

Регуляризация

В этом уроке мы познакомимся с такой задачей, как **Регуляризация** для борьбы с переобучением машины. Суть в том, что если стараться максимально сильно приблизить к случайным точкам степенную точку, постоянно увеличивая максимальную степень, то очень быстро такая модель будет не эффективно предсказывать другие результаты, и таким образом произойдет переобучение машины на полученных данных. Но как увидеть в тех формулах, которые мы используем, граничный переход от того, что еще можно использовать, и оно будет эффективным, в сторону переобучения, где же она, золотая середина? Давайте разбираться



Для простоты разбора мы не будем изучать новые датасеты, а пойдем по пути наименьшего сопротивления, взяв все данные из [статьи №5](#)

Давайте импортируем их для начала, и отобразим сразу на графике

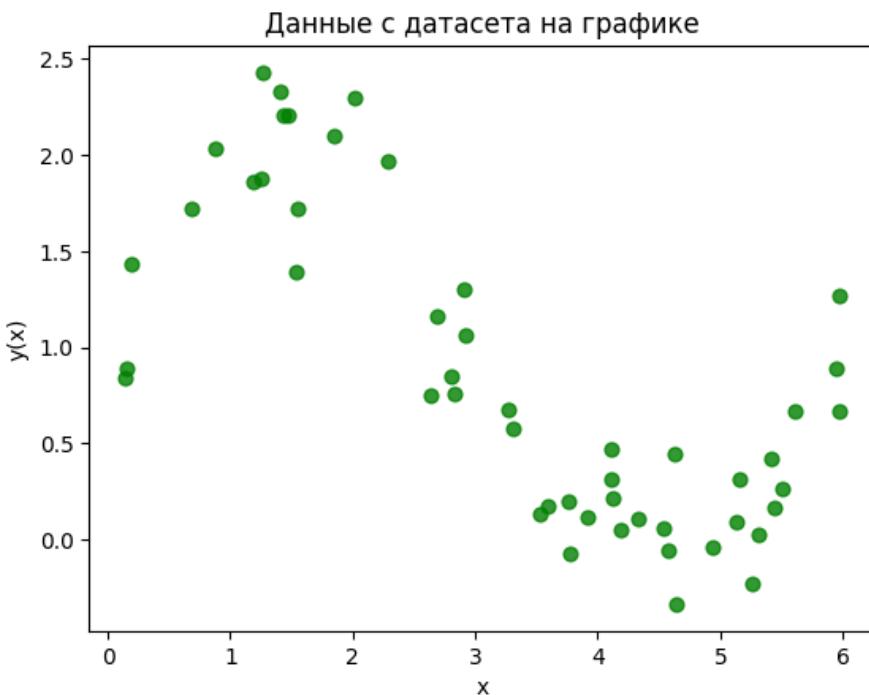
Импорт датасета

```
import pandas as pd
import numpy as np
data=pd.read_csv("3.10_non_linear.csv")
data.head()
```

	x_train	y_train
0	0.138368	0.838812
1	0.157237	0.889313
2	0.188684	1.430040
3	0.685553	1.717309

	x_train	y_train
4	0.874237	2.032588

```
import matplotlib.pyplot as plt
#%matplotlib inline
# основной график
plt.scatter(data.x_train, data.y_train, 40, 'g', 'o', alpha=0.8)
plt.title("Данные с датасета на графике")
plt.xlabel("x")
plt.ylabel("y(x)")
plt.show()
```



Теперь обучим полиномиальную регрессию для трех различных степеней: очень большой, средней, и маленькой. Чтобы сделать это быстро, давайте просто скопируем материал из предыдущей статьи. Кроме прочей информации будем вычислять т.н. *норму вектора* - это просто корень из суммы квадратов коэффициентов линейной регрессии. Чтобы избавиться от корня, будем вычислять квадрат нормы, обозначается как $\|w\|^2$.

```
from sklearn.linear_model import LinearRegression
from numpy.linalg import norm
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
def generate_degrees(source_data: list, degree: int):
    """Функция, которая принимает на вход одномерный массив, а возвращает n-мерный
    Для каждой степени от 1 до degree возводим x в эту степень
    """
    return np.array([
        source_data**n for n in range(1, degree + 1)
    ]).T

def train_polynomial(degree, data, checker):
    """Генерим данные, тренируем модель
    дополнительно рисуем график
    """
    X = generate_degrees(data['x_train'], degree)
    y = data.y_train.values
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=46)
```

```

model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_valid)
y_pred_train = model.predict(X_train)
error_valid = mean_squared_error(y_valid, y_pred)
error_train = mean_squared_error(y_train, y_pred_train)
print(
    "Степень полинома %d\nОшибка на валидации (тесте) %.3f\nОшибка на обучении %.3f" %
    (degree, error_valid, error_train)
)
print("-----")
print("Из-за того, что в процессе тренировки данные были перемешаны, давай их заново мы отсортируем по порядку:\nСортируем...\nСортируем...\nГотово!")
order_test = np.argsort(X_valid[:,0])
print("Индексы от наименьшего к наибольшему для массива X: ",order_test)
print("-----")
plt.scatter(X_valid[:,0][order_test], y_valid[order_test], 40, 'g', 'o', alpha=0.8)
print("Норма вектора весов |t||w| | = %.2f" % (norm(model.coef_)))
# визуализируем решение
x_linspace = np.linspace(data['x_train'].min(), data['x_train'].max(), num=100)
y_linspace = model.predict(generate_degrees(x_linspace, degree))
if checker ==1:
    plt.plot(x_linspace, y_linspace,label=f'Полиномиальная ф-я')
return error_valid, error_train, norm(model.coef_)

degrees = []
valid_errors = []
train_errors = []
w_norm = []

```

Проверим для 3-ей степени:

```

degree = 3
checker=1
error_valid, error_train, coef_norm = train_polynomial(degree, data,checker)

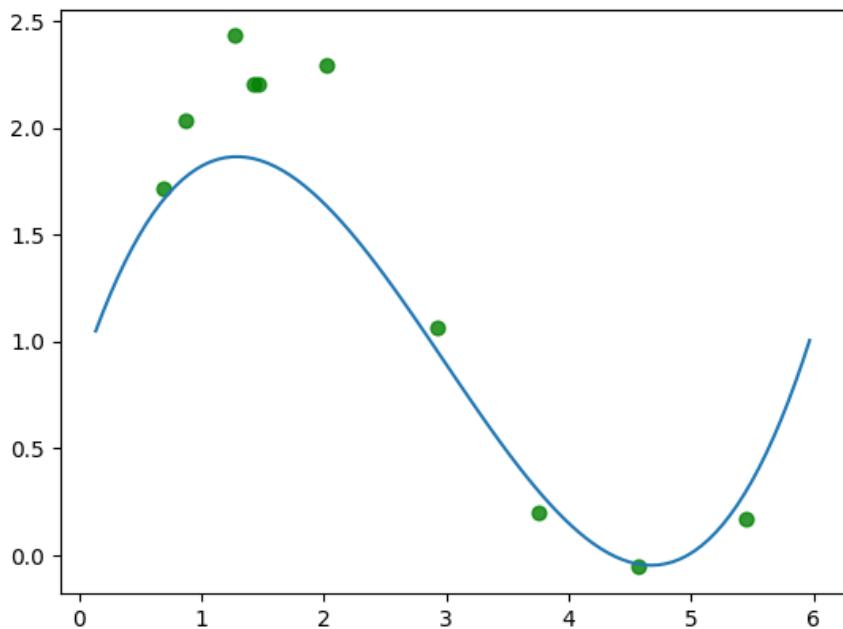
degrees.append(degree)
valid_errors.append(error_valid)
train_errors.append(error_train)
w_norm.append(coef_norm)

```

```

Степень полинома 3
Ошибка на валидации (тесте) 0.111
Ошибка на обучении 0.069
-----
Из-за того, что в процессе тренировки данные были перемешаны, давай их заново мы отсортируем по порядку:
Сортируем...
Сортируем...
Готово!
Индексы от наименьшего к наибольшему для массива X: [2 7 9 0 4 1 8 5 3 6]
-----
Норма вектора весов | |w| | = 2.00

```



```

degree = 5

error_valid, error_train, coef_norm = train_polynomial(degree, data, checker)

degrees.append(degree)
valid_errors.append(error_valid)
train_errors.append(error_train)
w_norm.append(coef_norm)

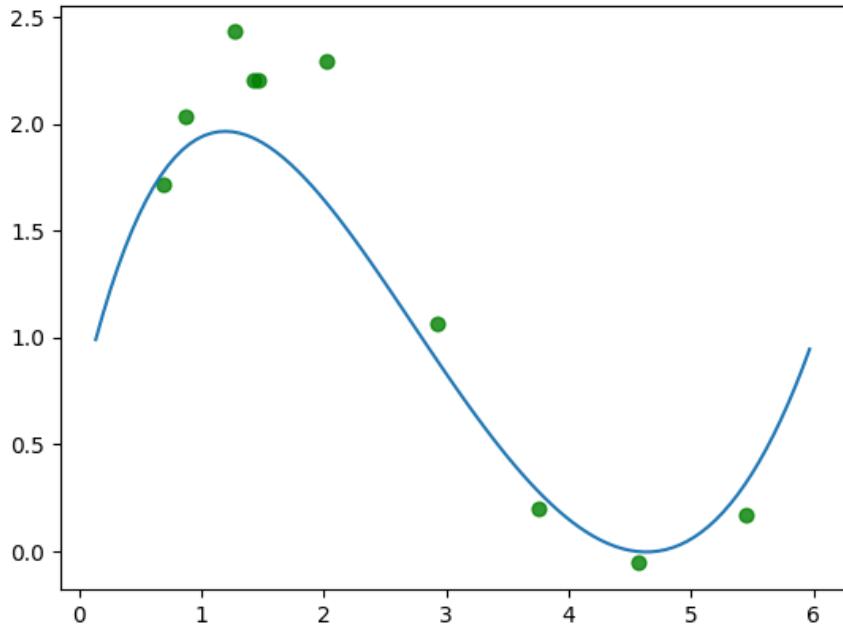
```

Степень полинома 5
 Ошибка на валидации (тесте) 0.090
 Ошибка на обучении 0.067

Из-за того, что в процессе тренировки данные были перемешаны, давай их заново мы отсортируем по порядку:
 Сортируем...
 Сортируем...
 Готово!

Индексы от наименьшего к наибольшему для массива X: [2 7 9 0 4 1 8 5 3 6]

Норма вектора весов ||w|| = 2.99



```

degree =9

error_valid, error_train, coef_norm = train_polynomial(degree, data,checker)

degrees.append(degree)
valid_errors.append(error_valid)
train_errors.append(error_train)
w_norm.append(coef_norm)

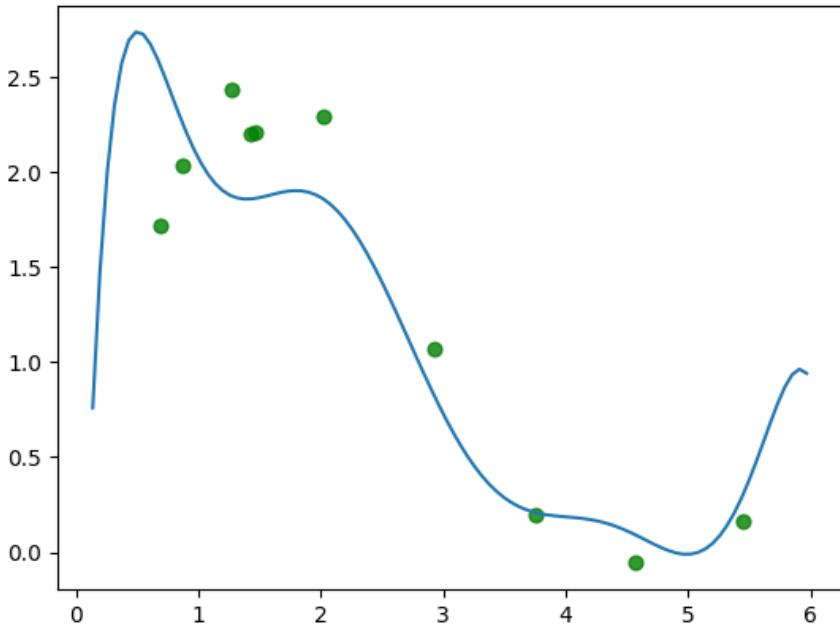
```

Степень полинома 9
Ошибка на валидации (тесте) 0.161
Ошибки на обучении 0.055

Из-за того, что в процессе тренировки данные были перемешаны, давай их заново мы отсортируем по порядку:
Сортируем...
Сортируем...
Готово!

Индексы от наименьшего к наибольшему для массива X: [2 7 9 0 4 1 8 5 3 6]

Норма вектора весов ||w|| = 86.06



Принцип

В этом абстрактном эксперименте: видны следующие закономерности

- степень полинома растёт → ошибка на тренировочных данных падает
- степень полинома растёт → ошибка на валидации растёт
- степень полинома растёт → сумма квадратов коэффициентов регрессии растёт

Ничего себе! Мы детектировали переобучение, о котором говорили в первом модуле. Для наглядности нарисуем график, на котором

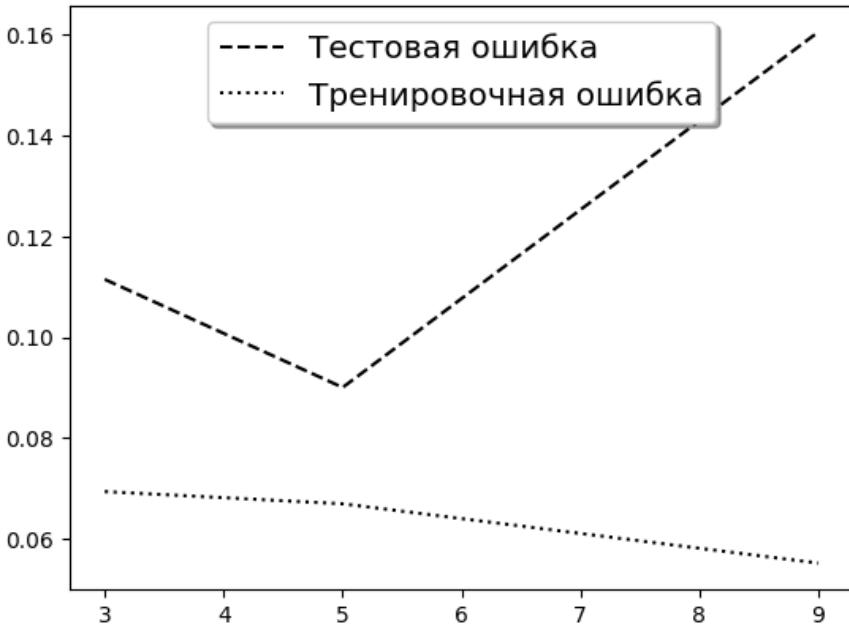
- по оси oX - степень полинома
- по оси oY - ошибка на валидации и ошибка на обучении

```
from matplotlib import pyplot
%matplotlib inline

fig, ax = plt.subplots()
ax.plot(degrees, valid_errors, 'k--', label='Тестовая ошибка')
ax.plot(degrees, train_errors, 'k:', label='Тренировочная ошибка')

legend = ax.legend(loc='upper center', shadow=True, fontsize='x-large')

plt.show()
```



Теория

осмотрите: С уменьшением ошибки на тренировке, мы увеличиваем ошибки на тестировочных данных! А значит на нужно как раз и исходить из нормы.

upd: в зависимости от выборки тестовых значений, и тренировочных (параметр `random_state=46`) кривые ошибок будут разными. Суть заключается в том, что здесь довольно просто понять, на каком этапе нам стоит остановиться (на 5 полиноме ошибка будет наименьшей). Но если даже такого перелома нет, то нужно будет выбрать нам пересечение этих кривых, если оно конечно будет.

И тут появляется вопрос: почему вообще случается переобучение с точки зрения математики? Вспомним, что мы получаем предсказания модели, используя коэффициенты регрессии, в виде

$$\hat{y} = w_0x_0 + \dots + w_nx_n$$

По этой формуле видно, что величина коэффициентов w по модулю сильно влияет на предсказания - чем больше коэффициенты линейной регрессии $w = [w_1, \dots, w_n]$, тем больше таргет y при одинаковом значении x , зависимость линейная.

Вспомним, что аналитическая формула для нахождения коэффициентов регрессии выглядит вот так

$$\bar{w} = (X^T X)^{-1} X^T Y$$

Такое решение получается, когда мы минимизируем функцию ошибок (N - число обучающих примеров):

$$L(y, w) = \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Самое простое решение, которое приходит на ум - каким-то образом "наказывать" нашу модель за слишком большие значения коэффициентов линейной регрессии w . Давайте добавим в функцию качества модели $L(y, w)$ дополнительное слагаемое, которое содержит квадрат нормы вектора весов $\|w\|^2$:

$$L(y, w) = \sum_{i=1}^N (y_i - \hat{y}_i)^2 + \alpha \cdot \sum_{i=1}^k w_i^2 = \sum_{i=1}^N (y_i - \hat{y}_i)^2 + \alpha \cdot \|w\|^2$$

С помощью этого трюка мы минимизируем функцию $L(x, w)$ как бы в двух направлениях:

- уменьшаем ошибку $y - \hat{y}$ (первый компонент формулы)
- следим за нормой коэффициентов регрессии $\|w\|^2$ - модель вынуждена делать коэффициенты как можно меньшими, потому что их размер влияет на функцию ошибки $L(x, w)$

Эксперимент показал: чем больше $\|w\|^2$, тем сильнее переобучение. Следовательно, сделав коэффициенты более маленькими, мы уменьшим переобучение! Кек в

Аналитическое решение новой функции ошибки имеет вид

$$\bar{w} = (X^T X + \alpha E)^{-1} X^T Y$$

Формула максимально похожа на формулу для вычисления аналитических коэффициентов. У тебя что, есть вопросы, про какую формулу я говорю?

Вспомним, что аналитическая формула для нахождения коэффициентов регрессии выглядит вот так

$$\bar{w} = (X^T X)^{-1} X^T Y$$

Единственное отличие - внутри скобок добавилось слагаемое αE , где E - [единичная матрица](#) размерности $k \times k$

$$E = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}$$

Как выбрать между двумя типами регуляризации?

- если фичей очень много (сотни), среди которых есть потенциально не важные - выбирай $L1$
- фичей мало и все они важны - выбирай $L2$

Реализация

Мы познакомимся с регуляризацией $L2$ - такую регуляризацию называют *гребневой*, а реализацию этой модели возьмём из `sklearn.linear_model.Ridge`. Натренируем две модели - одну с коэффициентом регуляризации $\alpha = 0.01$, а вторую - без регуляризации $\alpha = 0.0$:

```
from sklearn.linear_model import Ridge

# plt.scatter(data['x_train'], data['y_train'], 40, 'g', 'o', alpha=0.8, label='data')
a_ridge=0.01
a_linear=0.00
model_ridge = Ridge(alpha=a_ridge)
model_linear = Ridge(alpha=a_linear)
degree = 8

X = generate_degrees(data['x_train'], degree)
y = data['y_train']
# обучаем линейную регрессию с регуляризацией
model_ridge.fit(X, y)
model_linear.fit(X, y)

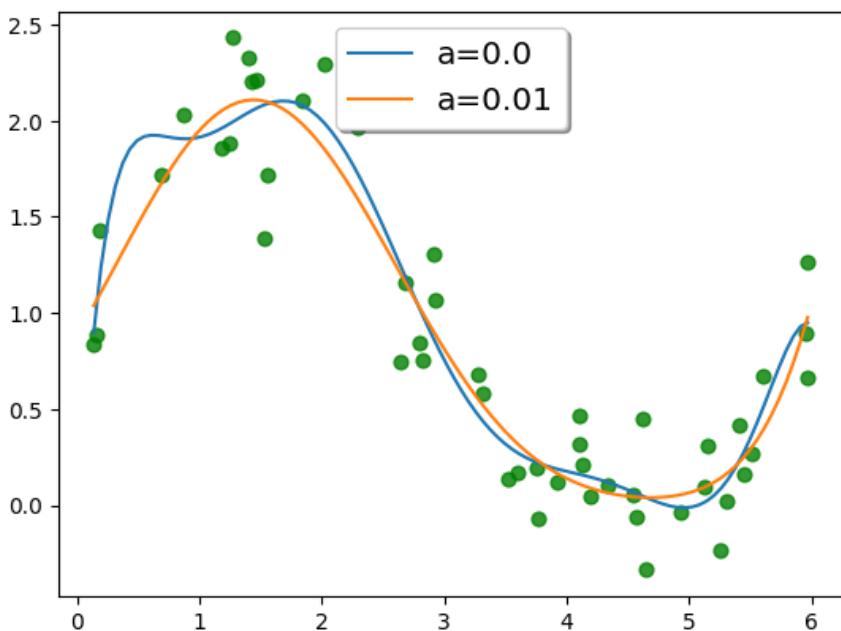
x_linspace = np.linspace(data['x_train'].min(), data['x_train'].max(), num=100)

y_linspace_linear = model_linear.predict(generate_degrees(x_linspace, degree))
y_linspace_ridge = model_ridge.predict(generate_degrees(x_linspace, degree))
from matplotlib import pyplot
%matplotlib inline

fig, ax = plt.subplots()
plt.scatter(data.x_train, data.y_train, 40, 'g', 'o', alpha=0.8)
plt.plot(x_linspace, y_linspace_linear, label=f'a={a_linear}')
plt.plot(x_linspace, y_linspace_ridge, label=f'a={a_ridge}')
legend = ax.legend(loc='upper center', shadow=True, fontsize='x-large')

plt.show()
```

```
C:\Python310\lib\site-packages\sklearn\linear_model\_ridge.py:212: LinAlgWarning: Ill-conditioned matrix
(rcond=1.01818e-17): result may not be accurate.
    return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```



Как видите, добавление весов даже с коэффициентом 0.01 сделало нашу кривую вполне приятной для глаз! Давайте посмотрим на нормы весов (узнать это можно по команде `coef`)

```
print("Норма вектора весов Ridge \|w\| = %.2f" % (norm(model_ridge.coef_)))
print("Норма вектора весов Linear \|w\| = %.2f" % (norm(model_linear.coef_)))
```

Норма вектора весов Ridge	$\ w\ = 1.32$
Норма вектора весов Linear	$\ w\ = 43.19$

Отлично! Но есть вопрос: как лучшим образом подобрать нам тогда альфу?...

Но мы быстро вспоминаем, что уже прошли метод регуляризации, когда сравнивая ошибки при разных степенях полинома понимали, какой из них лучше, и его выводили. Круто! Давайте реализуем этот код заново

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=43
)
print(X_train.shape, X_test.shape)
alphas = []
degree = 8
for i in range(12):
    q = 0.1 * (1.6 ** i)
    alphas.append(q)

best_alpha = alphas[0]
best_rmse = np.infty
from matplotlib import pyplot
%matplotlib inline
for alpha in alphas:
    model_ridge = Ridge(alpha=alpha)
    # обучаем линейную регрессию с регуляризацией
    model_ridge.fit(X_train, y_train)
    y_pred = model_ridge.predict(X_test)
    error = mean_squared_error(y_test, y_pred)
```

```

if error < best_rmse:
    best_rmse = error
    best_alpha = alpha
print("alpha =%.2f Ошибка %.5f" % (alpha, error))
print('-----\nЛучшая модель alpha=%2f с ошибкой RMSE=%5f\n-----' % (best_alpha, best_rmse))
model_ridge = Ridge(alpha=best_alpha)
model_ridge.fit(X_train,y_train)
x_linspace = np.linspace(data['x_train'].min(), data['x_train'].max(), num=100)
y_pred = model_ridge.predict(generate_degrees(x_linspace, degree))

fig, ax = plt.subplots()
plt.scatter(data.x_train, data.y_train, 40, 'g', 'o', alpha=0.8)
plt.plot(x_linspace, y_pred,label=f'a=Ridge ф-я')
plt.title('Регрессия Ridge')
legend = ax.legend(loc='upper center', shadow=True, fontsize='x-large')
degree = 3
checker=1
error_valid, error_train, coef_norm = train_polynomial(degree, data,checker)

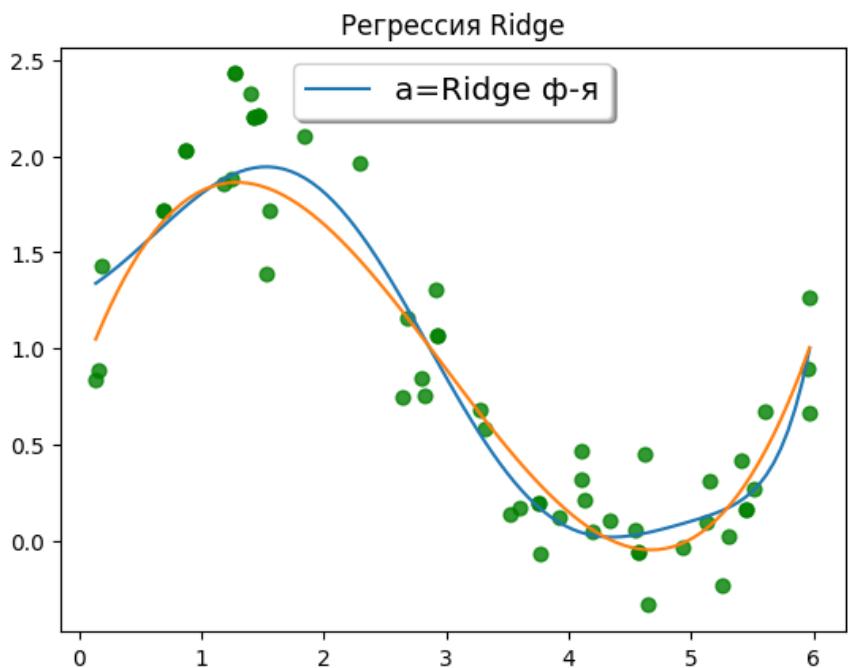
```

```

(40, 8) (10, 8)
alpha =0.10 Ошибка 0.08757
alpha =0.16 Ошибка 0.08432
alpha =0.26 Ошибка 0.08036
alpha =0.41 Ошибка 0.07605
alpha =0.66 Ошибка 0.07204
alpha =1.05 Ошибка 0.06906
alpha =1.68 Ошибка 0.06765
alpha =2.68 Ошибка 0.06811
alpha =4.29 Ошибка 0.07056
alpha =6.87 Ошибка 0.07516
alpha =11.00 Ошибка 0.08192
alpha =17.59 Ошибка 0.09048

-----
Лучшая модель alpha=1.68 с ошибкой RMSE=0.06765
-----
Степень полинома 3
Ошибка на валидации (тесте) 0.111
Ошибка на обучении 0.069
-----
Из-за того, что в процессе тренировки данные были перемешаны, давай их заново мы отсортируем по порядку:
Сортируем...
Сортируем...
Готово!
Индексы от наименьшего к наибольшему для массива X: [2 7 9 0 4 1 8 5 3 6]
-----
Норма вектора весов      ||w|| = 2.00

```



Конец

Как видите, сегодня мы разобрались в том, что такое регрессия Риджа, по английски это Ridge, и что она позволяет учитывать веса точек при нахождении наилучшей реализации машинного обучения
Ну а так как я устал писать это дело 3 рабочих дня, то я просто и резко закончу