МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
"ЛЭТИ" ИМ. В.И.УЛЬЯНОВА(ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе № 3
по дисциплине «Основы машинного обучения»
Тема: «Регрессия».

Студент гр.1304 _____ Поршнев Р.А.
Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Задание

1. Линейная регрессия.

1.1. Загрузите набор данных соответствующей цифре вашего варианта. Убедитесь, что загрузка прошла корректно.

1.2. Используя train_test_split разбейте выборку на обучающую и тестовую. Проверьте, что тестовая выборка соответствует обучающей. Можно испльзовать диаграммы рассеяния/нормированные гистограммы/boxplot/violin plot.

1.3. Проведите линейную регрессию используя LinearRegression. Получите коэффициенты регрессии и объясните полученные результаты.

1.4. Для обучающей и тестовой выборки рассчитайте коэффициент детерминации, MAPE, MAE. Объясните полученные значений метрик. Сравните метрики для обучающей и тестовой выборки, сделайте выводы о качестве обобщения полученной модели.

1.5. Повторите пункты 1.3 и 1.4 для модификаций линейной регрессии:

1.5.1. Лассо регрессия. Самостоятельно подберите гиперпараметры.

1.5.2. Гребневая регрессия. Самостоятельно подберите гиперпараметры.

1.5.3. ElasticNet. Самостоятельно подберите гиперпараметры.

1.5.4. Регрессия оптимизируемая градиентным спуском/

1.6. Постройте сводную таблицу для рассчитываемых метрик, и используемых методов (с разделением на обучающую и тестовую выборку). По таблице сделайте выводы о том, какой вид регрессии дал лучшую модель. Опишите какие проблемы могут возникнуть при применении каждой модели. Для объяснения результатов можно построить “карту высот (heat map)” с отображением корреляции признаков.

1.7. Для модели, которая дала лучшие результаты, постройте диаграмму рассеяния между предикторами и откликом. На диаграмме изобразите какое значение должно быть, и какое предсказывается. Визуально оцените качестве построенного регрессора.

2. Нелинейная регрессия.

2.1. Загрузите набор данных соответствующей букве вашего варианта. Убедитесь, что загрузка прошла корректно.

2.2. Используя train_test_split разбейте выборку на обучающую и тестовую. Проверьте, что тестовая выборка соответствует обучающей.

2.3. Проверьте работу стандартной линейной регрессии на загруженных данных. Постройте диаграмму рассеяния данных с выделенной полученной линией регрессии. Объясните полученный результат.

2.4. Конструируя полиномиальный признаки для разных степеней полинома найдите степень полинома наилучшим образом аппроксимирующая данные. Постройте график зависимости коэффициента детерминации от степени полинома(на одном графике изобразите линии для обучающей и тестовой выборки отдельно). Сделайте вывод о том, при какой степени полинома модель начинает переобучаться.

2.5. Для выбранной степени полинома, рассчитайте и проанализируйте полученные коэффициенты. Рассчитай значение метрик коэффициент детерминации, MAPE, MAE.

2.6. Для выбранной степени полинома, постройте диаграмму рассеяния данных с линией соответствующей полученному полиному. Сделайте выводы о качестве аппроксимации.

2.7. Для выбранной степени полинома, решите задачу нелинейной регрессии без конструирования полиномиальных признаков и используя библиотеку TensorFlow.

2.8. Рассчитай метрики коэффициент детерминации, MAPE, MAE, а также постройте диаграмму рассеяния для данных линией соответствующей полученному полиному, для модели полученной при помощи TensorFlow.

2.9. Сравните результаты полученные с/без конструирования полиномиальных признаков.

3. Оценка модели регрессии.

3.1. Загрузите набор данных Student_Performance.csv . Данный набор данных содержит информацию о характеристиках студента, а также качестве его обучения.

3.2. Проведите предобработку набора данных - замена текстовых данных, удаление null значений, удаление дубликатов. Разделите на обучающую и тестовую выборку.

3.3. Постройте модель, которая будет предсказывать значение признака Performance Index на основе остальных признаков. Модель выберите самостоятельно.

3.4. Проанализируйте полученную модель. Сделайте выводы о значимости/-информативности признаков. Опишите какие проблемы могут возникнуть при применении модели.

Выполнение работы.

Вариант 4Б.

1. Линейная регрессия.

1.1. Загрузите набор данных соответствующей цифре вашего варианта. Убедитесь, что загрузка прошла корректно.

Для загрузки набора данных необходимо импортировать библиотеку pandas и воспользоваться методом read_csv с передачей параметра пути датасета. Убедиться в корректности данных можно с помощью метода head, который выводит первые пять записей датасета. Также для последующих методов и функций, использующих параметр random_state, инициализирована переменная RS, равная 42. Реализация данного фрагмента кода представлена в Листинге 1.1.1, результат его выполнения – в Листинге 1.1.2.

Листинг 1.1.1 – Загрузка данных из файла как Pandas DataFrame

```
import pandas as pd


RS = 42
lin4 = pd.read_csv('sample_data/lab3_lin4.csv')
lin4.head()
```

Листинг 1.1.2 – Результат вывода первых пяти записей датафрейма

	x1	x2	y
0	1.1414	-0.0594	55.2686
1	-0.1517	-1.3385	-50.3732
2	-2.8140	0.1431	-145.5692
3	-0.6642	0.5702	-16.0423
4	-0.2069	-0.1609	-16.1340

Регрессия чувствительная к масштабу данных, поэтому нужно провести либо стандартизацию, либо нормализацию. Распространено мнение, что для регрессии стандартизация более предпочтительна, но чтобы появилось больше уверенности

в выборе варианта масштабирования, следует провести тест Шапиро-Уилка. Реализация соответствующей функции и её вызова представлены в Листинге 1.1.3, результат вызова – в Листинге 1.1.4.

Листинг 1.1.3 – Реализация теста Шапиро-Уилка

```
from scipy.stats import shapiro


def shapiro_test(data):
    columns = data.columns
    for column in columns:
        pvalue = shapiro(data[column]).pvalue
        print(f"column: {column}\tpvalue: {pvalue}")

shapiro_test(lin4.iloc[:, :-1])
```

Листинг 1.1.4 – Результат теста Шапиро-Уилка

```
column: x1      pvalue: 0.38795149326324463
column: x2      pvalue: 0.6063389778137207
```

Если задать $\alpha = 0.05$, то можно сделать вывод, что факторы имеют стандартное распределение и следует использовать стандартизацию (см. Листинг 1.1.5).

Листинг 1.1.5 – Реализация стандартизации данных

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler


lin4_np = StandardScaler().fit_transform(lin4)
lin4 = pd.DataFrame(data=lin4_np.T, index=['x1', 'x2', 'y']).T
```

1.2. Используя train_test_split разбейте выборку на обучающую и тестовую. Проверьте, что тестовая выборка соответствует обучающей. Можно испльзовать диаграммы рассеяния/нормированные гистограммы/boxplot/violin plot.

Для реализации разбиения выборки на обучающую и тестовую следует воспользоваться функцией train_test _split из sklearn.model_selection, которая принимает на вход массивы, которые нужно разбить, размер тестовой выборки относительно всех данных, а также значение параметра random_state, который нужен

для генерации одних и тех же выборок при разбиении данных. Для проверки разбиения на обучающую и тестовую выборки выбран анализ диаграммы рассения. Для отрисовки диаграмм необходимо создать холст, на которым будут отрисованы диаграммы рассения по каждому признаку, и отрисовать сами диаграммы рассеяния. Для отрисовки данного и последующих графиков будут использованы такие библиотеки, как matplotlib и seaborn. Реализация разбиения датасета и отрисовки диаграмм рассения представлена в Листинге 1.2.1, диаграммы рассеяния обучающей и тестовой выборок – на Рисунке 1.2.1.

Листинг 1.2.1 – Реализация разбиения датасета на обучающую и тестовую выборки и отрисовки диаграммы рассения

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(lin4.iloc[:, :-1],
                                                    lin4['y'],
                                                    test_size=0.3,
                                                    random_state=RS)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
sns.scatterplot(x=X_train['x1'], y=y_train, ax=axes[0],
                label='Обучающая выборка')
sns.scatterplot(x=X_test['x1'], y=y_test, ax=axes[0], c='orange',
                label='Тестовая выборка')
sns.scatterplot(x=X_train['x2'], y=y_train, ax=axes[1],
                label='Обучающая выборка')
sns.scatterplot(x=X_test['x2'], y=y_test, ax=axes[1], c='orange',
                label='Тестовая выборка')
fig.suptitle('Диаграммы рассеяния обучающей и тестовой выборок по признакам')
plt.show()
```

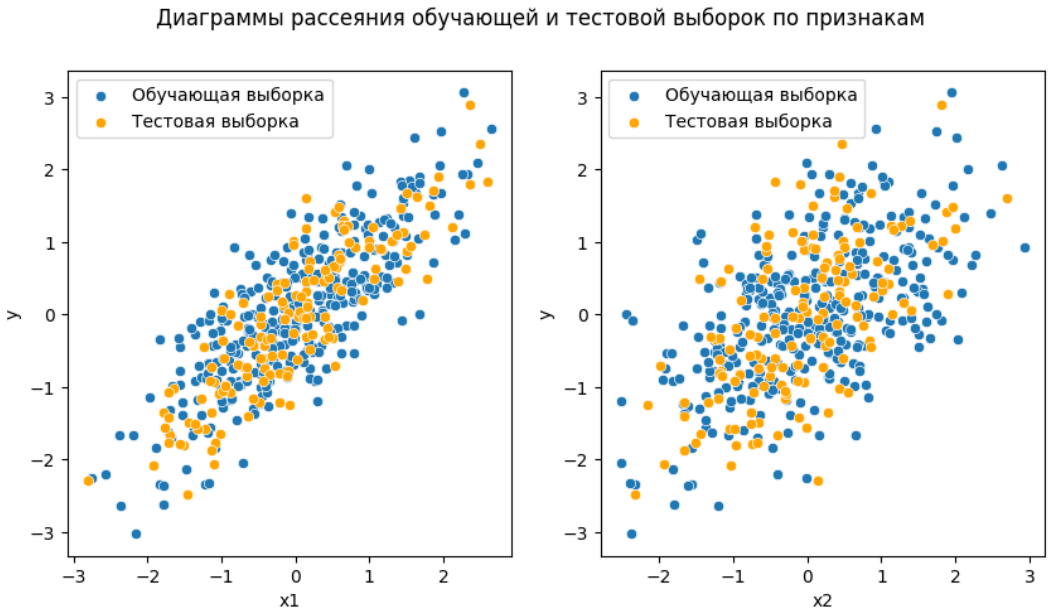

Рисунок 1.2.1 – Диаграммы рассеяния обучающей и тестовой выборок

Исходя из диаграмм рассеяния, можно сделать вывод, что выборки соответствуют друг другу, так как их распределения похожи.

1.3. Проведите линейную регрессию используя LinearRegression. Получите коэффициенты регрессии и объясните полученные результаты.

Для проведения линейной регрессии необходимо импортировать LinearRegression из sklearn.linear_model, создать экземпляр класса линейного регрессора и с помощью метода fit обучить модель на соответствующей выборке. После выполнения данных шагов можно извлечь коэффициенты линейной регрессии. Реализация вышеописанных шагов представлена в Листинге 1.3.1, результат выполнения данного кода – в Листинге 1.3.2.

Листинг 1.3.1 – Реализация обучения линейной регрессии и вывод её коэффициентов

```
from sklearn.linear_model import LinearRegression


lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
print(f"coeffs = {lin_reg.coef_}")
print(f"intercept = {lin_reg.intercept_}")
```

Листинг 1.3.2 – Коэффициенты линейной регрессии

```
coeffs = [0.81366501 0.56053529]
intercept = 0.0024023154635082443
```

Исходя из полученных коэффициентов, можно сделать следующие выводы:

• с ростом значения признака «x1» растёт значение зависимой переменной «y»;

• с ростом значения признака «x2» растёт значение зависимой переменной «y»;

• так как данные нормированы относительно стандартного распределения, то полученные коэффициенты линейной регрессии можно сравнивать между собой, значит, признак «x1» сильнее влияет на зависимую переменную «y».

1.4. Для обучающей и тестовой выборки рассчитайте коэффициент детерминации, MAPE, MAE. Объясните полученные значений метрик. Сравните метрики для обучающей и тестовой выборки, сделайте выводы о качестве обобщения полученной модели.

Для расчёта метрик, определённых в условии, реализована функция compute_regression_metrics. Также данная функция рассчитывает значения, на основе которых можно сделать вывод о зависимости остатков друг от друга (тест Дарбина-Уотсона), о гомоскедастичности (тест Уайта) и о распределении остатков (тест Шапиро-Уилка для небольших выборок и тест Жарко-Бера для объёмных). Реализация данной функции и её вызов представлены в Листинге 1.4.1, результат вызова представлен в Листинге 1.4.2.

Листинг 1.4.1 – Реализация функции расчёта метрик регрессии и её вызов для линейной регрессии

```
from sklearn.metrics import r2_score, mean_absolute_error, \
    mean_absolute_percentage_error
from statsmodels.stats.stattools import durbin_watson
from statsmodels. stats.diagnostic import het_white
import numpy as np
from scipy.stats import jarque_bera


def compute_regression_metrics(y_pred_train, y_pred_test, model):
    dw_train = durbin_watson(y_pred_train - y_train)
    dw_test = durbin_watson(y_pred_test - y_test)
    X_train_tmp = np.insert(X_train, 0, np.ones(len(X_train)), axis=1)
    X_test_tmp = np.insert(X_test, 0, np.ones(len(X_test)), axis=1)
    _, _, _, white_train = het_white(y_pred_train - y_train, X_train_tmp)
    _, _, _, white_test = het_white(y_pred_test - y_test, X_test_tmp)
    if len(y_pred_train) <= 5000:
        _, norm_distr_train = shapiro(y_pred_train - y_train)
        _, norm_distr_test = shapiro(y_pred_test - y_test)
    else:
        _, norm_distr_train = jarque_bera(y_pred_train - y_train)
        _, norm_distr_test = jarque_bera(y_pred_test - y_test)
    r2_train = r2_score(y_train, y_pred_train)
    r2_test = r2_score(y_test, y_pred_test)
    print(f"r2_train = {r2_train}; r2_test = {r2_test}")
    mae_train = mean_absolute_error(y_train, y_pred_train)
    mae_test = mean_absolute_error(y_test, y_pred_test)
    print(f"mae_train = {mae_train}; mae_test = {mae_test}")
    mape_train = mean_absolute_percentage_error(y_train, y_pred_train) * 100
    mape_test = mean_absolute_percentage_error(y_test, y_pred_test) * 100
    print(f"mape_train = {mape_train}; mape_test = {mape_test}")
    models_info[model] += [r2_train, r2_test, mae_train, mae_test,
                           mape_train, mape_test, dw_train, dw_test,
                           white_train, white_test, norm_distr_train, norm_distr_test]

y_pred_train = lin_reg.predict(X_train)
y_pred_test = lin_reg.predict(X_test)
models_info = {'LinearRegression': [lin_reg.coef_, lin_reg.intercept_]}
compute_regression_metrics(y_pred_train, y_pred_test, 'LinearRegression')
```

Листинг 1.4.2 – Полученные метрики для линейной регрессии

```
r2_train = 0.9957121249224917; r2_test = 0.9967883257132645
mae_train = 0.05102671267559768; mae_test = 0.04876932790875637
mape_train = 24.717129772737678; mape_test = 50.25875365386161
```

Исходя из полученных результатов, можно сделать вывод, что данная модель обладает хорошей обобщающей способностью ввиду довольно неплохих значений метрик, однако может быть подвержена переобучению ввиду того, что обучающая и тестовая выборка довольно сильно похожи друг на друга по качественному составу.

1.5. Повторите пункты 1.3 и 1.4 для модификаций линейной регрессии:

1.5.1. Лассо регрессия. Самостоятельно подберите гиперпараметры.

Для подбора параметров Лассо регресси, Гребневой и ElasticNet будет использована кросс-валидация, представленная в виде GridSearchCV из sklearn.model_selection. Реализация подбора наилучших параметров моделей представлена функцией print_best_estimator, которая принимает на вход факторы модели, зависимую переменную, экземпляр регрессора, а также параметры модели, среди которых нужно выбрать наилучший. Модели будет оцениваться по таким параметрам, как R^2, $NMAE$ и $NMAPE$. Наилучшая модель будет оценена по коэффициенту детерминации. Реализация данной функции и её вызов для Lasso регрессии представлены в Листинге 1.5.1.1, результат выполнения представлен в Листинге 1.5.1.2.

Листинг 1.5.1.1 – Реализация функции для подбора наилучших параметров регрессии и её вызов для Lasso регрессии

```
from sklearn.linear_model import Lasso, Ridge, ElasticNet
    from sklearn.model_selection import GridSearchCV


    def print_best_estimator(X, y, estimator, params):
        scoring = ['r2', 'neg_mean_absolute_error',
                   'neg_mean_absolute_percentage_error']
        gscv = GridSearchCV(estimator=estimator, param_grid=params, scoring=scoring,
                            refit='r2')
        gscv.fit(X, y)
        print(gscv.best_estimator_)
        print(f"R2 = {gscv.best_score_}")

print_best_estimator(lin4.iloc[:, :-1], lin4['y'],
        Lasso(), {'alpha': np.arange(0.001, 0.3001, 0.001)})
```

Листинг 1.5.1.2 – Наилучшие параметры Lasso регрессии

```
Lasso(alpha=0.001)
R2 = 0.9959107356595659
```

Наилучшие метрики достигаются при alpha равном 0.001. Если запустить данную модель при params от 0.0001 до 0.0011 с шагом 0.0001, то наилучшие метрики будут достигнуты при alpha равном 0.0001 (см. Листинге 1.5.1.3).

Листинг 1.5.1.3 – Наилучшие параметры Lasso регрессии

```
Lasso(alpha=0.0001)
R2 = 0.9959135656908537
```

Были проведены тесты и с меньшими начальными значениями alpha, в которых результат был аналогичен: наилучшие метрики достигаются при минимально возможно заданном значении alpha. Исходя из полученных данных, можно сделать вывод, что чем меньше alpha, тем лучше метрики. По итогам было выбрано alpha равное 1e-15 (см. метрики кросс-валидации в Листинге 1.5.1.4), так как это минимальное подобранное вручную значение, при котором не появлялось никакого предупреждения от интерпретатора, и при данном значении была обучена и протестирована модель. Реализация представлена в Листинге 1.5.1.5), метрики – в Листинге 1.5.1.6).

Листинг 1.5.1.4 – Наилучшие параметры Lasso регрессии

```
Lasso(alpha=1e-15)
R2 = 0.9959136752121207
```

Листинг 1.5.1.5 – Обучение и тестирование Lasso регрессии

```
lasso = Lasso(1e-15)
lasso.fit(X_train, y_train)
models_info['Lasso'] = [lasso.coef_, lasso.intercept_ ]
y_pred_train = lasso.predict(X_train)
y_pred_test = lasso.predict(X_test)
compute_regression_metrics(y_pred_train, y_pred_test, 'Lasso')
```

Листинг 1.5.1.6 – Метрики Lasso регрессии при использовании наилучших гиперпараметров

```
r2_train = 0.9957121249224917; r2_test = 0.9967883257132645
mae_train = 0.051026712675597684; mae_test = 0.048769327908756196
mape_train = 24.717129772737643; mape_test = 50.25875365386152
```

Исходя из полученных результатов, можно сделать вывод, что Lasso регрессия для данного датасета бесполезна, так как обучающие и тестовые выборки во время работы кросс-валидации получаются слишком похожими друг на друга, из-за чего нет необходимости в урезании коэффициентов регрессии и их подгонки одновременно и под обучающую выборку, и тестовую. Так как значение alpha получилось очень малым, можно сделать вывод, что Lasso регрессия по своей сути превратилась в линейную регрессию без каких либо модификаций.

1.5.2. Гребневая регрессия. Самостоятельно подберите гиперпараметры.

Для Гребневой регрессии были получены аналогичные выводы: чем меньше штраф, тем лучше метрики (см. Листинге 1.5.2.1, Листинге 1.5.2.2).

Листинг 1.5.2.1 – Подбор наилучших гиперпараметров Ridge регрессии

```
print_best_estimator(lin4.iloc[:, :-1], lin4['y'],
    Ridge(), {'alpha': np.arange(0.0001, 0.3001, 0.0001)})
```

Листинг 1.5.2.2 – Наилучшие гиперпараметры Ridge регрессии

```
Ridge(alpha=0.0001)
R2 = 0.9959136750313483
```

По причине, описанной в предыдущем пункте, было выбрано alpha равное 1e-15 (см. метрики кросс-валидации в Листинге 1.5.2.3) и при данном гиперпараметре обучена и протестирована модель. Реализация представлена в Листинге 1.5.2.4, метрики – в Листинге 1.5.2.5.

Листинг 1.5.2.3 – Наилучшие гиперпараметры Ridge регрессии

```
Ridge(alpha=1e-15)
R2 = 0.9959136752121207
```

Листинг 1.5.2.4 – Обучение и тестирование модели Ridge регрессии

```
ridge = Ridge(1e-15)
ridge.fit(X_train, y_train)
models_info['Ridge'] = [ridge.coef_, ridge.intercept_]
y_pred_train = ridge.predict(X_train)
y_pred_test = ridge.predict(X_test)
compute_regression_metrics(y_pred_train, y_pred_test, 'Ridge')
```

Листинг 1.5.2.5 – Метрики Ridge регрессии при использовании наилучших гиперпараметров

```
r2_train = 0.9957121249224917; r2_test = 0.9967883257132645
mae_train = 0.05102671267559767; mae_test = 0.04876932790875634
mape_train = 24.717129772737824; mape_test = 50.25875365386133
```

Исходя из полученных результатов, можно сделать аналогичный вывод: Ridge регрессия бесполезна для данного датасета, как и Lasso по той же причине.

1.5.3. ElasticNet. Самостоятельно подберите гиперпараметры.

Для ElasticNet были получены аналогичные выводы: чем меньше штраф, тем лучше метрики (см. Листинге 1.5.3.1, Листинге 1.5.3.2).

Листинг 1.5.3.1 – Подбор наилучших гиперпараметров ElasticNet регрессии

```
print_best_estimator(lin4.iloc[:, :-1], lin4['y'],
    ElasticNet(), {'alpha': np.arange(0.01, 1.01, 0.01), \
                   'l1_ratio': np.arange(0.01, 1.01, 0.01)})
```

Листинг 1.5.3.2 – Наилучшие гиперпараметры ElasticNet регрессии

```
ElasticNet(alpha=0.01, l1_ratio=0.01)
R2 = 0.995807647137067
```

Аналогично причинам, описанным в Ridge и Lasso регрессиях, были вручную подобраны значения alpha и l1_ratio (см. метрики кросс-валидации в Листинге 1.5.3.3) и при данных гиперпараметрах обучена и протестирована модель. Реализация представлена в Листинге 1.5.3.4, метрики – в Листинге 1.5.3.5.

Листинг 1.5.3.3 – Наилучшие гиперпараметры ElasticNet

```
ElasticNet(alpha=1e-08, l1_ratio=1e-08)
R2 = 0.9959136752048924
```

Листинг 1.5.3.4 – Обучение и тестирование модели ElasticNet регрессии

```
en = ElasticNet(alpha=1e-8, l1_ratio=1e-8)
en.fit(X_train, y_train)
models_info['ElasticNet'] = [en.coef_, en.intercept_]
y_pred_train = en.predict(X_train)
y_pred_test = en.predict(X_test)
compute_regression_metrics(y_pred_train, y_pred_test, 'ElasticNet')
```

Листинг 1.5.3.5 – Метрики ElasticNet регрессии при использовании наилучших гиперпараметров

```
r2_train = 0.9957121249224916; r2_test = 0.9967883257812304
mae_train = 0.05102671269855512; mae_test = 0.04876932680638668
mape_train = 24.71712968081688; mape_test = 50.258752786321125
```

Исходя из полученных результатов, можно сделать аналогичный вывод: ElasticNet регрессия бесполезна для данного датасета, как Lasso и Ridge по той же причине.

1.5.4. Регрессия оптимизируемая градиентным спуском.

Для выполнения данного пункта реализована функция gradient_descent, которая принимает на вход факторы и независимую переменную, а возвращает вектор коэффициентов линейной регрессии. Подбор коэффициентов осуществляется с помощью модифицированной версии градиентоного спуска, которая не требует задания скорости обучения, так как автоматически рассчитывается на каждой итерации алгоритма решением следующей задачи оптимизации:

$$Q(\mathbf{W_{k-1}} - \alpha \nabla Q(\mathbf{W_{k-1}})) \to min \tag{1}$$

при том, что:

$$Q(\mathbf{W_k}) = ||\mathbf{X}\mathbf{W_k} - \mathbf{Y}||^2 \tag{2}$$

$$\mathbf{W_k} = \mathbf{W_{k-1}} - \alpha \nabla Q(\mathbf{W_{k-1}}) \tag{3}$$

Для решения первого уравнения достаточно найти дифференциал функции потерь относительно скорости обучения и разделить на приращение, что приведёт к получению градиента, который следует приравнять к нулю. После чего необхо-

димо из полученного уравнения выразить α, что выглядит следующим образом:

$$\alpha = \frac{[\nabla Q(\mathbf{W_{k-1}})]^{\mathbf{T}}\mathbf{X}^{\mathbf{T}}(\mathbf{X}\mathbf{W_{k-1}} - \mathbf{Y})}{[\nabla Q(\mathbf{W_{k-1}})]^{\mathbf{T}}\mathbf{X}^{\mathbf{T}}\mathbf{X}\nabla Q(\mathbf{W_{k-1}})} \tag{1}$$

Перед запуском алгоритма случайным образом генерируются коэффициенты регрессии и итеративно данные коэффициенты уточняются. Однако стоит отметить, что из-за несовершенства вычислений с плавающей точкой в Python, в какой-то момент значение α могло оказаться в окрестности нуля слева, что могло привести к непредсказуемым значениям. Поэтому, если на какой-то итерации $\alpha < 0$, то текущая итерация сбрасывается и генерируются новые начальные коэффициенты регрессии. Алгоритм завершает работу, если изменение коэффициентов не преодолело некоторый порог ϵ, либо число итераций достигло 1000. Реализация функции gradient_descent, её запуск, обучение и тестирование представлены в Листинге 1.5.4.1, метрики - в Листинге 1.5.4.2.

Листинг 1.5.4.1 – Реализация функции градиентного спуска, её запуск, обучение и тестирование модели

```
import tensorflow as tf


def gradient_descent(X, y):
    X_tf = tf.constant(X, dtype=tf.float64)
    X_tf_T = tf.transpose(X_tf)
    y_tf = tf.constant(y, dtype=tf.float64)
    w = tf.Variable([np.random.randn() for _ in range(len(X[0]))],
                    dtype=tf.float64)
    epoch = 0
    while epoch < 1000:
        with tf.GradientTape() as tape:
            y_pred = tf.linalg.matvec(X_tf, w)
            loss = tf.square(tf.norm(y_pred - y_tf, ord='euclidean'))
        grad_true = tape.gradient(loss, w)
        grad = tf.reshape(grad_true, shape=[len(X[0]), 1])
        grad_T = tf.transpose(grad)
        numerator = tf.linalg.matvec(tf.matmul(grad_T, X_tf_T),
                                     tf.linalg.matvec(X_tf, w) - y_tf)
        denominator = tf.matmul(tf.matmul(grad_T, X_tf_T), X_tf) @ grad
        alpha = tf.squeeze(numerator / denominator)
        if alpha < 0:
            w = tf.Variable([np.random.randn() for _ in range(len(X[0]))],
                            dtype=tf.float64)
            epoch = 0
            continue
        if tf.reduce_all(tf.math.less(tf.abs(alpha*grad_true),
                                      tf.constant(1e-4, dtype=tf.float64))):
            break
        w.assign_add(-(alpha*grad_true))
        epoch += 1
    return w

X_train_tmp = np.insert(X_train, 0, np.ones(len(X_train)), axis=1)
X_test_tmp = np.insert(X_test, 0, np.ones(len(X_test)), axis=1)
w = gradient_descent(X_train_tmp, y_train)
y_pred_train = tf.linalg.matvec(X_train_tmp, w)
y_pred_test = tf.linalg.matvec(X_test_tmp, w)
models_info['GradientDescent'] = [w[1:].numpy(), w[0].numpy()]
compute_regression_metrics(y_pred_train, y_pred_test, 'GradientDescent')
```

Листинг 1.5.4.2 – Метрики линейной регрессии при подборе коэффициентов с помощью градиентного спуска

```
r2_train = 0.9957121243012466; r2_test = 0.9967883637292002
mae_train = 0.051026959163137514; mae_test = 0.04876791448947375
mape_train = 24.712167314185756; mape_test = 50.26145444267187
```

1.6. Постройте сводную таблицу для рассчитываемых метрик, и используемых методов (с разделением на обучающую и тестовую выборку). По таблице сделайте выводы о том, какой вид регрессии дал лучшую модель. Опишите какие проблемы могут возникнуть при применении каждой модели. Для объяснения результатов можно построить “карту высот (heat map)” с отображением корреляции признаков.

На предыдущих шагах можно было заметить некоторую переменную model_info. Данная переменная предназначена для записи таких данных, как коэффициенты регрессии, R^2, MAE, $MAPE$, результат теста Дарбина-Уотсона, Уайта и Шапиро-Уилка для остатков обучающей и тестовой выборок. Данная переменная необходима для формирования таблицы в виде pandas DataFrame, что реализовано в Листинге 1.6.1.

Листинг 1.6.1 – Реализация формирования сводной таблицы

```
models_info_pd = pd.DataFrame(models_info, index=['coeffs', 'intercept',
                                'r2_train', 'r2_test',
                                'mae_train', 'mae_test',
                                'mape_train', 'mape_test',
                                'dw_train', 'dw_test',
                                'white_train', 'white_test',
                                'norm_distr_train',
                                'norm_distr_test']).T
models_info_pd
```

Таблица представлена в Листинге 1.6.2.

Листинг 1.6.2 – Сводная таблица моделей регрессии

	coeffs	intercept	r2_train	r2_test
LinearRegression	[0.8136650070277945, 0.560535291286712]	0.002402	0.995712	0.996788
Lasso	[0.8136650070277932, 0.5605352912867111]	0.002402	0.995712	0.996788
Ridge	[0.8136650070277943, 0.5605352912867123]	0.002402	0.995712	0.996788
ElasticNet	[0.813664998492682, 0.5605352857957059]	0.002402	0.995712	0.996788
GradientDescent	[0.8136618262588496, 0.5605268595123033]	0.002402	0.995712	0.996788

	mae_train	mae_test	mape_train	mape_test	dw_train	dw_test
LinearRegression	0.051027	0.048769	24.71713	50.258754	2.189571	2.234512
Lasso	0.051027	0.048769	24.71713	50.258754	2.189571	2.234512
Ridge	0.051027	0.048769	24.71713	50.258754	2.189571	2.234512
ElasticNet	0.051027	0.048769	24.71713	50.258753	2.189571	2.234512
GradientDescent	0.051027	0.048769	24.714436	50.262251	2.189613	2.234444

	white_train	white_test	norm_distr_train	norm_distr_test
LinearRegression	0.539287	0.450069	0.765397	0.227989
Lasso	0.539287	0.450069	0.765397	0.227989
Ridge	0.539287	0.450069	0.765397	0.227989
ElasticNet	0.539287	0.450069	0.765467	0.227979
GradientDescent	0.539277	0.449724	0.765555	0.227922

Также построена карта высот (см. Листинге 1.6.3, Рисунок 1.6.1), исходя из которой, можно сделать вывод, что отсутствует проблема мультиколлинеарности, а также было подтверждено равенство, что коэффициент линейной регрессии при каждом факторе равен коэффициенту корреляции между данным фактором и независимой переменной «y» при условии, что стандартные отклонения факторов и целевого признака равны 1.

Листинг 1.6.3 – Реализация отрисовки корреляционной матрицы

```
sns.heatmap(lin4.corr(), cmap='coolwarm', annot=True).set_title('Карта высот')
```

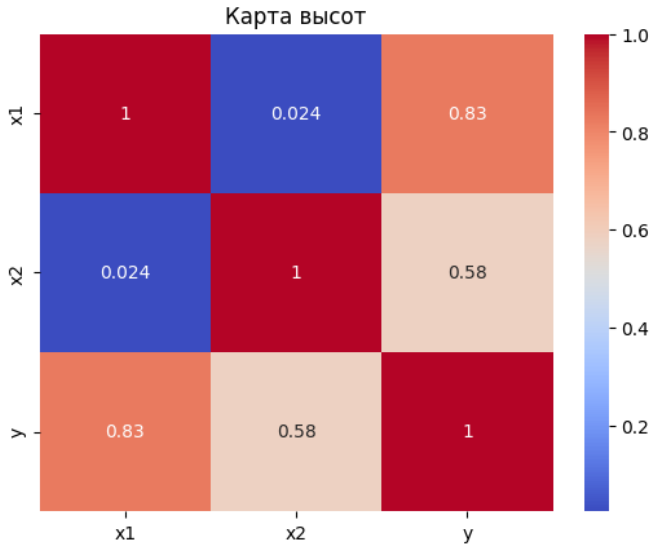

Рисунок 1.6.1 – Корреляционная матрица

Исходя из полученных результатов, можно сделать следующие выводы:

• Lasso, Ridge и ElasticNet оказались бесполезными для данного датасета, так как они стремились минимизировать свои штрафы, приближаясь к линейной регрессии без каких-либо модификаций, что связано со схожестью выборок и, как следствие, отсутствием необходимости в введении штрафов.

• Для каждой модели выполнены такие условия, как отсутствие зависимости остатков (значения dw_train и dw_test около 2), гомоскедастичность (значения white_train и white_test больше 0.05), нормальное распределение остатков (значения norm_distr_train и norm_distr_test больше 0.05).

• Так как Lasso, Ridge и ElasticNet при своих наилучших параметрах оказались линейной регрессией без модификаций, то лучшей моделью может быть либо LinearRegression, либо GradientDescent. Значения параметров R^2 и MAE одинаковые для данных двух регрессоров, однако значение $MAPE$ на тестовой выборке GradientDescent больше. Так как датасеты небольшие, более предпочтительным вариантом является использование аналитического решения. К тому же, значение $MAPE$ на тестовой выборке LinearRegression лучше. Следовательно, LinearRegresson является наилучшим регрессором для данного датасета.

• Все рассмотренные модели подвержены переобучению, так обучающие и

тестовые выборки были очень похожи друг на друга, и, как следствие, были полученые очень хорошие метрики.

1.7. Для модели, которая дала лучшие результаты, постройте диаграмму рассеяния между предикторами и откликом. На диаграмме изобразите какое значение должно быть, и какое предсказывается. Визуально оцените качестве построенного регрессора.

Так как LinearRegression уже обучена, достаточно вызвать метод predict для получения предсказанных значений и отобразить их на каждой диаграмме рассеяния вместе с истинными значениями. Реализация отрисовки представлена в Листинге 1.7.1, диаграммы рассеяния – на Рисунке 1.7.1.

Листинг 1.7.1 – Реализация отрисовки диаграмм рассеяния

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))
sns.scatterplot(x=lin4['x1'], y=lin4['y'], ax=axes[0],
                label='Исходные значения', s=20)
sns.scatterplot(x=lin4['x1'], y=lin_reg.predict(lin4.iloc[:, :-1]), ax=axes[0], c='orange',
                label='Предсказанные значения', s=20)
sns.scatterplot(x=lin4['x2'], y=lin4['y'], ax=axes[1],
                label='Исходные значения', s=20)
sns.scatterplot(x=lin4['x2'], y=lin_reg.predict(lin4.iloc[:, :-1]), ax=axes[1], c='orange',
                label='Предсказанные значения', s=20)
fig.suptitle('Диаграммы рассеяния с исходными значениями и предсказанными')
plt.show()
```

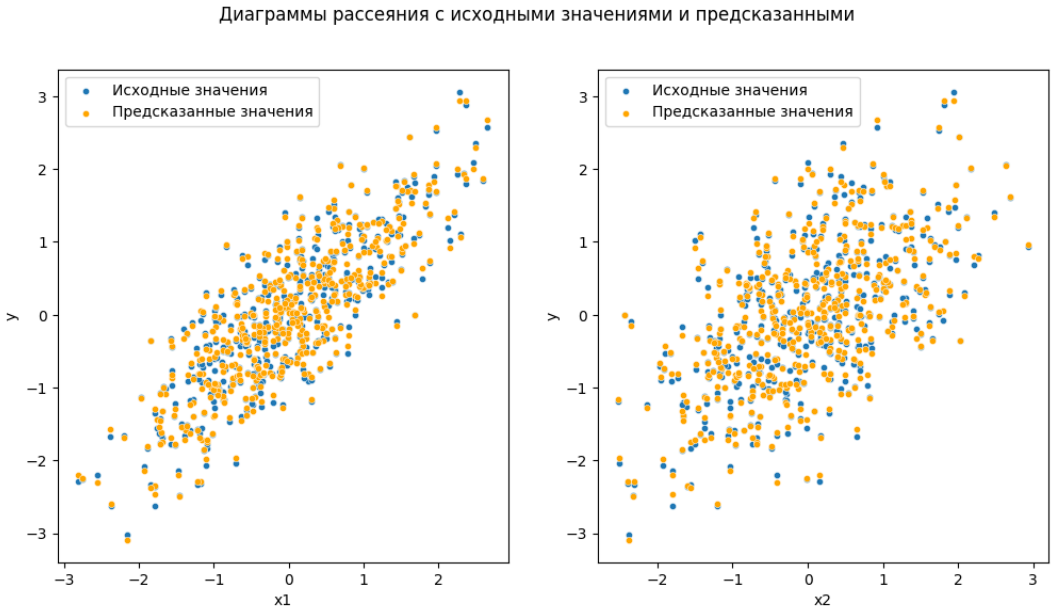

Рисунок 1.7.1 – Диаграммы рассеяния с исходными значениями и предсказанными

Исходя из данных на диаграммах, можно сделать вывод о хорошем качестве регрессии, так как предсказанные значения близки к истинным.

2. Нелинейная регрессия.

2.1. Загрузите набор данных соответствующей букве вашего варианта. Убедитесь, что загрузка прошла корректно.

Для загрузки данных необходимо воспользоваться методом read_csv, для проверки загруженных данных – методом head. Результат проверки загруженных данных представлен в Листинге 2.1.1.

Листинг 2.1.1 – Результат вывода первых пяти записей датафрейма

	x	y
0	1.2429	0.2452
1	-0.6314	-1.0334
2	0.9256	1.9695
3	0.6894	0.3605
4	-0.1864	-0.1788

Так как регрессия чувствительна к масштабу данных, следует провести стан-

дартизацию или нормализацию. Так как данные не подчиняются нормальному распределению (см. Листинге 2.1.2), следует воспользоваться нормализацией, но зачастую в задачах регрессии рекомендуется проделать стандартизацию, что и будет сделано (см. Листинг 2.1.3). В случае плохих метрик будет проделана нормализация.

Листинг 2.1.2 – Результат теста Шапиро-Уилка

```
column: x       pvalue: 4.5834321581172954e-12
column: y       pvalue: 2.640414301861415e-14
```

Листинг 2.1.3 – Реализация стандартизация данных

```
poly2_np = StandardScaler().fit_transform(poly2)
poly2 = pd.DataFrame(data=poly2_np.T, index=['x', 'y']).T
```

2.2. Используя train_test_split разбейте выборку на обучающую и тестовую. Проверьте, что тестовая выборка соответствует обучающей.

Для разбиения выборки на обучающую и тестовую необходимо воспользоваться функцией train_test _split, а для проверки соответствия выборок была выбрана диаграмма рассеяния. Реализация разбиения датасета и отрисовки диаграмм рассеяния представлена в Листинге 2.2.1, диаграмма рассеяния – на Рисунке 2.2.1.

Листинг 2.2.1 – Реализация разбиения датасета и отрисовки диаграмм рассеяния

```
X_train, X_test, y_train, y_test = train_test_split(poly2.iloc[:, :-1],
                                    poly2['y'], test_size=0.3,
                                    random_state=RS)
sns.scatterplot(x=X_train['x'], y=y_train, label='Обучающая выборка')
sns.scatterplot(x=X_test['x'], y=y_test, c='orange',
                label='Тестовая выборка').set_title(
                    'Диаграммы рассеяния обучающей \nи тестовой выборок по признакам')
plt.show()
```

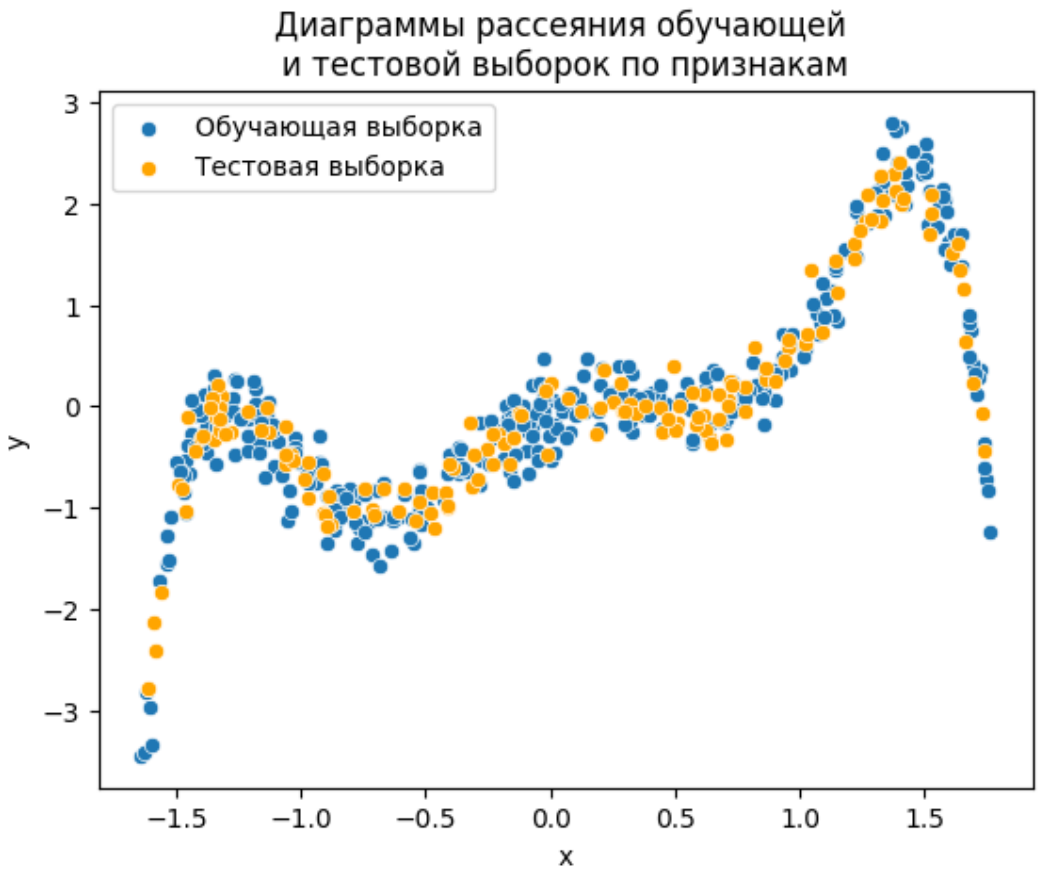

Рисунок 2.2.1 – Диаграмма рассеяния обучающей и тестовой выборок

Исходя из диаграммы рассеяния, можно сделать вывод, что обучающая и тестовая выборки соответствуют друг другу, так как имеют похожие распределения.

2.3. Проверьте работу стандартной линейной регрессии на загруженных данных. Постройте диаграмму рассеяния данных с выделенной полученной линией регрессии. Объясните полученный результат.

Для отрисовки диаграммы рассеяния и линии регрессии была реализована функция draw_regression_line, которая принимает на вход предсказанные значения целевого признака и отрисовывает диаграмму рассеяния для обучающей и тестовой выборок, а также линию регрессии. Реализация данной функции, обучение и тестирование линейной регрессии, а также вызов функции отрисовки диаграммы рассеяния и линии регрессии представлены в Листинге 2.3.1, результат отрисовки – на Рисунке 2.3.1.

Листинг 2.3.1 – Реализация разбиения датасета и отрисовки диаграмм рассеяния

```
def draw_regression_line(y_pred):
    sns.scatterplot(x=X_train['x'], y=y_train, label='Обучающая выборка')
    sns.scatterplot(x=X_test['x'], y=y_test, c='orange',
                    label='Тестовая выборка').set_title(
                        'Диаграмма рассеяния обучающей \nи тестовой выборок\n и линия
                        ↪  регрессии')
    sns.lineplot(x=np.arange(-1.7, 1.8, 0.01), y=y_pred, c='red',
                 label='Линия регрессии')
lin_reg.fit(X_train, y_train)
y_pred_train = lin_reg.predict(X_train)
y_pred_test = lin_reg.predict(X_test)
models_info = {'LinearRegression': [lin_reg.coef_, lin_reg.intercept_]}
compute_regression_metrics(y_pred_train, y_pred_test, 'LinearRegression')
draw_regression_line(lin_reg.predict(np.arange(-1.7, 1.8, 0.01).reshape(-1, 1)))
```

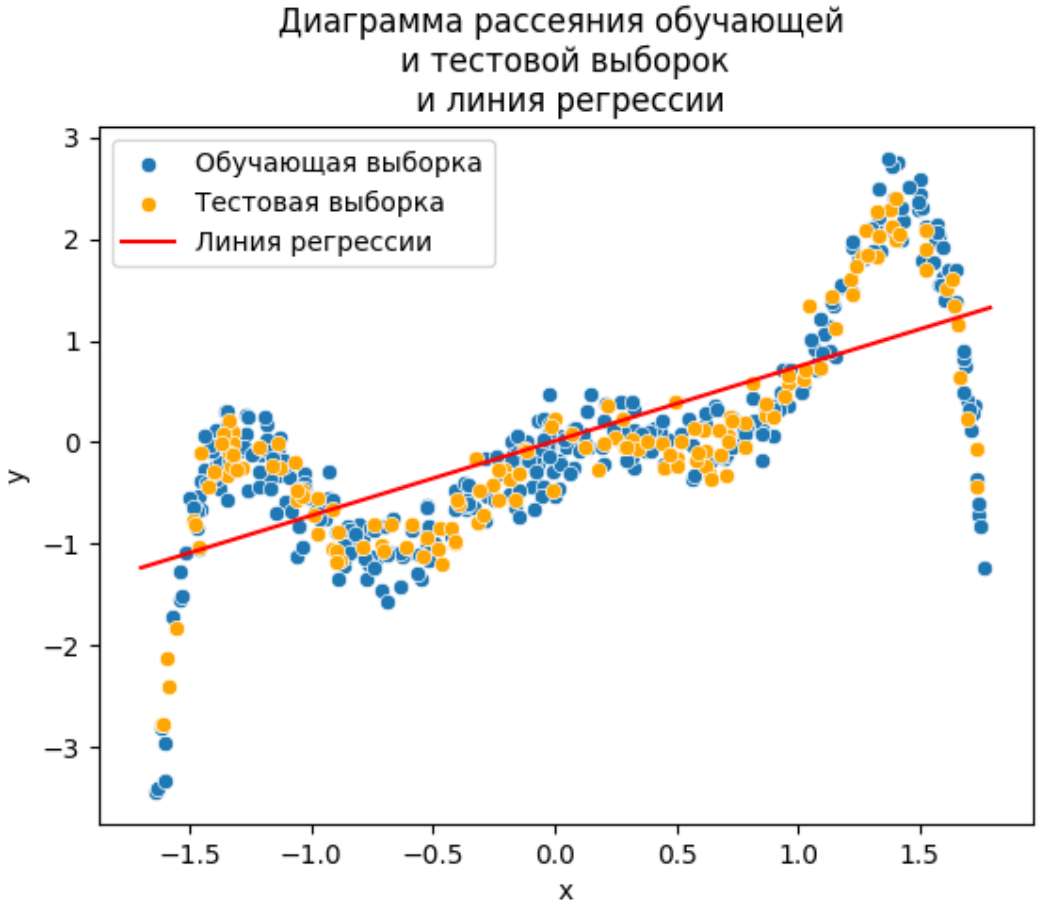

Рисунок 2.3.1 – Диаграмма рассеяния обучающей и тестовой выборок и линия регрессии

Исходя из данных на Рисунке 2.3.1, можно сделать вывод, что линейная

регрессия не подходит для данного датасета, так как для остатков не выполняется условие гомоскедастичности и они не подчиняются нормальному распределению.

2.4. Конструируя полиномиальный признаки для разных степеней полинома найдите степень полинома наилучшим образом аппроксимирующая данные. Постройте график зависимости коэффициента детерминации от степени полинома(на одном графике изобразите линии для обучающей и тестовой выборки отдельно). Сделайте вывод о том, при какой степени полинома модель начинает переобучаться.

Реализация отрисовки коэффициента детерминации от степени полинома для обучающей и тестовой выборок представлена в Листинге 2.4.1. Идея заключается в последовательном переборе степени полинома, обучении на данной модели, её тестировании и расчёта коэффициентов детерминации. График зависимости представлен на Рисунке 2.4.1.

Листинг 2.4.1 – Реализация перебора степени полинома и расчёта коэффициента детерминации для обучающей и тестовой выборок

```
from sklearn.preprocessing import PolynomialFeatures


r2_train_list, r2_test_list = [], []
for i in range(2, 40):
    X_train_polynom = PolynomialFeatures(i, include_bias=False).fit_transform(X_train)
    X_test_polynom = PolynomialFeatures(i, include_bias=False).fit_transform(X_test)
    lin_reg.fit(X_train_polynom, y_train)
    y_pred_train = lin_reg.predict(X_train_polynom)
    y_pred_test = lin_reg.predict(X_test_polynom)
    r2_train_list.append(r2_score(y_train, y_pred_train))
    r2_test_list.append(r2_score(y_test, y_pred_test))
plt.plot(np.arange(2, 40), r2_train_list, c='blue', label='r2_train')
plt.plot(np.arange(2, 40), r2_test_list, c='orange', label='r2_test')
plt.title('Зависимость коэффициента детерминации\n от степени полинома')
plt.ylim((0, 1))
plt.legend()
plt.grid()
plt.xlabel('Степень полинома')
plt.ylabel('$R^2$')
plt.xticks(np.arange(0, 40, 5))
plt.show()
```

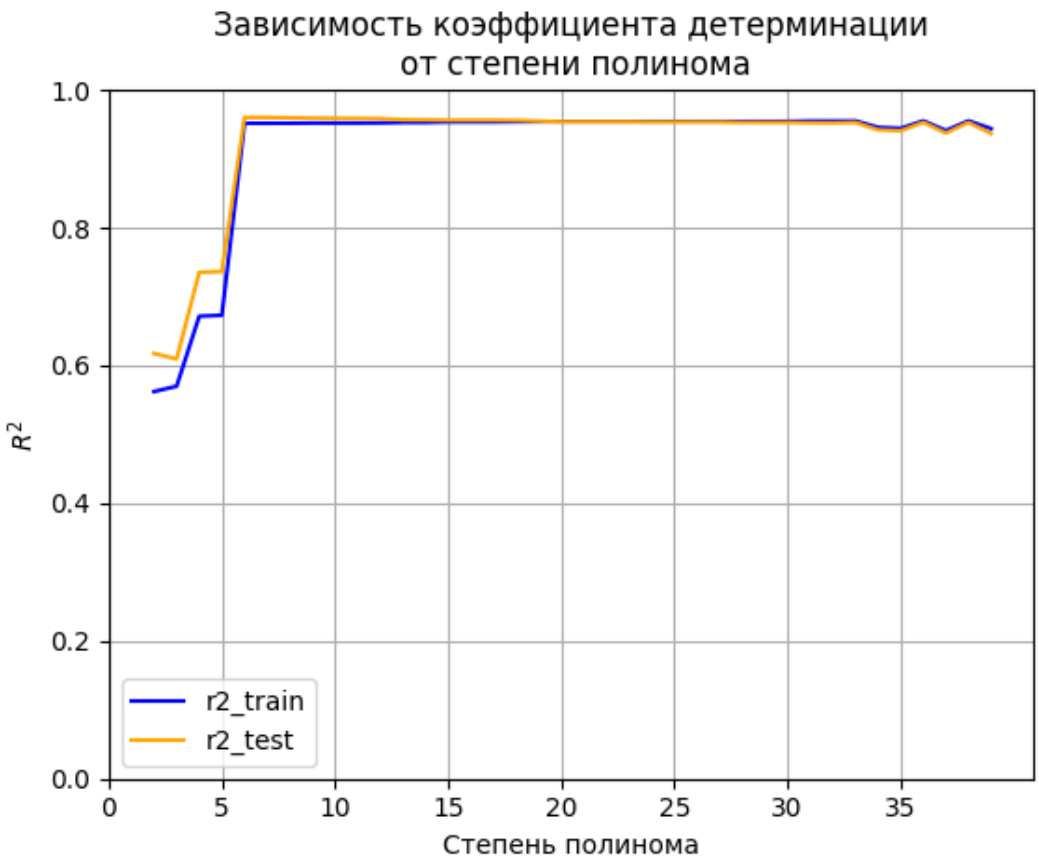

Рисунок 2.4.1 – Зависимость коэффициентов детерминации от степени полинома

Исходя из данных на Рисунке 2.4.1, можно сделать вывод, что наилучшая аппроксимация достигается при степени полинома, равной 6, так как при данной модели коэффициент детерминации для обучающей и тестовой выборок одинаково высок. С ростом степени, начиная с шестой, коэффициент детерминации тестовой выборки уменьшается, сближаясь с коэффициентом детерминации обучающей выборки, а затем начинаются некоторые колебания графика, что говорит о начале переобучения. Таким образом, переобучение начинается при степени полинома, равной 33.

2.5. Для выбранной степени полинома, рассчитайте и проанализируйте полученные коэффициенты. Рассчитай значение метрик коэффициент детерминации, MAPE, MAE.

Для реализации данного задания необходмо обучить и протестировать модель при степеним полинома, равной 6, и воспользоваться реализованной ранее

функцией compute_regression_metrics. Реализация вышеописнных шагов представлена в Листинге 2.5.1, метрики – в Листинге 2.5.2.

Листинг 2.5.1 – Реализация обучения, тестирования и расчёта метрик полиномиальной модели

```
X_train_polynom = PolynomialFeatures(6, include_bias=False).fit_transform(X_train)
X_test_polynom = PolynomialFeatures(6, include_bias=False).fit_transform(X_test)
lin_reg.fit(X_train_polynom, y_train)
print(f"coeffs = {lin_reg.coef_}")
print(f"intercept = {lin_reg.intercept_}")
y_pred_train = lin_reg.predict(X_train_polynom)
y_pred_test = lin_reg.predict(X_test_polynom)
models_info['PolynomalRegression'] = [lin_reg.coef_, lin_reg.intercept_]
compute_regression_metrics(y_pred_train, y_pred_test, 'PolynomalRegression')
```

Листинг 2.5.2 – Метрики полиномиальной модели при степени, равной 6

```
coeffs = [ 1.08563669 -2.63136731 -0.77934903  3.76719426  0.34714399 -1.09193164]
intercept = -0.037206129149112324
r2_train = 0.9516868902698403; r2_test = 0.9601095894162115
mae_train = 0.1756879093504469; mae_test = 0.1564958706255694
mape_train = 83.70000157075901; mape_test = 117.33836009813152
```

Исходя из полученных коэффициентов, можно сделать вывод, что наибольший вклад в предсказание вносят коэффициенты при чётвёртой и второй степенях.

2.6. Для выбранной степени полинома, постройте диаграмму рассеяния данных с линией соответствующей полученному полиному. Сделайте выводы о качестве аппроксимации.

Для выполнения данного задания необходимо воспользоваться ранее реализованной функцией draw_regression_line, передав ей соответствующий аргумент (см. Листинг 2.6.1).

Листинг 2.6.1 – Реализация вызова функции для отрисовки диаграммы рассеяния и линии регрессии

```
X = PolynomialFeatures(6, include_bias=False).fit_transform(
    np.arange(-1.7, 1.8, 0.01).reshape(-1, 1))
draw_regression_line(lin_reg.predict(X))
```

Диаграмма рассеяния и линия регрессии представлены на Рисунке 2.6.1.

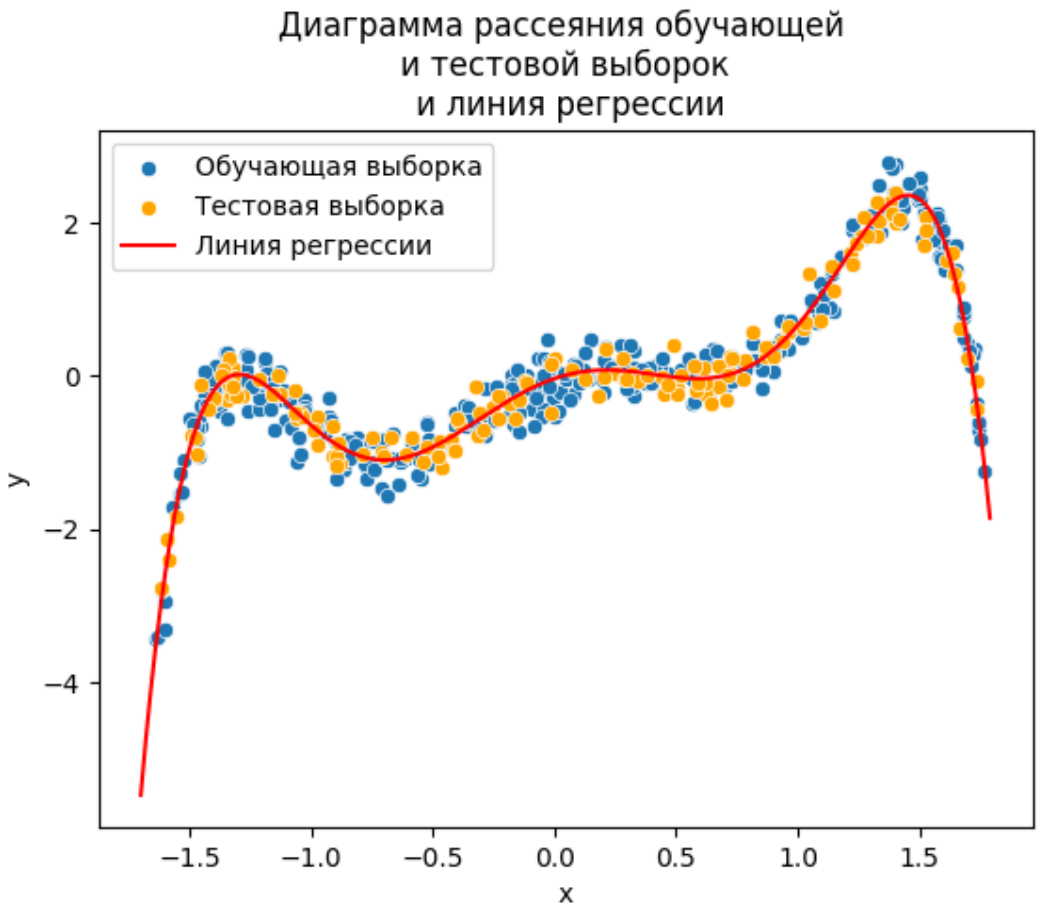

Рисунок 2.6.1 – Диаграмма рассеяния обучающей и тестовой выборок и линия регрессии

Исходя из диаграммы рассеяния и линии регрессии, можно сделать вывод, что аппроксимация хорошо описывает данные.

2.7. Для выбранной степени полинома, решите задачу нелинейной регрессии без конструирования полиномиальных признаков и используя библиотеку TensorFlow.

Для выполнения данного задания была использована ранее реализованная функция gradient_descent, которая запускается 10 раз и среди всех решений выбирается наилучшее. Наилучшее решение выбирается по значению коэффициента детерминации на тестовой выборке. Реализация представлена в Листинге 2.7.1. Также стоит отметить, что признаки не конструировались и на предыдущем шаге, так как в данном датасете всего один фактор, поэтому отличие данного пункта от предыдущего заключается в том, что в теперь коэффициенты подбираются с

помощью градиентного спуска, а для вычислений используется tensorflow.

Листинг 2.7.1 – Реализация подбора наилучших коэффициентов с помощью градиентного спуска

```
X_train_polynom = PolynomialFeatures(6).fit_transform(X_train)
X_test_polynom = PolynomialFeatures(6).fit_transform(X_test)
w = gradient_descent(X_train_polynom, y_train)
y_pred_test = tf.linalg.matvec(X_test_polynom, w)
y_pred_train = tf.linalg.matvec(X_train_polynom, w)
max_r2_score = r2_score(y_test, y_pred_test)
best_w = w
for i in range(10):
    w = gradient_descent(X_train_polynom, y_train)
    y_pred_train = tf.linalg.matvec(X_train_polynom, w)
    y_pred_test = tf.linalg.matvec(X_test_polynom, w)
    if r2_score(y_test, y_pred_test) > max_r2_score:
        max_r2_score = r2_score(y_test, y_pred_test)
        best_w = w
```

2.8. Рассчитай метрики коэффициент детерминации, MAPE, MAE, а также постройте диаграмму рассеяния для данных линией соответствующей полученному полиному, для модели полученной при помощи TensorFlow.

Для расчёта метрик использована ранее реализованная функция compute_regression_metrics (см. Листинг 2.8.1), полученные метрики представлены в Листинге 2.8.2.

Листинг 2.8.1 – Реализация получения предсказаний для обучающей и тестовой выборок, а также вызов функции для получения требуемых метрик

```
y_pred_train = tf.linalg.matvec(X_train_polynom, best_w)
y_pred_test = tf.linalg.matvec(X_test_polynom, best_w)
models_info['GradientDescent'] = [best_w[1:].numpy(), best_w[0].numpy()]
compute_regression_metrics(y_pred_train, y_pred_test, 'GradientDescent')
```

Листинг 2.8.2 – Метрики, полученные при использовании градиентного спуска

```
r2_train = 0.867648718485019; r2_test = 0.8866538280290676
mae_train = 0.29577301399196787; mae_test = 0.2753123052882973
mape_train = 125.21077029101431; mape_test = 318.13896245864356
```

Для отрисовки диаграммы рассеяния и линии регрессии достаточно вызвать ранее реализованную функцию draw_regression_line с метками для фактора в диа-

пазоне от -1.7 до 1.8 с шагом 0.01. Реализация формирования меток и вызов функции для отрисовки линии регресси представлена в Листинге 2.8.3, диаграмма рассеяния – на Рисунке 2.8.1.

Листинг 2.8.3 – Метрики, полученные при использовании градиентного спуска

```
X = PolynomialFeatures(6, include_bias=False).fit_transform(
    np.arange(-1.7, 1.8, 0.01).reshape(-1, 1))
y_pred = tf.linalg.matvec(X, best_w)
draw_regression_line(y_pred)
```

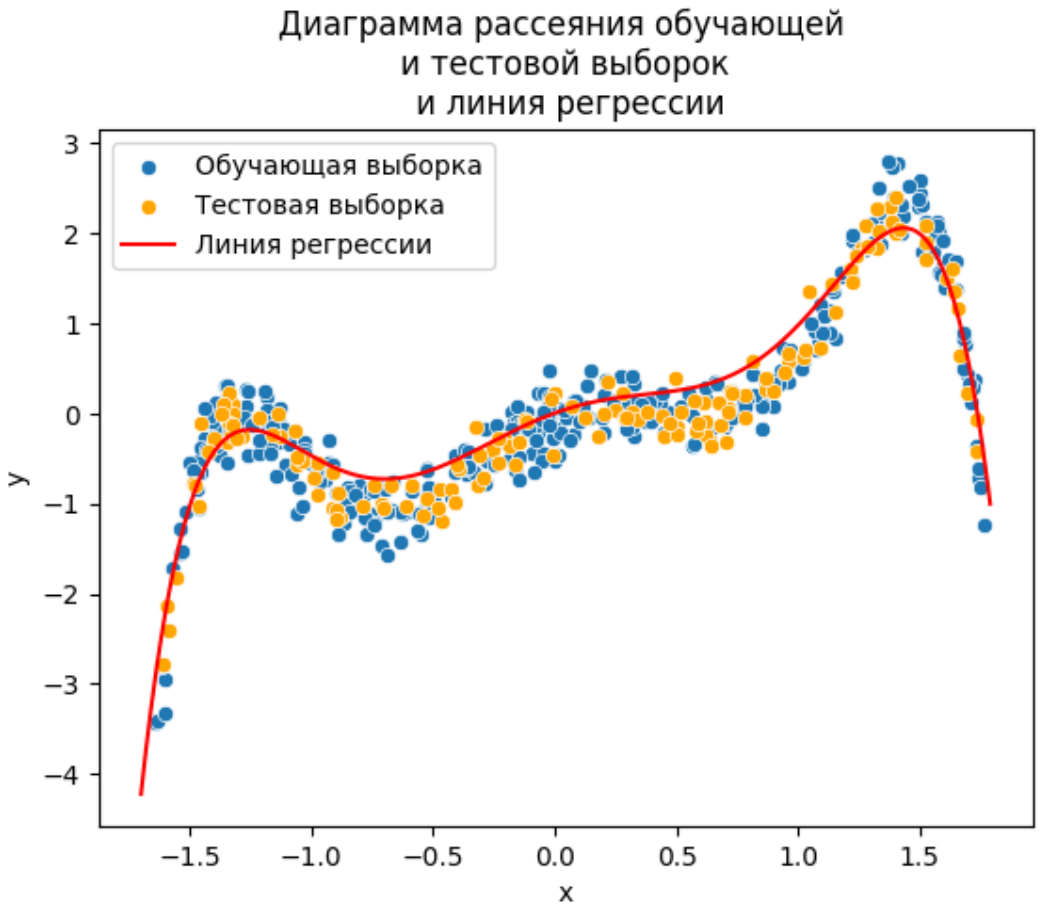

Рисунок 2.8.1 – Диаграмма рассеяния обучающей и тестовой выборок и линия регрессии

2.9. Сравните результаты полученные с/без конструирования полиномиальных признаков.

Так как в течение тестирования различных моделей регрессии все полученные данные сохранялись в словарь models_info, то для формирования таблицы

достаточно преобразовать словарь в pandas DataFrame, что представлено в Листинге 1.6.1, а таблица – в Листинге 2.9.1.

Листинг 2.9.1 – Сравнение различных моделей для нелинейной регрессии

	coeffs	intercept	r2_train	r2_test
LinearRegression	[0.7329049744328467]	0.011608	0.528582	0.568458
PolynomalRegression	[1.0856366873504253, ...	-0.037206	0.951687	0.96011
GradientDescent	[-1.0253746449037824, -0.04...	0.687263	0.867649	0.886654

	mae_train	mae_test	mape_train	mape_test	dw_train	dw_test
LinearRegression	0.538333	0.529896	220.253864	653.008023	1.864989	1.919171
PolynomalRegression	0.175688	0.156496	83.700002	117.33836	2.016992	2.024651
GradientDescent	0.295773	0.275312	125.21077	318.138962	1.75378	1.488391

	white_train	white_test	norm_distr_train	norm_distr_test
LinearRegression	0.0	0.0	0.0	0.00036
PolynomalRegression	0.126662	0.246912	0.132532	0.814414
GradientDescent	0.001134	0.089164	0.406108	0.05912

Исходя из данных в таблице выше, можно сделать следующие выводы:

• Модель LinearRegression не применима к нелинейной регрессии по следующим причинам: низкие метрики, отсутствие гомоскедастичности (white_train и white_test меньше 0.05), остатки не подчиняются нормальному распределению (norm_distr_train и norm_distr_test меньше 0.05).

• Нелинейная регрессия, оптимизированная градиентным спуском не применима для данного датасета, так как остатки могут зависеть от друга и для тренеровочной выборки не выполняется условие гомоскедастичности.

• Наиболее предпочтительным вариантом является PolynomialFeatures, так как данная модель показывает хорошие метрики, отсутствует автокорреляция, остатки гомоскедастичны и нормально распределены.

• Стоит отметить, что модель PolynomialFeatures для данного датасета подвержена переобучению, так как метрики одинаково хороши как для обучающей, так и для тестовой выборок, что связано со схожестью данных.

• Ввиду хороших метрик необходимость в замене стандартизации на нормализацию отсутствует.

3. Оценка модели регрессии.

3.1. Загрузите набор данных Student_Performance.csv. Данный набор данных содержит информацию о характеристиках студента, а также качестве его обучения.

Для загрузки набора данных необходимо воспользовться методом из pandas, а именно read_csv с передачей параметра пути датаесета. Для проверки загруженных данных применён метод head. Реализация выполнения вышеописанных шагов представлена в Листинге 3.1.1.

Листинг 3.1.1 – Результат вывода первых пяти записей датасета

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
0	7	99	Yes	9	1	91.0
1	4	82	No	4	2	65.0
2	8	51	Yes	7	2	45.0
3	5	52	Yes	5	2	36.0
4	7	75	No	8	5	66.0

3.2. Проведите предобработку набора данных - замена текстовых данных, удаление null значений, удаление дубликатов. Разделите на обучающую и тестовую выборку.

Для удаления записей с null признаками необходимо воспользоваться методом dropna из pandas с удалением непосредственно в самом датафрейме. Для удаления дубликатов следует воспользоваться методом drop_dublicates с параметром удаления в самом датафрейме. Значение 'Yes' было заменено на 1 и 'No' – на 0 непосредственно в самом датафрейме. Так как стандартизация хорошо себя показала на предыдущих заданиях, здесь она так же будет применена. В случае плохих показателей стандартизация будет заменена на нормализацию. Для разбиения выборки на обучающую и тестовую был использован метод train_test_split в соотношении 0.7:0.3 соответственно. Реализация вышеописанных шагов представлена в Листинге 3.2.1.

Листинг 3.2.1 – Предобработка данных и разбиение датасета на обучающую и тестовую выборки

```
students.dropna(inplace=True)
students.drop_duplicates(inplace=True)
students.replace(['Yes', 'No'], [1, 0], inplace=True)
students_np = StandardScaler().fit_transform(students)
students = pd.DataFrame(data=students_np.T, index=students.columns).T
X_train, X_test, y_train, y_test = train_test_split(students.iloc[:, :-1], students['Performance
↪  Index'], test_size=0.3, random_state=RS)
```

3.3. Постройте модель, которая будет предсказывать значение признака Performance Index на основе остальных признаков. Модель выберите самостоятельно.

Для начала следует проверить как работает LinearRegression на даннном датасете. Реализация обучения, тестирования и вывода метрик представлена в Листинге 3.3.1, метрики – в Листинге 3.3.2.

Листинг 3.3.1 – Реализация обучения, тестирования и вывода метрик LinearRegression

```
lin_reg.fit(X_train, y_train)
y_pred_train = lin_reg.predict(X_train)
y_pred_test = lin_reg.predict(X_test)
models_info = {'LinearRegression': [lin_reg.coef_, lin_reg.intercept_]}
compute_regression_metrics(y_pred_train, y_pred_test, 'LinearRegression')
```

Листинг 3.3.2 – Метрики LinearRegression

```
r2_train = 0.9888937265788157; r2_test = 0.9881593003934889
mae_train = 0.08400118700224841; mae_test = 0.0857056572122495
mape_train = 31.94253200514306; mape_test = 34.164903740505004
```

Исходя из полученных результатов, можно сделать вывод, что для данного датасета следует применять линейную регрессию и её модификации.

Далее следует подобрать наилучшие параметры Lasso регрессии с помощью кросс-валидации, реализованной в функции print_best_estimator. При вариации коэффициента alpha от 0.001 до 1.0 с шагом 0.001 наилучшее значение коэффициента детерминации оказалось равным 0.9888599729933663 при alpha равном 0.001.

Следовательно, коэффициент alpha нужно уменьшить. Теперь коэффициент alpha варьировался от 0.000001 до 0.001 с шагом 0.000001. Тогда коэффициент детерминации оказался равным 0.9888599729933663 при alpha равном 4e-5. В данном контексте данную модель можно считать наиболее оптимальной. Реализация вызова функции кросс-валидации представлена в Листинге 3.3.3, обучение, тестирование и расчёт метрик представлены в Листинге 3.3.4, метрики – в Листинге 3.3.5.

Листинг 3.3.3 – Подбор наиболее оптимального значения alpha для Lasso регрессии

```
print_best_estimator(X_train, y_train, Lasso(), {'alpha': np.arange(0.000001, 0.001,
↪   0.000001)})
```

Листинг 3.3.4 – Реализация обучения, тестирования и вывода метрик Lasso регрессии

```
lasso = Lasso(4e-5)
lasso.fit(X_train, y_train)
models_info['Lasso'] = [lasso.coef_, lasso.intercept_ ]
y_pred_train = lasso.predict(X_train)
y_pred_test = lasso.predict(X_test)
compute_regression_metrics(y_pred_train, y_pred_test, 'Lasso')
```

Листинг 3.3.5 – Метрики Lasso регрессии

```
r2_train = 0.9888937186864745; r2_test = 0.9881584584204298
mae_train = 0.08400105352849233; mae_test = 0.08570874768496263
mape_train = 31.94250596510308; mape_test = 34.16484728030022
```

Далее следует подобрать наилучшие параметры Ridge регрессии с помощью кросс-валидации, реализованной в функции print_best_estimator. Реализация вызова функции кросс-валидации представлена в Листинге 3.3.6, результат её выполнения – в Листинге 3.3.7, обучение, тестирование и расчёт метрик представлены в Листинге 3.3.8, метрики – в Листинге 3.3.9.

Листинг 3.3.6 – Подбор наиболее оптимального значения alpha для Ridge регрессии

```
print_best_estimator(X_train, y_train, Ridge(), {'alpha': np.arange(0.001, 1.0, 0.001)})
```

Листинг 3.3.7 – Результат подбора наиболее оптимального значения alpha для Ridge регрессии

```
Ridge(alpha=0.156)
R2 = 0.9888645292903895
```

Листинг 3.3.8 – Реализация обучения, тестирования и вывода метрик Ridge регрессии

```
ridge = Ridge(alpha=0.156)
ridge.fit(X_train, y_train)
models_info['Ridge'] = [ridge.coef_, ridge.intercept_]
y_pred_train = ridge.predict(X_train)
y_pred_test = ridge.predict(X_test)
compute_regression_metrics(y_pred_train, y_pred_test, 'Ridge')
```

Листинг 3.3.9 – Метрики Ridge регрессии

```
r2_train = 0.9888937260739591; r2_test = 0.9881592505503204
mae_train = 0.08400121502653807; mae_test = 0.0857057631948808
mape_train = 31.94192037910659; mape_test = 34.164299620536895
```

Далее следует подобрать наилучшие параметры ElasticNet регресси с помощью кросс-валидации, реализованной в функции print_best_estimator. При вариации коэффициентов alpha и l1_ratio от 0.1 до 1.0 с шагом 0.1 наилучшее значение коэффициента детерминации оказалось равным 0.9796126597252524 при alpha и l1_ration равными 0.1. Следовательно, гиперпараметры нужно уменьшить. В последующие разы начальные гиперпараметры уменьшались в 10 раз, как и шаг, и каждый раз коэффициент детерминации рос, а коэффициенты принимали минимально возможные значения. Но при вариации коэффициентов от 0.0001 до 0.001 с шагом 0.0001 наилучшее значение коэффициента детерминации оказалось равным 0.9888645242817141, alpa равно 0.001, l1_ratio – 0.009. Данные гиперпараметры являются наиболее оптимальными на данном множестве. Реализация вызова функции кросс-валидации представлена в Листинге 3.3.10, обучение, тестирование и расчёт метрик представлены в Листинге 3.3.11, метрики – в Листинге 3.3.12.

Листинг 3.3.10 – Подбор наиболее оптимальных гиперпараметров для ElasticNet регрессии

```
print_best_estimator(X_train, y_train, ElasticNet(),
 {'alpha': np.arange(0.0001, 0.001, 0.0001), 'l1_ratio': np.arange(0.0001, 0.001, 0.0001)})
```

Листинг 3.3.11 – Реализация обучения, тестирования и вывода метрик ElasticNet регрессии

```
en = ElasticNet(alpha=0.0001, l1_ratio=0.0009)
en.fit(X_train, y_train)
models_info['ElasticNet'] = [en.coef_, en.intercept_]
y_pred_train = en.predict(X_train)
y_pred_test = en.predict(X_test)
compute_regression_metrics(y_pred_train, y_pred_test, 'ElasticNet')
```

Листинг 3.3.12 – Метрики ElasticNet регрессии

```
r2_train = 0.9888937166664861; r2_test = 0.9881590702879347
mae_train = 0.08400133507338792; mae_test = 0.08570615808462072
mape_train = 31.93982664412043; mape_test = 34.1622306342434
```

Также следует протестировать модель, оптимизируемую градиентным спуском. Для надёжности алгоритм прогоняется 10 раз и выбирается тот вектор коэффициентов, который имеет наиболее высокий коэффициент детерминации на тестовой выборке. Реализация представлена в Листинге 3.3.13, метрики модели – в Листинге 3.3.14.

Листинг 3.3.13 – Оптимизация градиентным спуском

```
X_train_tmp = np.insert(X_train, 0, np.ones(len(X_train)), axis=1)
X_test_tmp = np.insert(X_test, 0, np.ones(len(X_test)), axis=1)
w = gradient_descent(X_train_tmp, y_train)
y_pred_test = tf.linalg.matvec(X_test_tmp, w)
max_r2_score = r2_score(y_test, y_pred_test)
best_w = w
for i in range(10):
    w = gradient_descent(X_train_tmp, y_train)
    y_pred_test = tf.linalg.matvec(X_test_tmp, w)
    if r2_score(y_test, y_pred_test) > max_r2_score:
        max_r2_score = r2_score(y_test, y_pred_test)
        best_w = w
y_pred_train = tf.linalg.matvec(X_train_tmp, best_w)
y_pred_test = tf.linalg.matvec(X_test_tmp, best_w)
models_info['GradientDescent'] = [best_w[1:].numpy(), best_w[0].numpy()]
compute_regression_metrics(y_pred_train, y_pred_test, 'GradientDescent')
```

Листинг 3.3.14 – Метрики при оптимизации градиентным спуском

```
r2_train = 0.988893726328376; r2_test = 0.9881594387127457
mae_train = 0.0840010267294998; mae_test = 0.08570523104986089
mape_train = 31.94186385580554; mape_test = 34.16456237331019
```

Для каждой модели была собрана информация и построена таблица, аналогичная п.1-2 (см. Листинге 3.3.15).

Листинг 3.3.15 – Сравнение различных моделей для линейной регрессии

	coeffs	intercept	r2_train	r2_test
LinearRegression	[0.3840430082026589, 0.91...	-0.001075	0.988894	0.988159
Lasso	[0.38400360657702004, 0.91...	-0.001075	0.988894	0.988158
Ridge	[0.3840342474294577, 0.91...	-0.001075	0.988894	0.988159
ElasticNet	[0.3840041689603308, 0.91...	-0.001076	0.988894	0.988159
GradientDescent	[0.38407409312097573, 0.91...	-0.00107	0.988894	0.98816

	mae_train	mae_test	mape_train	mape_test	dw_train	dw_test
LinearRegression	0.084001	0.085706	31.942532	34.164904	1.968227	2.020821
Lasso	0.084001	0.085709	31.942506	34.164847	1.968226	2.020746
Ridge	0.084001	0.085706	31.94192	34.1643	1.968224	2.020803
ElasticNet	0.084001	0.085706	31.939827	34.162231	1.968213	2.020742
GradientDescent	0.084001	0.085704	31.940314	34.163455	1.968217	2.020844

	white_train	white_test	norm_distr_train	norm_distr_test
LinearRegression	0.6366	0.85389	0.55195	0.297029
Lasso	0.631237	0.853622	0.553706	0.297236
Ridge	0.629202	0.853789	0.552715	0.297132
ElasticNet	0.631919	0.853442	0.555341	0.297488
GradientDescent	0.623661	0.854019	0.5508	0.296546

Исходя из табличных данных, можно сделать следующие выводы:

• В целом, все модели имеют практически одинаковые высокие метрики, однако ElascticNet имеет наилучшие значения по mape_train и mape_test. Следовательно, данная модель является наилучшей.

• Во всех моделях отсутствует зависимость остатков друг от друга, для них выполнено условие гомоскедастичности и они нормально распределены.

• Метрики доволно высокие, заменять стандартизацию на нормализацию нет необходимости.

3.4. Проанализируйте полученную модель. Сделайте выводы о значимости/-информативности признаков. Опишите какие проблемы могут возникнуть при применении модели.

ElasticNet имеет высокие метрики, что говорит о хорошей обобщающей способности в рамках данного датасета. Однако стоит отметить, что обучающая и тестовая выборки похожи, что подтверждается малыми значениями коэффициентов alpha и l1_ratio: нет острой необходимости штрафовать обучающую выборку

за большие значения коэффициентов регрессии для адаптивности к тестовым данным, так как тестовая выборка похожа на обучающую. Однако стоит отметить, что при поступлении нетипичных для данного датасета данных, может быть выявлена переобученность модели.

Для данной модели вектор коэффициентов имеет следующий вид: [0.38400417, 0.91832973, 0.0144939 , 0.04153552, 0.02886487]. Значит, успех студента больше всего зависит от предыдущего результата и количества часов сна, что выглядит вполне реалистично. Для проверки коэффициентов регрессии была построена корреляционная матрица (см. Рисунок 3.4.1), на основе которой можно убедиться в равенстве коэффициента линейной регрессии при каждом факторе и коэффициента корреляции данного фактора с зависимой переменной при условии, что стандартные отклонения факторов и зависимой переменной равны 1, а так как данные стандартизированы, то данное условие выполняется. Также, исходя из матричных данных, можно сделать вывод об отсутствии мультиколлинеарности между признаками и о наличии линейности между каждым регрессором и откликом.

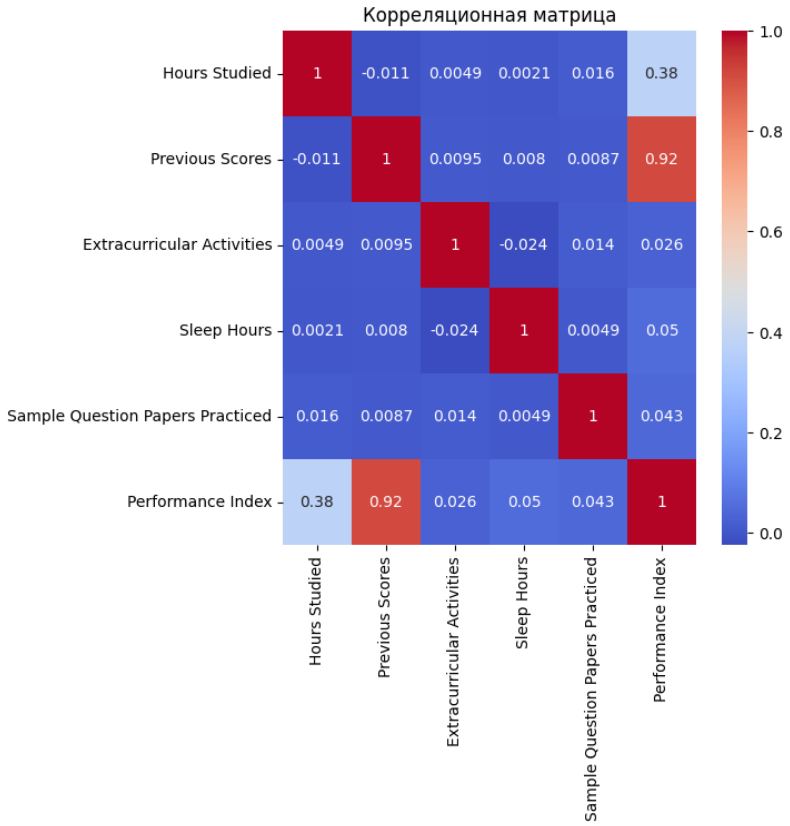

Рисунок 3.4.1 – Корреляционная матрица

Реализация отрисовки диаграмм рассеяния исходных данных и предсказанных представлена в Листинге 3.4.1, отрисовка – на Рисунке 3.4.2.

Листинг 3.4.1 – Реализация отрисовки диаграмм рассеяния исходных данных и предсказанных

```
y_pred = en.predict(students.iloc[:, :-1])
fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(20, 4))
columns = students.columns[:-1]
for i, column in enumerate(columns):
    sns.scatterplot(x=students[column], y=students['Performance Index'],
                    ax=axes[i], s=5, label='Исходные данные')
    sns.scatterplot(x=students[column], y=y_pred, ax=axes[i], c='orange',
                    label='Предсказанные значения', s=5)
fig.suptitle('Диаграммы рассеяния для каждого признака')
plt.tight_layout()
plt.show()
```

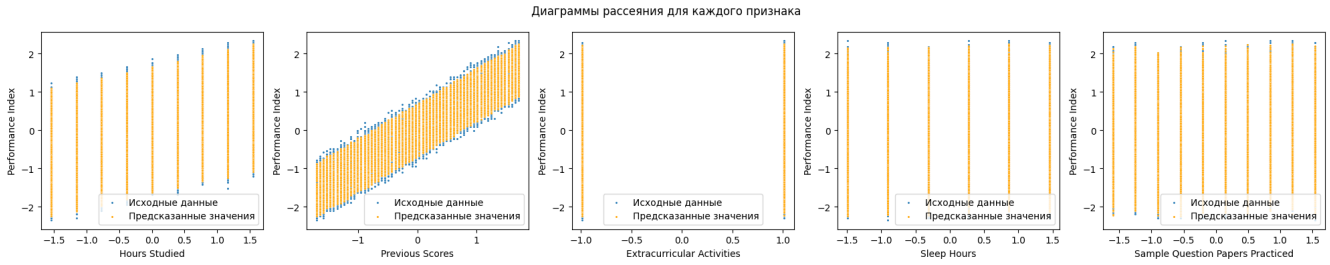

Рисунок 3.4.2 – Диаграммы рассеяния для каждого признака

Вывод.

В ходе выполнения лабораторной работы была изучена линейная, полиномиальная, Lasso, Ridge и ElasticNet регрессии, изучены такие метрики, как коэффициент детерминации, $MAE, MAPE$, а также тесты на гомоскедастичность и независимость остатков регрессии. Был изучен подход подбора коэффициентов регрессии в виде нахождении минимума функции потерь с помощью градиентного спуска и выведена формула для автоматического расчёта скорости обучения на каждой итерации алгоритма.