

Лабораторная работа №8.

Полиномиальная регрессия

Цель лабораторной работы: научиться применять разработанный пайплайн для тиражирования кода с целью решения задачи полиномиальной регрессии.

Основные задачи:

- получение навыков рефакторинга кода в проектах машинного обучения;
- изучение поведения модели полиномиальной регрессии при изменении степени полинома;
- исследование свойств набора данных в рамках задачи полиномиальной регрессии.

Ход выполнения индивидуального задания:

1. Построение модели полиномиальной регрессии

1.1 Загрузка и визуализация набора данных

Первым делом импортируем все необходимые библиотеки для выполнения лабораторной работы, а также загрузим наш набор данных с помощью **pandas** и выведем первые пять строк:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

data_path = "https://archive.ics.uci.edu/ml/machine-learning-
databases/forest-fires/forestfires.csv"
```

```
data = pd.read_csv(data_path)

print(data.head())
```

Как результат получаем следующий вывод:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

Проведём **предобработку** нашего набора данных. А именно: **удалим аномалии** (нулевые и слишком большие значения) из целевой переменной `area`, а также применим **логарифмическое преобразование** для **улучшения** работы алгоритмов машинного обучения:

```
data['area'] = np.log1p(data['area'])
data = data[(data['area'] > 0) & (data['area'] < 50)].copy()
data.reset_index(drop=True, inplace=True)

print(data.head())
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	9	9	jul	tue	85.8	48.3	313.4	3.9	18.0	42	2.7	0.0	0.3015
1	1	4	sep	tue	91.0	129.5	692.6	7.0	21.7	38	2.2	0.0	0.357674
2	2	5	sep	mon	90.9	126.5	686.5	7.0	21.9	39	1.8	0.0	0.385262
3	1	2	aug	wed	95.5	99.9	513.3	13.2	23.3	31	4.5	0.0	0.438255
4	8	6	aug	fri	90.1	108.0	529.8	12.5	21.2	51	8.9	0.0	0.476234

Разделим наши данные на **матрицу признаков** и **зависимую переменную**, после чего выведем первые 10 значений на экран:

```
X = data.drop('area', axis=1)
y = data['area']

print("\nМатрица признаков")
print(X[:10])
print("\nЗависимая переменная")
print(y[:10])
```

Матрица признаков													
	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	9	9	jul	tue	85.8	48.3	313.4	3.9	18.0	42	2.7	0.0	0.3015
1	1	4	sep	tue	91.0	129.5	692.6	7.0	21.7	38	2.2	0.0	0.357674
2	2	5	sep	mon	90.9	126.5	686.5	7.0	21.9	39	1.8	0.0	0.385262
3	1	2	aug	wed	95.5	99.9	513.3	13.2	23.3	31	4.5	0.0	0.438255
4	8	6	aug	fri	90.1	108.0	529.8	12.5	21.2	51	8.9	0.0	0.476234
5	1	2	jul	sat	90.0	51.3	296.3	8.7	16.6	53	5.4	0.0	0.3015
6	2	5	aug	wed	95.5	99.9	513.3	13.2	23.8	32	5.4	0.0	0.385262
7	6	5	aug	thu	95.2	131.7	578.8	10.4	27.4	22	4.0	0.0	0.385262
8	5	4	mar	mon	90.1	39.7	86.6	6.2	13.2	40	5.4	0.0	0.3015
9	8	3	sep	tue	84.4	73.4	671.9	3.2	24.2	28	3.6	0.0	0.3015

Зависимая переменная

0	0.307485
1	0.357674
2	0.385262
3	0.438255
4	0.476234
5	0.536493
6	0.570980
7	0.641854
8	0.667829
9	0.672944

1.2 Обработка категориальных признаков

В нашем наборе данных имеется два **категориальных признака**, которые необходимо обработать и привести в **удобный** для машинного обучения вид. Для этого воспользуемся такими средствами как

`ColumnTransformer` и `OneHotEncoder`:

```
ct = ColumnTransformer(
    transformers=[
        ("encoder", OneHotEncoder(drop='first'), [2, 3])
    ],
    remainder="passthrough"
)

X_encoded = ct.fit_transform(X)

print("\nМатрица признаков после обработки категориальных признаков:")
print(X_encoded[:5])
```

Матрица признаков после обработки категориальных признаков:

[0.	0.	0.	1.	0.	0.	0.	0.	0.	0.	0.	2.7
[0.	1.	0.	9.	9.	85.8	48.3	313.4	3.9	18.	42.	2.2
[0.	0.	0.	0.	0.	0.	0.	0.	1.	0.	0.	1.8
[0.	0.	0.	0.	0.	0.	0.	0.	1.	1.	0.	4.5
[1.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	8.9

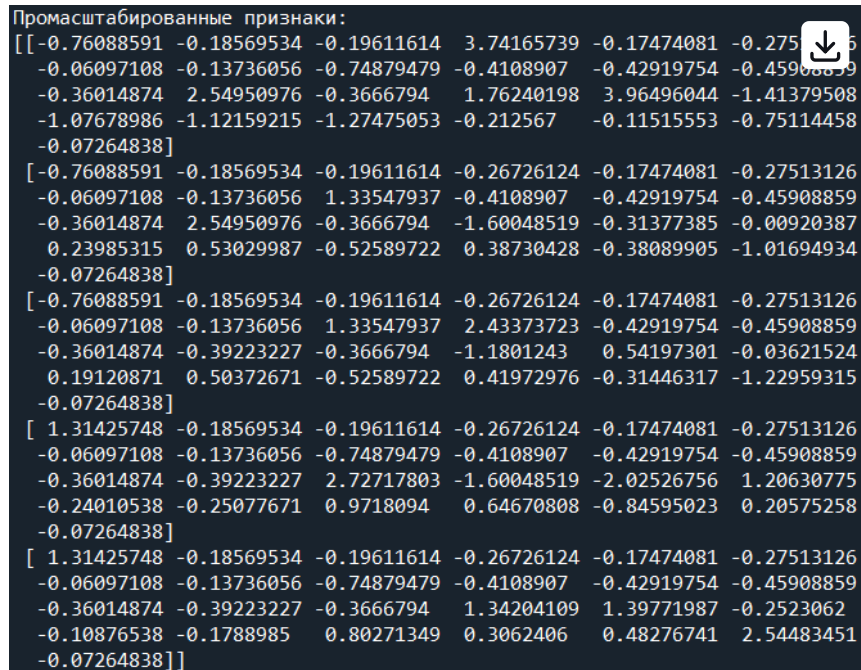
1.3 Масштабирование признаков

При помощи класса `StandardScaler` мы масштабируем признаки таким образом, чтобы их **среднее значение** стало **0**, а **стандартное отклонение** — **1**. Это помогает избежать доминирования признаков с большими значениями и улучшает сходимость модели.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)
print("\nПромасштабированные признаки:")
print(X_scaled[:5])
```

```
print("\n-----
-----")
```

Промасштабированные признаки:



```
[[-0.76088591 -0.18569534 -0.19611614  3.74165739 -0.17474081 -0.27513126
 -0.06097108 -0.13736056 -0.74879479 -0.4108907  -0.42919754 -0.45908859
 -0.36014874  2.54950976 -0.3666794  1.76240198  3.96496044 -1.41379508
 -1.07678986 -1.12159215 -1.27475053 -0.212567   -0.11515553 -0.75114458
 -0.07264838]
 [-0.76088591 -0.18569534 -0.19611614 -0.26726124 -0.17474081 -0.27513126
 -0.06097108 -0.13736056  1.33547937 -0.4108907  -0.42919754 -0.45908859
 -0.36014874  2.54950976 -0.3666794 -1.60048519 -0.31377385 -0.00920387
  0.23985315  0.53029987 -0.52589722  0.38730428 -0.38089905 -1.01694934
 -0.07264838]
 [-0.76088591 -0.18569534 -0.19611614 -0.26726124 -0.17474081 -0.27513126
 -0.06097108 -0.13736056  1.33547937  2.43373723 -0.42919754 -0.45908859
 -0.36014874 -0.39223227 -0.3666794 -1.1801243  0.54197301 -0.03621524
  0.19120871  0.50372671 -0.52589722  0.41972976 -0.31446317 -1.22959315
 -0.07264838]
 [ 1.31425748 -0.18569534 -0.19611614 -0.26726124 -0.17474081 -0.27513126
 -0.06097108 -0.13736056 -0.74879479 -0.4108907  -0.42919754 -0.45908859
 -0.36014874 -0.39223227  2.72717803 -1.60048519 -2.02526756  1.20630775
 -0.24010538 -0.25077671  0.9718094  0.64670808 -0.84595023  0.20575258
 -0.07264838]
 [ 1.31425748 -0.18569534 -0.19611614 -0.26726124 -0.17474081 -0.27513126
 -0.06097108 -0.13736056 -0.74879479 -0.4108907  -0.42919754 -0.45908859
 -0.36014874 -0.39223227 -0.3666794  1.34204109  1.39771987 -0.2523062
 -0.10876538 -0.1788985  0.80271349  0.3062406  0.48276741  2.54483451
 -0.07264838]]
```

1.4 Разделение выборки на тестовую и тренировочную. Обучение модели

Разделим выборку на **тестовую** и **тренировочную**, используя функцию

`train_test_split` :

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=42)
```

Теперь нашей задачей является **обучение модели полиномиальной регрессии** на разных значениях **степени**. В нашем случае будем сравнивать работу модели на степенях **от 1 до 4**. По сути, **полиномиальная модель** это **модель линейной регрессии**, которая обучается на **полиномиальных признаках**. Для этого сначала **преобразуем** признаки в полиномиальные с помощью `PolynomialFeatures`. Уже после преобразования мы можем обучать модель линейной регрессии:

```
degrees = [1, 2, 3, 4]

for i, degree in enumerate(degrees, 1):
    poly_reg = PolynomialFeatures(degree=degree)
    X_train_poly = poly_reg.fit_transform(X_train)
    X_test_poly = poly_reg.transform(X_test)

    lin_reg = LinearRegression()
```

```
lin_reg.fit(X_train_poly, y_train)
y_test_pred = lin_reg.predict(X_test_poly)
```

1.5 Визуализация результатов

Теперь построим графики зависимости **предсказанных значений** от **фактических** с **линией регрессии**: для каждой модели с разной степенью:

```
degrees = [1, 2, 3, 4]

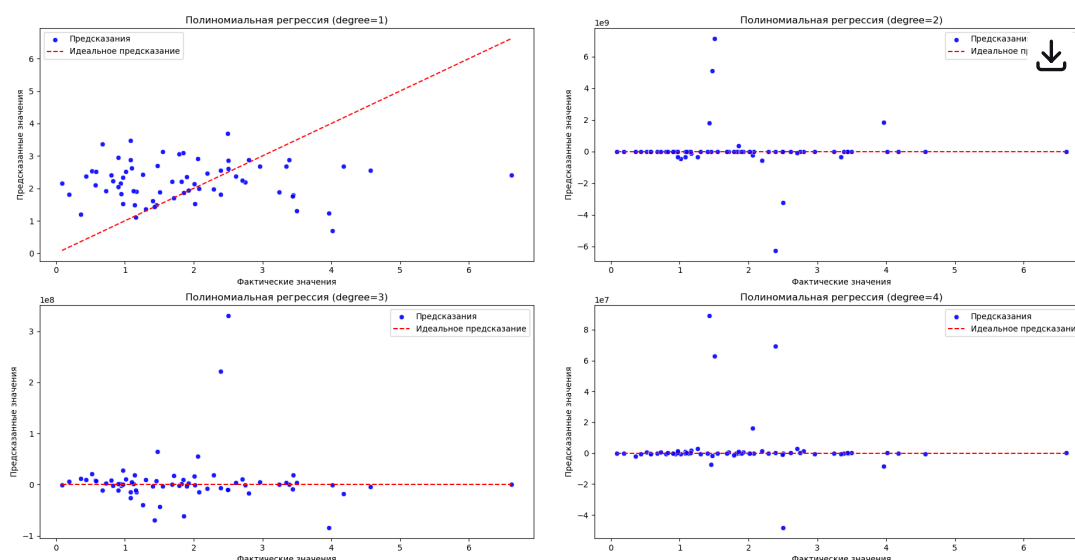
plt.figure(figsize=(12, 10))

for i, degree in enumerate(degrees, 1):
    poly_reg = PolynomialFeatures(degree=degree)
    X_train_poly = poly_reg.fit_transform(X_train)
    X_test_poly = poly_reg.transform(X_test)

    lin_reg = LinearRegression()
    lin_reg.fit(X_train_poly, y_train)
    y_test_pred = lin_reg.predict(X_test_poly)

    plt.subplot(2, 2, i)
    sns.scatterplot(x=y_test, y=y_test_pred, alpha=0.9, color='blue',
label="Предсказания")
    plt.plot([y_test.min(), y_test.max()], [y_test.min(),
y_test.max()], 'r--', label="Идеальное предсказание")
    plt.xlabel('Фактические значения')
    plt.ylabel('Предсказанные значения')
    plt.title(f'Полиномиальная регрессия (degree={degree})')
    plt.legend()

plt.tight_layout()
```



⚠ Сначала может показаться, что линии идеального предсказания не совпадают на разных графиках. На самом деле они

одинаковые, просто из-за разных масштабов графиков они кажутся непохожими друг на друга.

Контрольные вопросы

1. Этот признак используется для того, чтобы учесть *свободный член* (*intercept*) в модели, то есть константу β_0 . В линейной регрессии модель может быть записана как:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

Чтобы включить свободный член в уравнение, мы добавляем фиктивную переменную с постоянным значением 1.0. Это позволяет модели правильно интерпретировать влияние константы на зависимую переменную.

2. Фиктивная переменная (или дамми-переменная) — это бинарная переменная, принимающая значения 0 или 1, которая используется для представления категориальных признаков в числовом виде. Например, если у нас есть категориальный признак "Цвет", который принимает значения "красный", "синий", "зеленый", мы можем создать три фиктивные переменные: одна для каждого цвета. Однако, чтобы избежать мультиколлинеарности (ситуации, когда переменные линейно зависимы), обычно удаляется одна из фиктивных переменных. Это делается, чтобы избежать избыточности и не нарушить модель.
3. Полиномиальная регрессия создаётся с использованием комбинации двух классов из библиотеки sklearn:
 - **PolynomialFeatures** (из sklearn.preprocessing) — для создания полиномиальных признаков.
 - **LinearRegression** (из sklearn.linear_model) — для построения линейной модели на этих преобразованных данных.

Полиномиальная регрессия — это всё та же линейная регрессия, но на расширенном наборе признаков.

4. При полиномиальной регрессии входные признаки X трансформируются путём добавления их степенных комбинаций. Если у нас есть один признак X , то при `degree=2` преобразование создаст:

$$[1, X, X^2]$$

Если у нас несколько признаков, то для `degree=2` получится:

$$[1, X_1, X_2, X_1^2, X_2^2, X_1 X_2]$$

Полиномы более высоких степеней включают **кубические, четвёртые и т. д. степени**, а также их **комбинации**. Таким образом, полиномиальные признаки позволяют модели **учитывать нелинейные зависимости**.

5. **Да, масштабирование признаков не только возможно, но и рекомендуется.** Полиномиальная регрессия приводит к созданию новых признаков, которые могут сильно различаться по величине. Это может **ухудшить сходимость градиентных методов** и сделать модель **чувствительной к выбросам**.