Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторным работам №1-2 по дисциплине "Методы машинного обучения"

Тема Модель полиномиальной регрессии

Студент Варламова Е. А.

Группа ИУ7-23М

Оценка (баллы) _________________

Преподаватели Солодовников Владимир Игоревич

Москва — 2024 г.

СОДЕРЖАНИЕ

1 Теоретическая часть

1.1 Полиномиальная регрессия

Полиномиальная регрессия – это метод восстановления зависимости между независимыми и зависимыми переменными при помощи полиномиальной функции. Он часто используется для приближения нелинейного поведения данных и улучшения качества предсказаний по сравнению с линейной регрессией. Полиномиальная регрессия позволяет уловить сложные взаимосвязи в данных и учитывать нелинейные зависимости.

Целью данной лабораторной работы является изучение модели полиномиальной регрессии.

Для этого необходимо решить следующие задачи:

- формализовать задачу;
- описать алгоритм работы ПО, решающего поставленную задачу;
- привести особенности реализации ПО, решающего поставленную задачу;
- провести исследование зависимости среднеквадратичной ошибки регрессии от степени полинома;
- провести исследование зависимости значения функционала эмпирического риска на обучающей и контрольной выборках от степени полинома.

1.2 Постановка задачи

1.2.1 Задача 1

Создать обучающую выборку с использованием функции

$$y(x) = \theta_1 x + \theta_2 \sin(x) + \theta_3 \tag{1.1}$$

с добавлением шума с нормальным распределением.

Построить модель полиномиальной регрессии, аппроксимирующей данные обучающей выборки. Исходить из того, что степень полинома (начальный закон генерации обучающей выборки) неизвестен. Обучение проводить методом наименьших квадратов.

1.2.2 Задача 2

Феномен Рунге – это эффект нежелательных осцилляций, возникающий при использовании полиномов высоких степеней для интерполяции.

Функция:

$$y(x) = \frac{1}{1+25x^2}, x \in [-2, 2] \tag{1.2}$$

Обучающая выборка:

$$S_l : x_i = \frac{4(i-1)}{l-1} - 2, i = 1, \ldots, l \tag{1.3}$$

Контрольная выборка:

$$S_k : x_i = \frac{4(i-0.5)}{l-1} - 2, i = 1, \ldots, l-1. \tag{1.4}$$

Рассчитать функционал эмпирического риска (функционал качества) для обучающей и контрольной выборок (вывести графики). Оценить обобщающую способность (generalization ability). Найти оптимальную степень полинома для аппроксимации.

1.3 Функционал эмпирического риска

Функционал эмпирического риска (empirical risk functional) используется в машинном обучении для измерения качества модели на обучающей выборке. Он представляет собой среднее значение функции потерь (loss function) на обучающих примерах.

Для задачи регрессии, наши данные состоят из пар (x_i, y_i), где x_i - входное значение, а y_i - соответствующее целевое значение. Пусть $h(x)$ - модель, а $\ell(h(x), y)$ - функция потерь. Тогда эмпирический риск $R(h)$ может быть записан следующим образом:

$$R(h) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i), y_i)$$

Здесь N - количество обучающих примеров, и сумма берется по всем парам (x_i, y_i). Функция потерь $\ell(h(x_i), y_i)$ оценивает разницу между предсказанным значением $h(x_i)$ и истинным значением y_i.

В данной работе используется квадратичная функция потерь, а, соотвественно, функционал эмпирического риска равен среднеквадратичной ошибке.

1.4 Обобщающая способность

Обобщающая способность (generalization ability) модели является ее способностью хорошо предсказывать новые, невиданные ранее данные после обучения на имеющемся наборе обучающих данных. Обобщающая способность – это ключевой критерий эффективности модели и важна для того, чтобы избежать переобучения.

Обобщающая способность зависит от сбалансированности модели между точностью на обучающем наборе и способностью обобщаться на новые данные. Модель с хорошей обобщающей способностью сможет давать точные предсказания на новых данных, не привязываясь к особенностям обучающего набора.

Для оценки обобщающей способности модели после обучения ее на обучающем наборе, обычно используют разделение данных на обучающую и тестовую выборки, а также кросс-валидацию. Это помогает оценить, насколько модель способна хорошо предсказывать на новых данных.

1.5 Описание алгоритма

Схема алгоритма, вычисляющего оптимальную степень полинома по обучающей выборке, представлена на рисунке 1.1.

Данный алгоритм используется в обеих задачах, однако во второй задаче среднеквадратичная ошибка вычисляется не на обучающей, а на контрольной выборке.

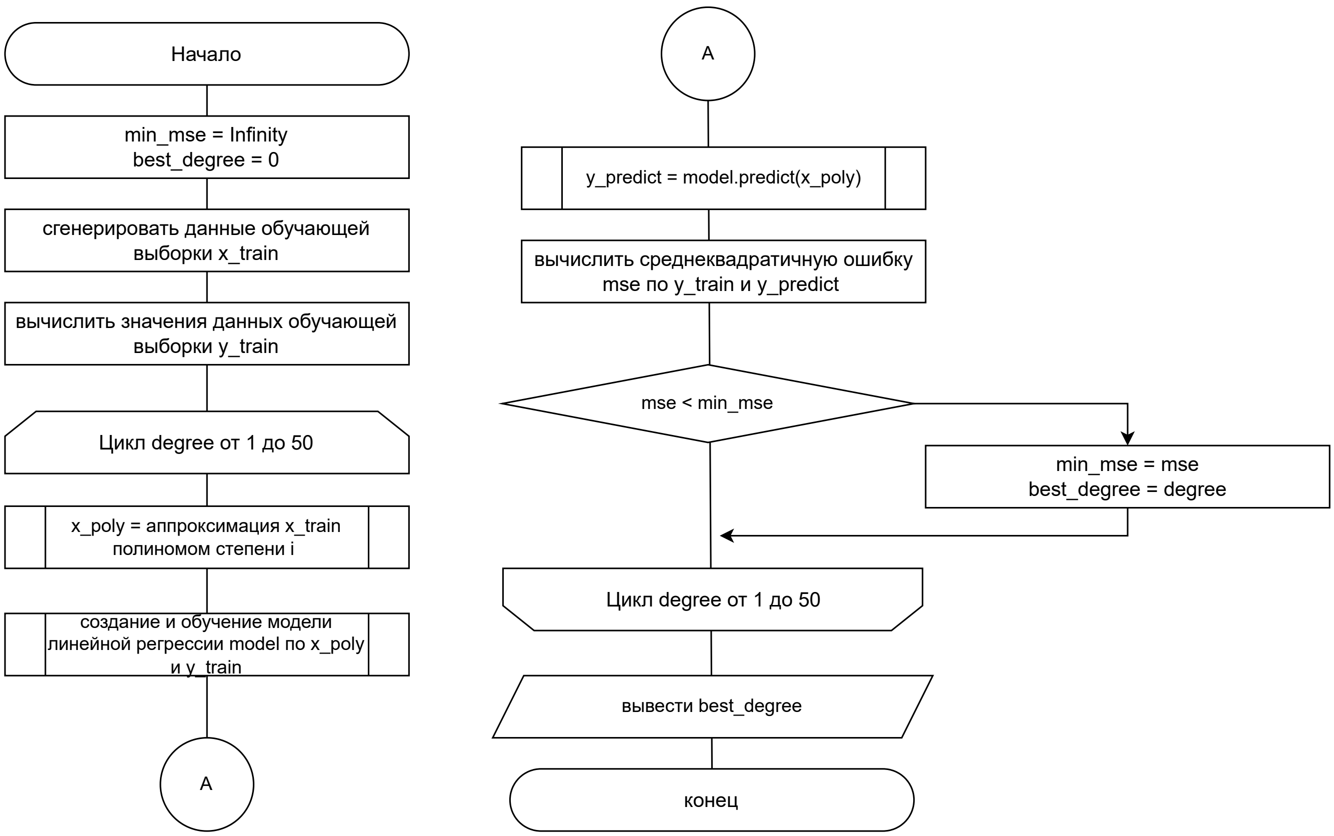


Рис. 1.1: Схема работы алгоритма

2 Практическая часть

2.1 Выбор средств разработки

В качестве языка программирования был использован язык Python, поскольку этот язык кроссплатформенный и для него разработано огромное количество библиотек и модулей, решающих разнообразные задачи.

В частности, имеются библиотеки, включающие в себя алгоритмы аппроксимации полиномом и линейной регрессии в библиотеке [1].

Для создания графиков была выбрана библиотека matplotlib [2], доступная на языке Python, так как она предоставляет удобный интерфейс для работы с данными и их визуализации.

2.2 Исследование ПО

2.2.1 Задача 1

В листинге 2.1 представлен код, вычисляющий оптимальную степень полинома по обучающей выборке и рисует зависимость среднеквадратичной ошибки модели по обучающей выборке от степени полинома.

Листинг 2.1: код

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

theta_1 = 2
theta_2 = 1
theta_3 = 0.5

np.random.seed(0)
X_train = np.linspace(0, 10, 100)
y_train = theta_1 * X_train + theta_2 * np.sin(X_train) + theta_3 + np.
    random.normal(0, 0.5, 100)

```

```python
min_mse = float('inf')
best_degree = 0
degrees = list(range(1, 50))
errors = []

for degree in degrees:
    poly_features = PolynomialFeatures(degree=degree)
    X_poly = poly_features.fit_transform(X_train.reshape(-1, 1))
    model = LinearRegression()
    model.fit(X_poly, y_train)

    y_pred = model.predict(X_poly)

    mse = mean_squared_error(y_train, y_pred)
    errors.append(mse)
    if mse < min_mse:
        min_mse = mse
        best_degree = degree

print(f"Optimal degree of poly: {best_degree}")

best_poly_features = PolynomialFeatures(degree=best_degree)
X_poly_best = best_poly_features.fit_transform(X_train.reshape(-1, 1))

best_model = LinearRegression()
best_model.fit(X_poly_best, y_train)

plt.plot(degrees, errors)
plt.xlabel('Degree of poly')
plt.ylabel('Value error')
plt.title('Dependence of value error on degree of poly')
plt.show()
```

На рисунках 2.1-2.2 показан график тренировочных данных и аппроксимирующего их полинома 13 и 20 степеней соответственно. Видим, что полином 13-ой степени лучше аппроксимирует тренировочные данные, чем 20-ой.

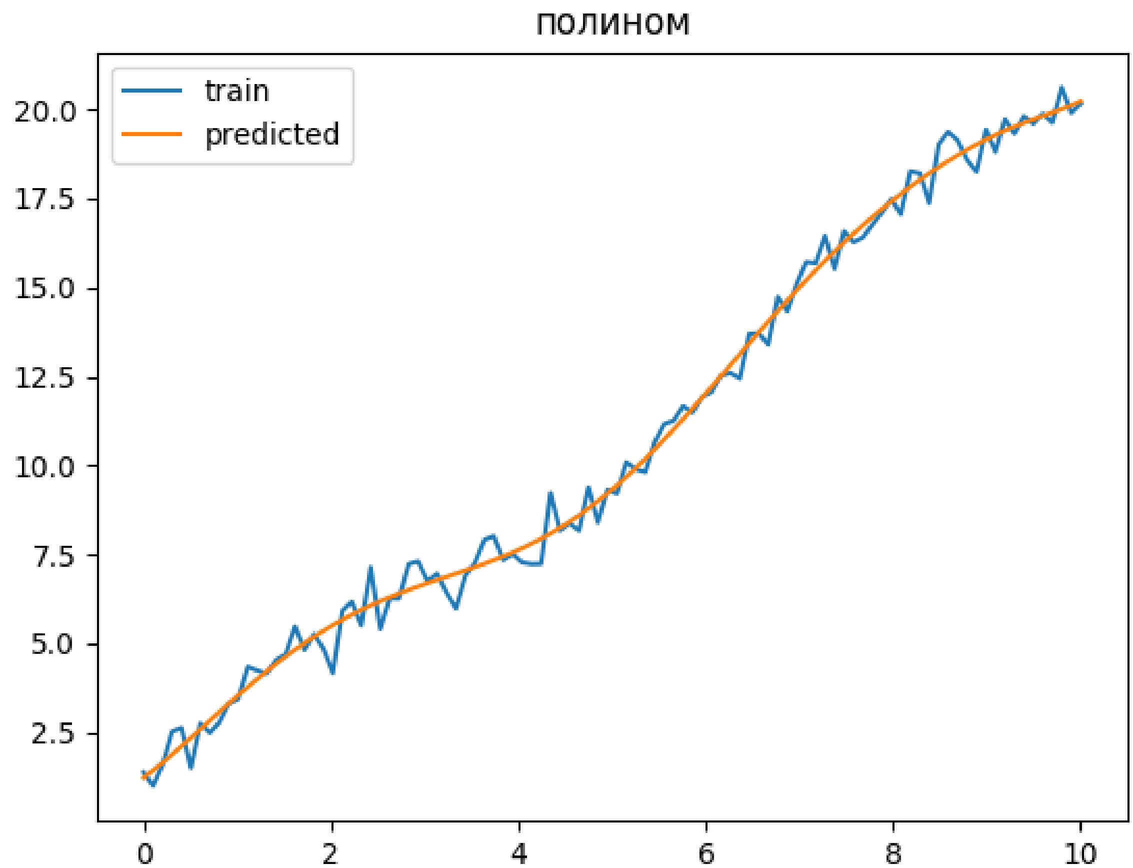


Рис. 2.1:

На рисунке 2.3 показана зависимость значения ошибки от степени полинома. Видно, что увеличение степени полинома необязательно даёт лучшие результаты в смысле уменьшения ошибки.

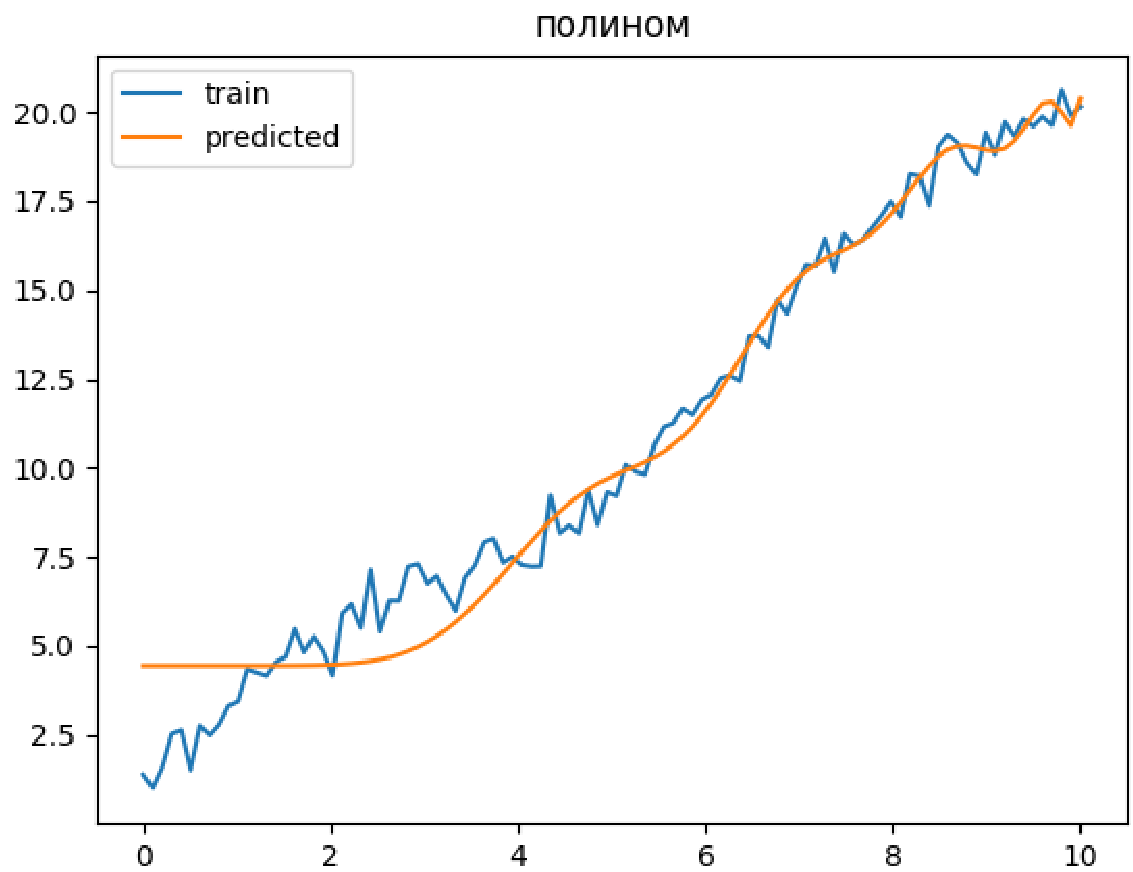


Рис. 2.2:

Результат работы ПО – было вычислено, что оптимальная степень полинома равна 13.

Рис. 2.3:

2.2.2 Задача 2

В листинге 2.2 представлен код, вычисляющий оптимальную степень полинома по контрольной выборке и рисует зависимость среднеквадратичной ошибки модели (функционала эмпирического риска) для обучающей и контрольной выборок от степени полинома.

Листинг 2.2: код

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

def true_function(x):
    return 1 / (1 + 25 * x**2)

l = 21
```

```python
X_train = np.array([4 * (i - 1) / (l - 1) - 2 for i in range(1, l + 1)]).reshape(-1, 1)
X_control = np.array([4 * (i - 0.5) / (l - 1) - 2 for i in range(1, l)]).reshape(-1, 1)
y_train = true_function(X_train)
y_control = true_function(X_control)

def fit_polynomial_regression(X, y, degree):
    poly_features = PolynomialFeatures(degree=degree)
    X_poly = poly_features.fit_transform(X)
    model = LinearRegression()
    model.fit(X_poly, y)
    return model, poly_features


def calculate_error(model, poly_features, X, y):
    X_poly = poly_features.transform(X)
    y_pred = model.predict(X_poly)
    return mean_squared_error(y, y_pred)


degrees = np.arange(1, 50)
train_errors = []
control_errors = []

for degree in degrees:
    model, poly_features = fit_polynomial_regression(X_train, y_train, degree)
    train_error = calculate_error(model, poly_features, X_train, y_train)
    control_error = calculate_error(model, poly_features, X_control, y_control)
    train_errors.append(train_error)
    control_errors.append(control_error)

plt.plot(degrees, train_errors, label='Train Error')
plt.plot(degrees, control_errors, label='Control Error')
plt.xlabel('Degree of Polynomial')
plt.ylabel('Mean Squared Error')
plt.title('Error vs Polynomial Degree')
plt.legend()
plt.show()


optimal_degree = degrees[np.argmin(control_errors)]
print(f'Optimal polynomial degree for approximation: {optimal_degree}')
```

На рисунках 2.4-2.7 показан график тренировочных и контрольных данных и аппроксимирующих их полиномов 5, 10, 16 и 20 степеней соответственно.

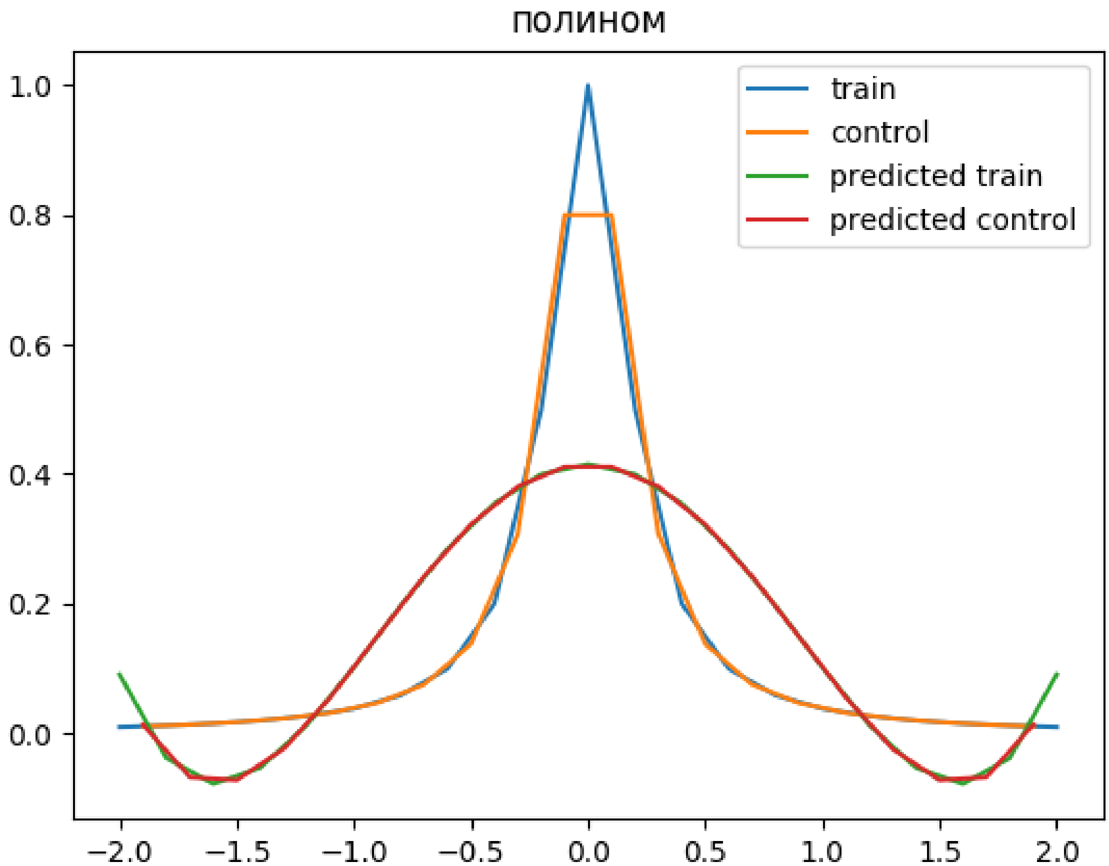

полином
train
control
predicted train
predicted control
1.0
0.8
0.6
0.4
0.2
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Рис. 2.4:

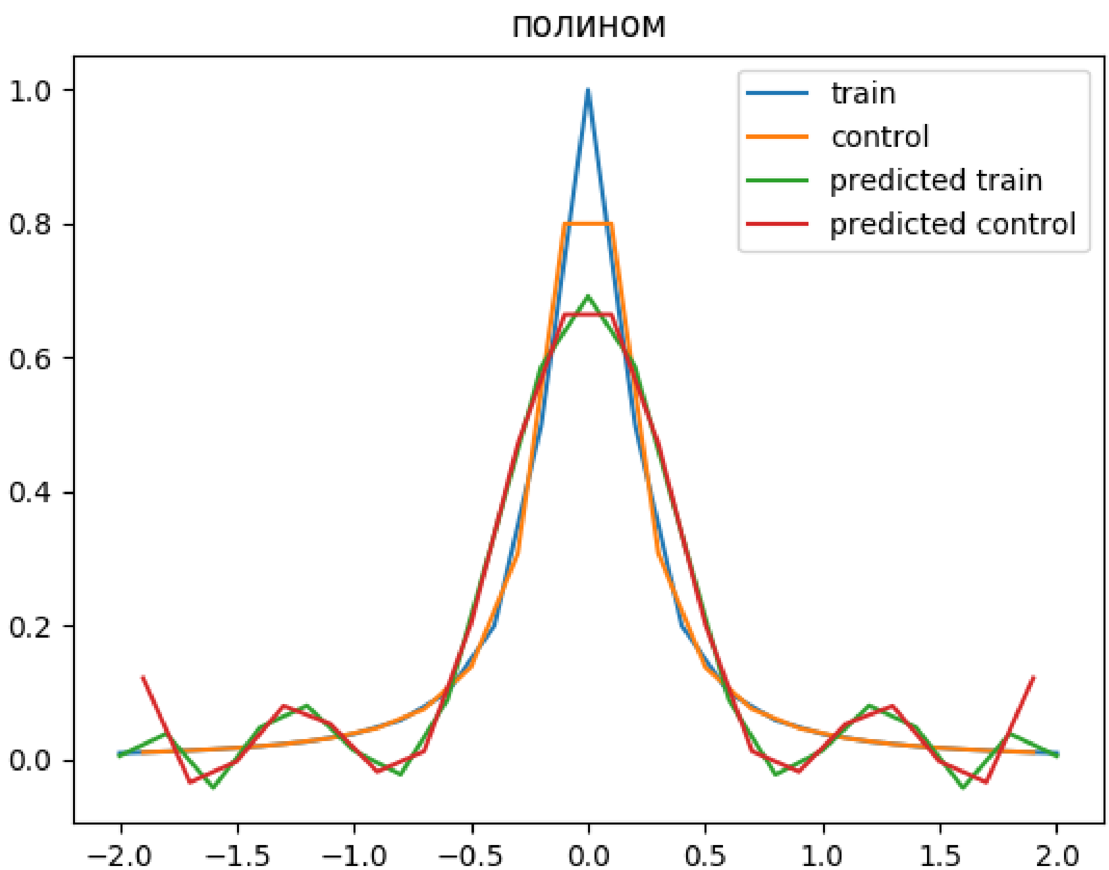
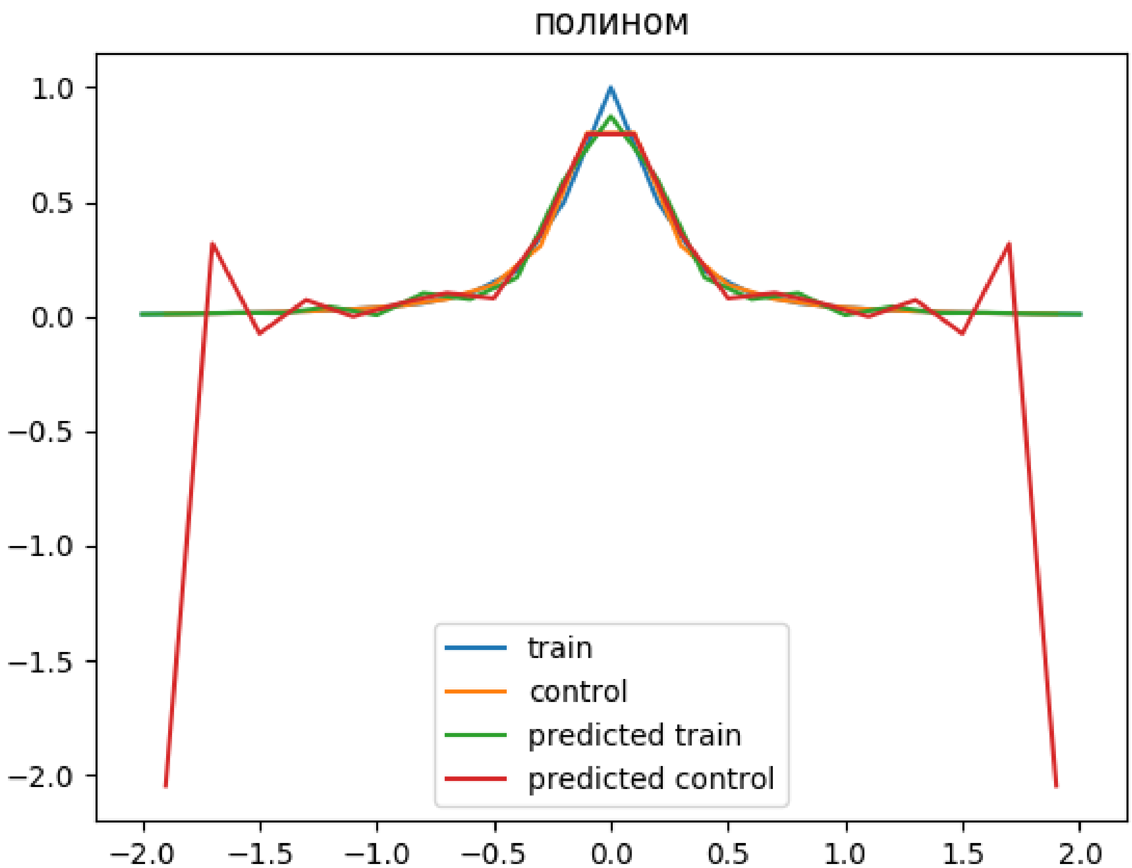

полином
train
control
predicted train
predicted control
1.0
0.8
0.6
0.4
0.2
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Рис. 2.5:



полином
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
train
control
predicted train
predicted control
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Рис. 2.6:

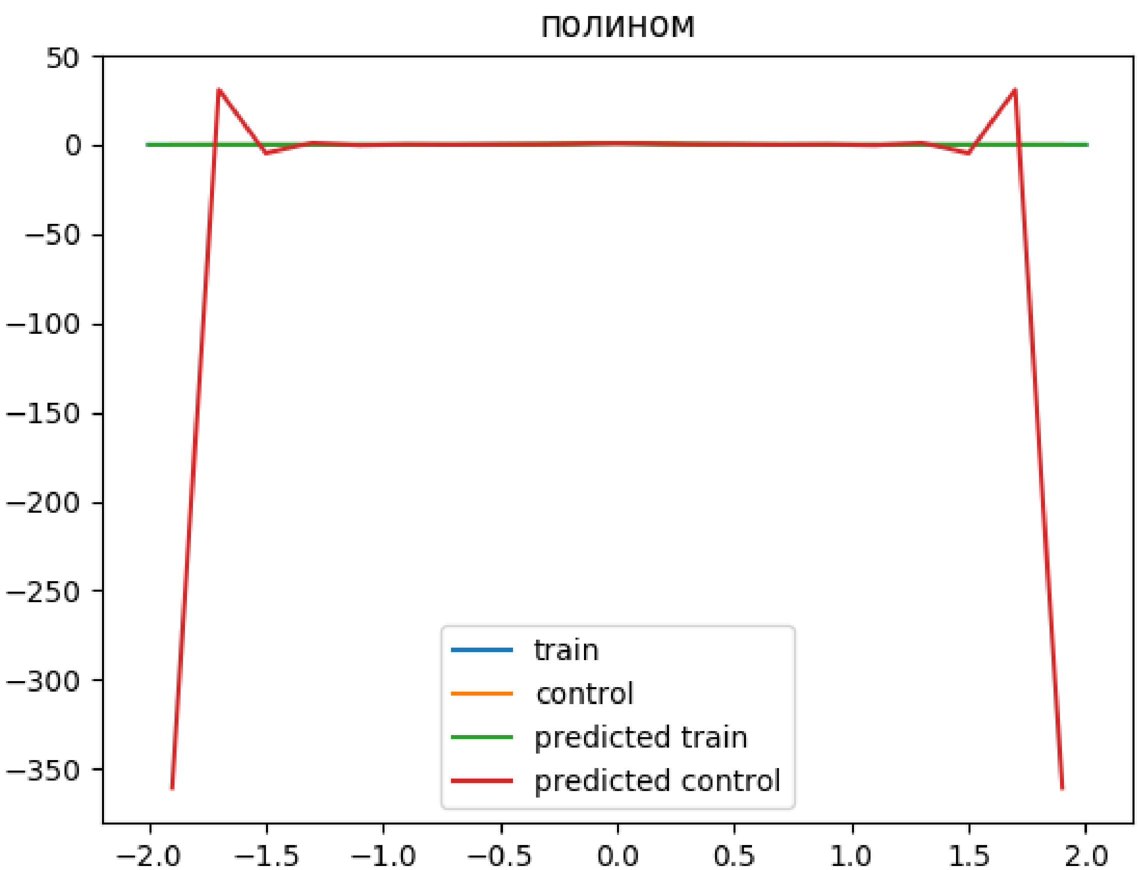

полином
50
0
−50
−100
−150
−200
−250
−300
−350
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
train
control
predicted train
predicted control

Рис. 2.7:

На рисунке 2.8 показана зависимость значения ошибки от степени полинома. Как и в предыдущей задаче, видно, что увеличение степени полинома необязательно даёт лучшие результаты в смысле уменьшения ошибки.

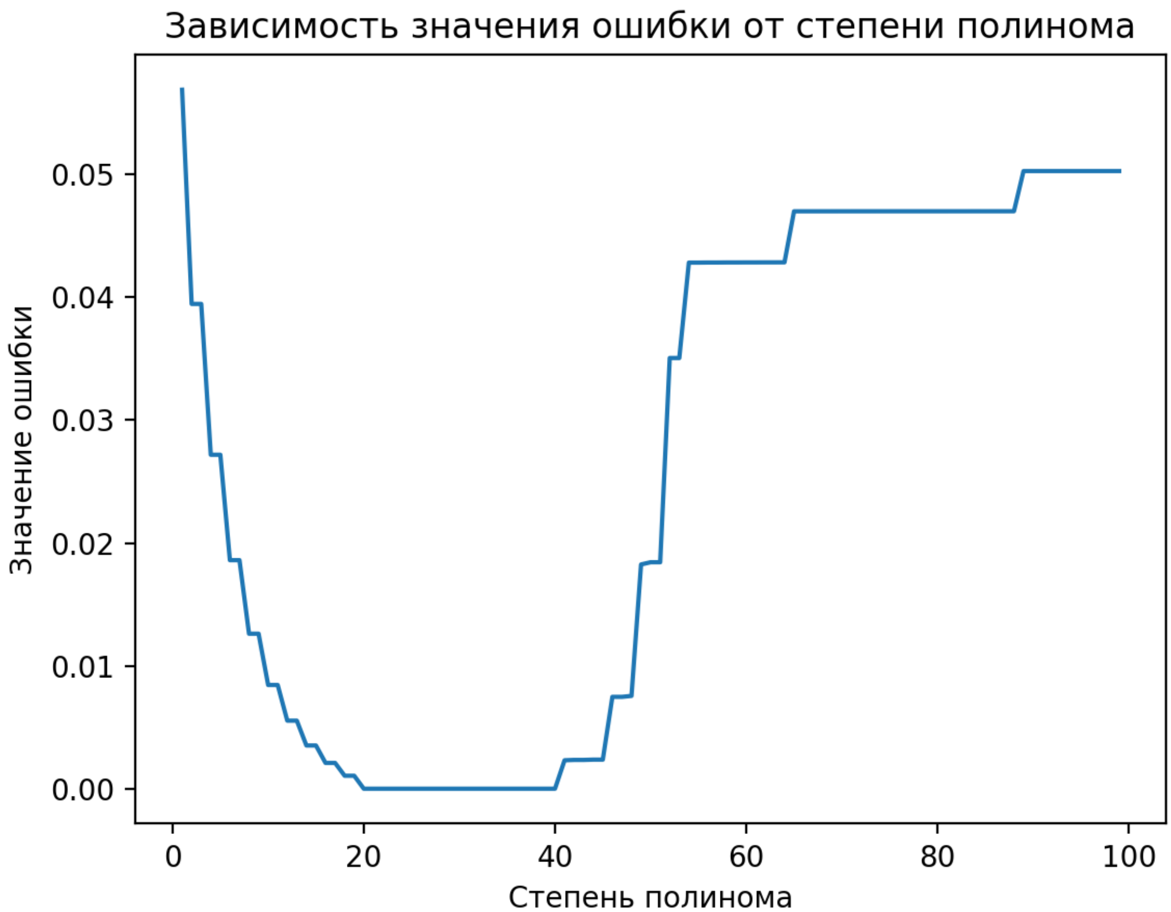


Рис. 2.8:

На рисунке 2.9 показана зависимость значения ошибки от степени полинома для обучающей и контрольной выбоорок. Видим, что для контрольной выборки с определённого значения степени полинома ошибка стремительно растёт, что демонстрирует эффект Рунге – эффект нежелательных осцилляций или колебаний вблизи крайних точек интерполяции при использовании полиномов высоких степеней.

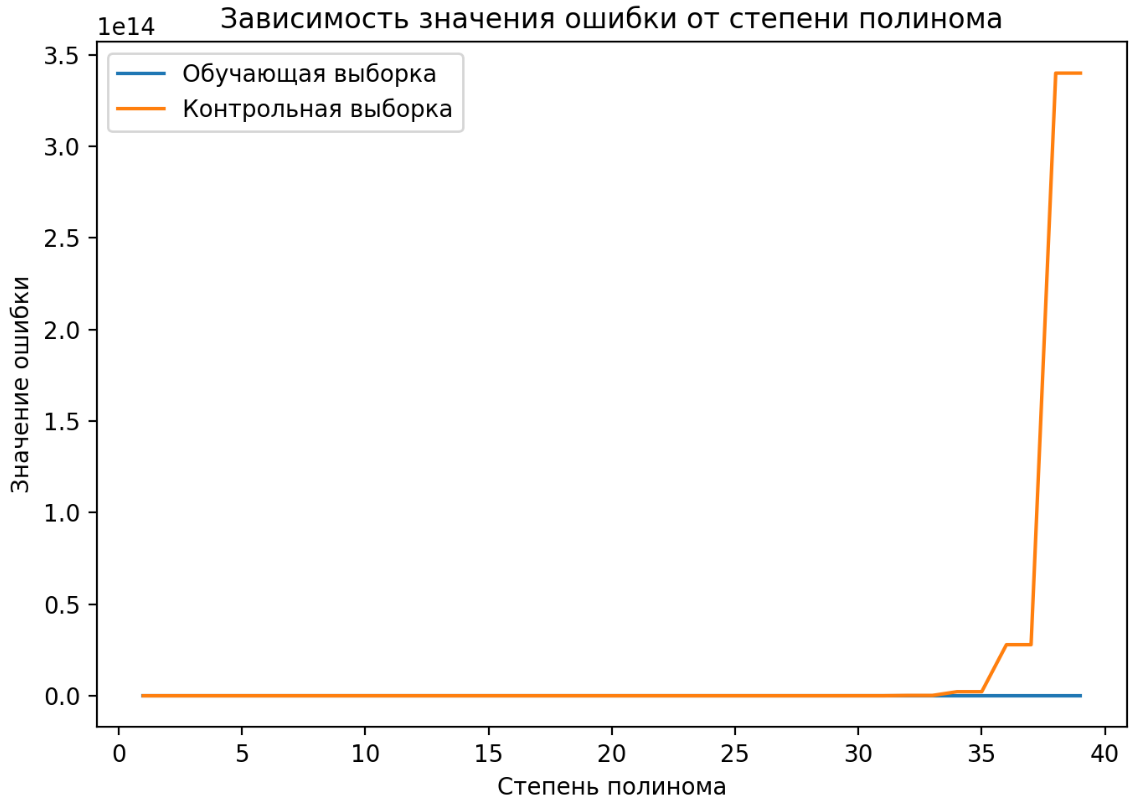


Рис. 2.9:

Рассмотрим интервал низких степеней полинома для получения оптимальной степени полинома на рисунке 2.10.

Видим, что оптимальная степень полинома равна 10.

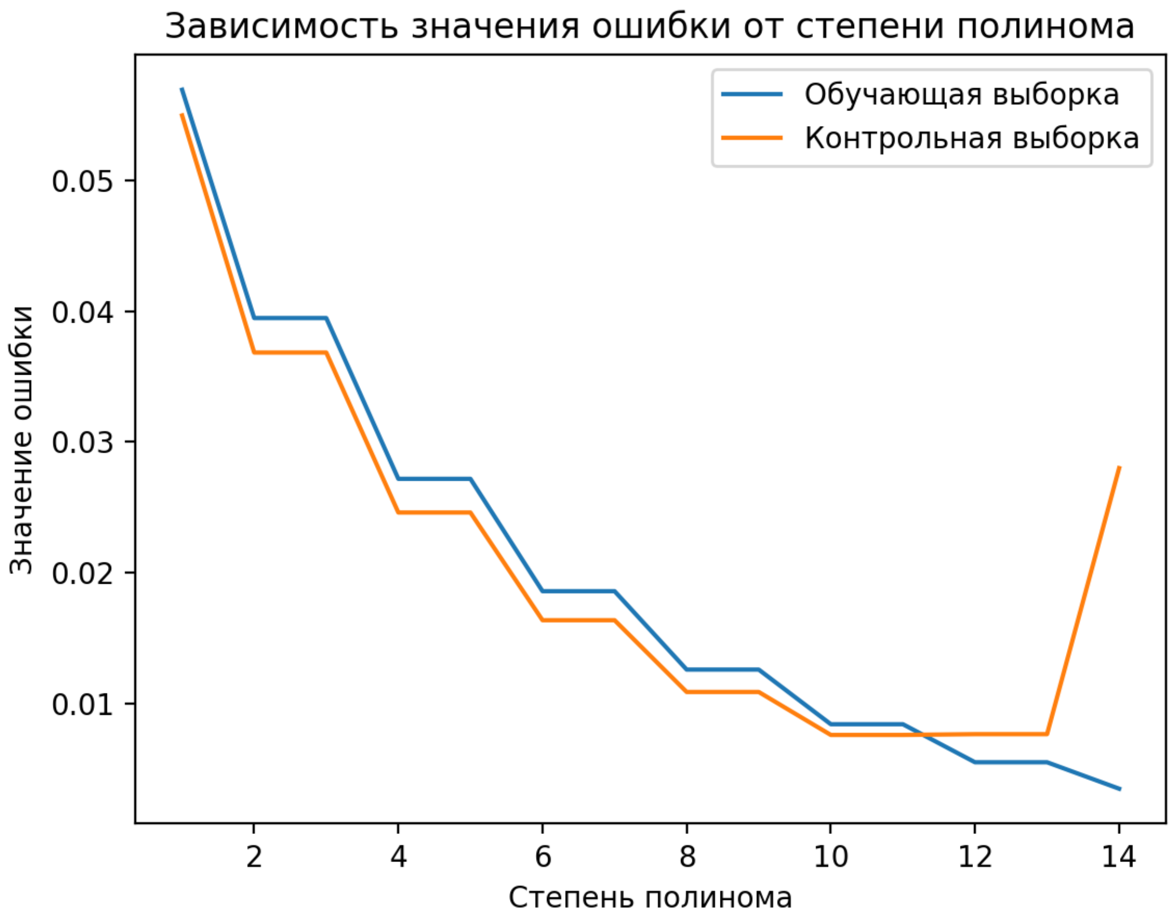

Зависимость значения ошибки от степени полинома
Обучающая выборка
Контрольная выборка
0.05
0.04
0.03
0.02
0.01
Значение ошибки
2
4
6
8
10
12
14
Степень полинома

Рис. 2.10:

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Scikit-learn: Machine learning in Python / F. Pedregosa [и др.]. — 2011.
2. Библиотека визуализации данных matplotlib [Электронный ресурс]. — Режим доступа: URL: https://matplotlib.org (дата обращения: 13.12.2023).