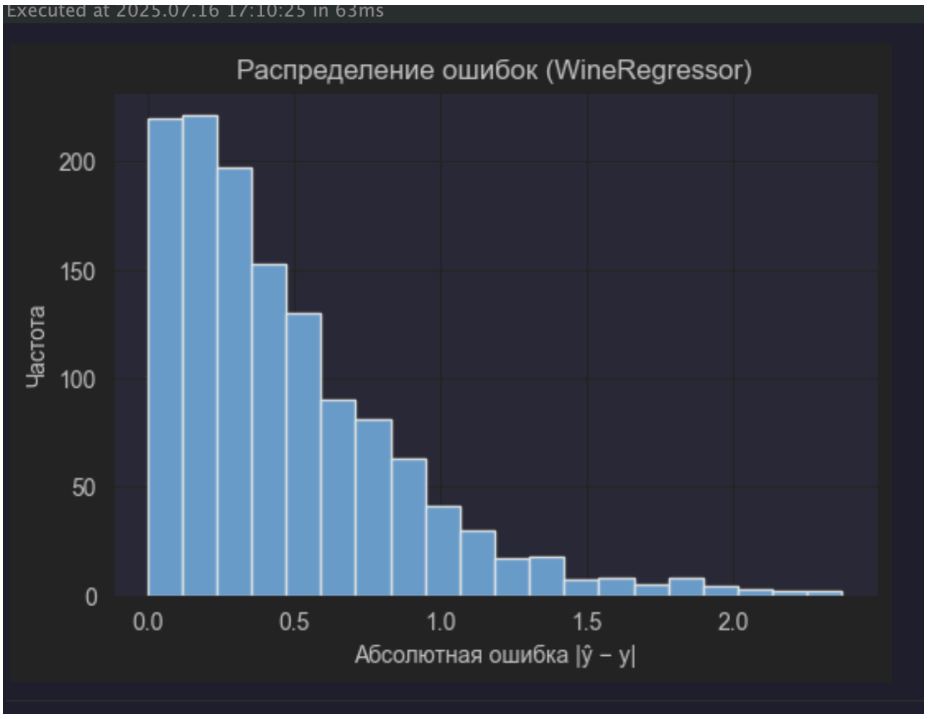
Нейросеть была сначала проверена с помощью инструмента optuma. Данный инструмент позволил нам подобрать оптимальные параметры для нашей модели. На основе полученных параметров обучили нейросеть.

Нейросеть обучалась 300 эпох, в конечном итоге на тестовой выборке обученная нейросеть показала результат MAE = 0.4767

```
Epoch 292 | Train Loss: 0.1776 | Val MAE: 0.4725 | LR: 1.50e-06
Epoch 293 | Train Loss: 0.1799 | Val MAE: 0.4753 | LR: 1.50e-06
Epoch 294 | Train Loss: 0.1757 | Val MAE: 0.4716 | LR: 1.50e-06
Epoch 295 | Train Loss: 0.1772 | Val MAE: 0.4724 | LR: 1.00e-06
Epoch 296 | Train Loss: 0.1764 | Val MAE: 0.4746 | LR: 1.00e-06
Epoch 297 | Train Loss: 0.1758 | Val MAE: 0.4721 | LR: 1.00e-06
Epoch 298 | Train Loss: 0.1753 | Val MAE: 0.4739 | LR: 1.00e-06
Epoch 299 | Train Loss: 0.1766 | Val MAE: 0.4726 | LR: 1.00e-06
Epoch 300 | Train Loss: 0.1746 | Val MAE: 0.4767 | LR: 1.00e-06
```

В качестве визуализации результатов работы использовались графики scatter plot граф, на котором показано как близко предсказанные классы лежат к идеальной диагонали + гистограмма распределения абсолютных ошибок, на которой показано насколько часто модель ошибается на N-единиц





Исходя из показаний двух графиков можно сказать:

1) Модель неплохо предсказывает средние классы (5, 6, 7), но ей очень сложно даются крайние классы (впоследствии мы выясним, что крайних классов как раз-таки крайне мало, и предложим варианты исправления данной проблемы)
2) Модель преимущественно имеет наиболее частые предсказания, которые укладываются в ошибку меньше 0.6, что хорошо в задачах с ограниченной шкалой, но далеко от идеала

Какие улучшения можно сделать для данной модели

1) Обогатить датасет: датасет насчитывает порядка 6 тысяч записей, что является недостаточным количеством для обучения полноценных нейросетей (особенно мало крайних классов, таких как 3, 4, 8 и 9). Это даст нейросети больше информации для обучения, а также поможет ей более точно понять и предсказывать крайние классы.
2) Изменить датасет: одним из возможных решений может стать объединение крайних классов (например, 3 и 4 или 8 и 9). Такое решение освободит нейросеть от поиска сложных зависимостях в данных, которых крайне мало, что должно облегчить работу для нее
3) SMOTE обогащение или аугментация: можно воспользоваться инструментов генерации синтетических данных, что уравновесит баланс классов и даст модели большее количество данных, с которыми можно работать
4) Feature Engineering: важным этапом в оптимизации модели является feature engineering.
    a) Можно убрать низко-коррелирующие атрибуты (модель не будет путаться бесполезными данными, а сфокусируется лишь на самых полезных)
    b) Можно вывести новые атрибуты путем группировки ч, преобразования или бинаризации некоторых фич
5) Пересмотр архитектуры:
    a) Можно изменить слои существующей нейросети
    b) Можно пересмотреть задачу и переделать нашу модель под задачу классификации
    c) Можно отказаться от столь тяжелого и сложного варианта, в пользу ансамблей или других моделей классического машинного обучения (например, Random Forest или XGBoosting)

Данные предложения не являются 100% методами повышения точности модели, поэтому очень важно всегда тестировать и проверять новую модель, на наличие улучшений.

Более подробное описание работы можно посмотреть в notebook!