



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 лабораторной работе №2 по дисциплине "Методы машинного обучения"

Тема Модель полиномиальной регрессии - Регуляризация.

Студент Варламова Е. А.

Группа ИУ7-23М

Оценка (баллы) _____

Преподаватели Солодовников Владимир Игоревич

СОДЕРЖАНИЕ

1	Теоретическая часть	3
1.1	Полиномиальная регрессия. Регуляризация.	3
1.2	Постановка задачи	3
1.3	Функционал эмпирического риска	4
1.4	Регуляризация	5
1.5	Описание алгоритма	6
2	Практическая часть	7
2.1	Выбор средств разработки	7
2.2	Исследование ПО	7
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	13

1 | Теоретическая часть

1.1 Полиномиальная регрессия. Регуляризация.

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.

Регуляризация в статистике и машинном обучении – метод добавления некоторых дополнительных ограничений к условию с целью предотвратить переобучение. Чаще всего эта информация имеет вид штрафа за сложность модели.

Целью данной лабораторной работы является применение регуляризации к 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

Функция:

$$y(x) = \frac{1}{1 + 25x^2}, x \in [-2, 2] \quad (1.1)$$

Обучающая выборка:

$$S_l : x_i = \frac{4(i-1)}{l-1} - 2, i = 1, \dots, l \quad (1.2)$$

Контрольная выборка:

$$S_k : x_i = \frac{4(i-0.5)}{l-1} - 2, i = 1, \dots, l-1. \quad (1.3)$$

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.

Для оптимальной модели полиномиальной регрессии, а также для модели полиномов меньшей и большей степеней (± 3) вывести значения коэффициентов полинома (всего 3 полинома).

К выбранным моделям (полиномам соответствующих степеней) применить метод регуляризации с использованием гребневой регрессии (ридж-регрессии) и Лассо-регрессии. Вывести значения коэффициентов полинома. Повторить для различных значений параметра .

Рассчитать функционал эмпирического риска (функционал качества) для всех полученных моделей на обучающей и контрольной выборках (вывести графики).

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, соответственно, функционал эмпирического риска равен среднеквадратичной ошибке.

1.4 Регуляризация

Регуляризация в статистике и машинном обучении – метод добавления некоторых дополнительных ограничений к условию с целью предотвратить переобучение. Чаще всего эта информация имеет вид штрафа за сложность модели.

Если выбрана излишне сложная модель при недостаточном объеме данных, то в итоге может быть получена модель, которая хорошо описывает обучающую выборку, но не обобщается на тестовую. Переобучение в большинстве случаев проявляется в том, что итоговые модели имеют слишком большие значения параметров. Одним из способов борьбы с негативным эффектом излишнего подстраивания под данные – использование регуляризации, т.е. добавление некоторого штрафа за большие значения коэффициентов у линейной модели. Тем самым запрещаются слишком "резкие" изгибы, и предотвращается переобучение.

Наиболее часто используемые виды регуляризации – L_1 и L_2 , а также их линейная комбинация – эластичная сеть.

В представленных ниже формулах для эмпирического риска Q приняты следующие обозначения: L – функция потерь, β – вектор параметров $g(x, \beta)$ из модели алгоритма, λ – неотрицательный гиперпараметр (коэффициент регуляризации).

Если в качестве функционал качества используется сумма квадратов остатков (Residual Sum of Squares – RSS), тогда изначально:

$$L(y_i, g(x_i, \beta)) = (g(x_i, \beta) - y_i)^2 \quad (1.4)$$

$$RSS = Q(\beta, X^l) = \sum_{i=1}^l (L(y_i, g(x_i, \beta))) = \sum_{i=1}^l (g(x_i, \beta) - y_i)^2 \quad (1.5)$$

L_2 -регуляризация (ridge regularization) или регуляризация Тихонова (Tikhonov regularization):

$$Q(\beta, X^l) = \sum_{i=1}^l L(y_i, g(x_i, \beta)) + \lambda \sum_{j=1}^n \beta_j^2 \quad (1.6)$$

Минимизация регуляризованного соответствующим образом эмпирического риска приводит к выбору такого вектора параметров β , которое не слишком

сильно отклоняется от нуля. В линейных классификаторах это позволяет избежать проблем мультиколлинеарности и переобучения.

L_1 -регуляризация (lasso regularization) или регуляризация через манхэттенское расстояние:

$$Q(\beta, X^l) = \sum_{i=1}^l L(y_i, g(x_i, \beta)) + \lambda \sum_{j=1}^n |\beta_j| \quad (1.7)$$

Данный вид регуляризации также позволяет ограничить значения вектора β . Однако, к тому же он обладает интересным и полезным на практике свойством – обнуляет значения некоторых параметров, что в случае с линейными моделями приводит к отбору признаков.

1.5 Описание алгоритма

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

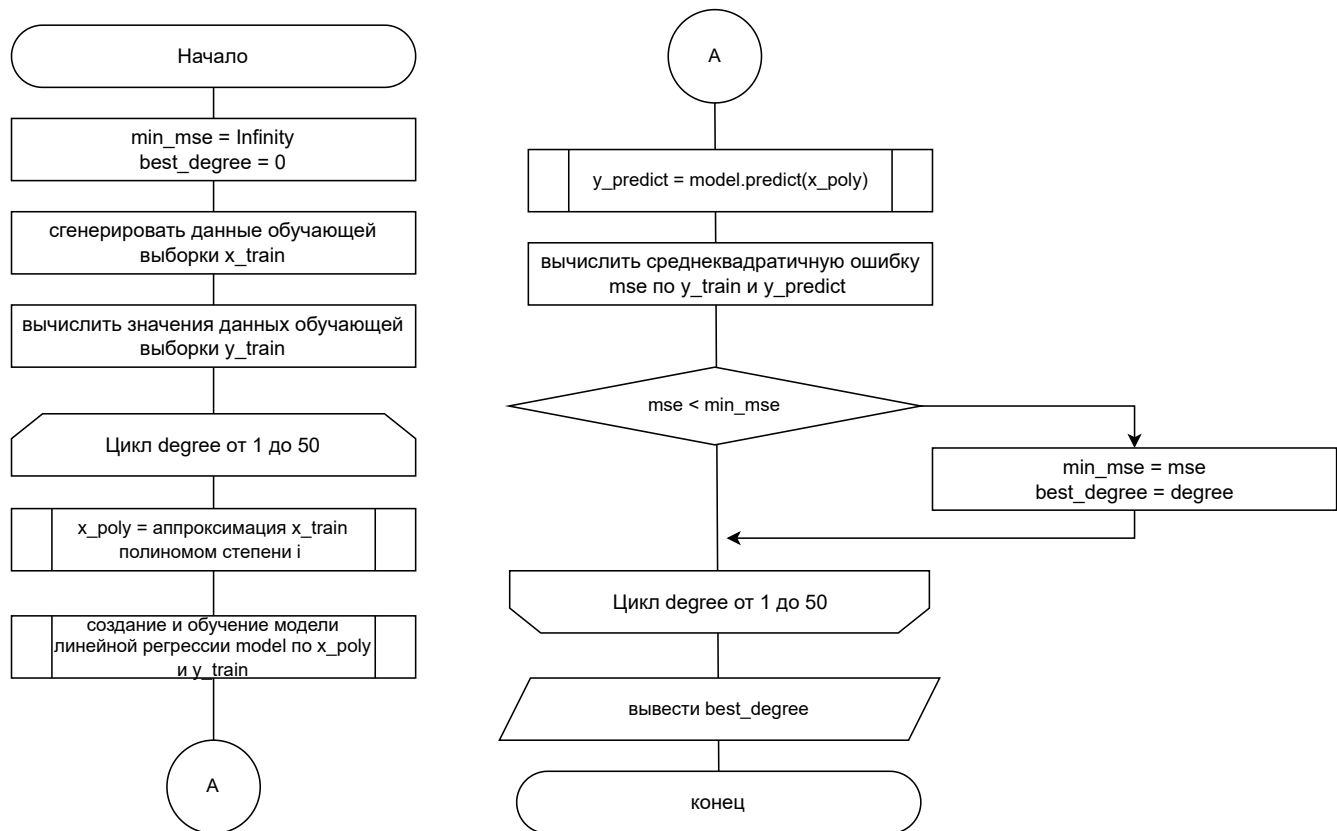


Рис. 1.1: Схема работы алгоритма

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, линейной регрессии и регуляризации 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

В листинге 2.1 представлен код, выводящий максимальное по модулю значение коэффициентов полинома и значение ошибок на обучающей и контрольной выборках для моделей полиномиальной регрессии без регуляризации и с регуляризацией с помощью Лассо и Ридж методов для степеней полинома 10, 13 и 20 и для разных параметров α . Кроме того, для указанных моделей предусмотрен интерфейс вывода графиков фнукционала эмпирического риска в зависимости от степени полинома с возможностью варьировать параметры методов регуляризации.

Листинг 2.1: код

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.preprocessing import PolynomialFeatures
4 from sklearn.linear_model import LinearRegression
5 from sklearn.metrics import mean_squared_error
6 from sklearn.linear_model import Ridge
7 from sklearn.linear_model import Lasso
8 from matplotlib.widgets import Slider
9 import warnings
10 warnings.filterwarnings("ignore")
```

```

11 def true_function(x):
12     return 1 / (1 + 25 * x**2)
13
14 l = 21
15 X_train = np.array([4 * (i - 1) / (l - 1) - 2 for i in range(1, l + 1)]).
    reshape(-1, 1)
16 X_control = np.array([4 * (i - 0.5) / (l - 1) - 2 for i in range(1, l)]).
    reshape(-1, 1)
17 y_train = true_function(X_train)
18 y_control = true_function(X_control)
19 degrees = [10, 13, 16]
20 def fit_polynomial_regression(X, y, degree, model):
21     poly_features = PolynomialFeatures(degree=degree)
22     X_poly = poly_features.fit_transform(X)
23     model.fit(X_poly, y)
24     return poly_features
25
26
27 def calculate_error(model, poly_features, X, y):
28     X_poly = poly_features.transform(X)
29     y_pred = model.predict(X_poly)
30     return mean_squared_error(y, y_pred), y_pred
31 def get_errs(model, X_train, y_train, X_control, y_control):
32     train_errors = []
33     control_errors = []
34     for degree in degrees:
35         poly_features = fit_polynomial_regression(X_train, y_train, degree,
            model)
36         train_error, y_p_t = calculate_error(model, poly_features, X_train,
            y_train)
37         control_error, y_p_c = calculate_error(model, poly_features,
            X_control, y_control)
38         train_errors.append(train_error)
39         control_errors.append(control_error)
40     return train_errors, control_errors
41
42 def update_ridge(val):
43     train_errors_ridge, control_errors_ridge = get_errs(Ridge(alpha=val),
        X_train, y_train, X_control, y_control)
44     ridge_plot_train.set_ydata(train_errors_ridge)
45     ridge_plot_control.set_ydata(control_errors_ridge)
46     fig.canvas.draw_idle()
47
48 def update_lasso(val):
49     train_errors_lasso, control_errors_lasso = get_errs(Lasso(alpha=val),
        X_train, y_train, X_control, y_control)
50     lasso_plot_train.set_ydata(train_errors_lasso)
51     lasso_plot_control.set_ydata(control_errors_lasso)
52     fig.canvas.draw_idle()
53

```



```

54 def output_errors():
55     models = [LinearRegression(),
56               Lasso(alpha=0.1),
57               Lasso(alpha=10),
58               Lasso(alpha=100),
59               Lasso(alpha=1000),
60               Ridge(alpha=0.1),
61               Ridge(alpha=10),
62               Ridge(alpha=100),
63               Ridge(alpha=1000)]
64     names = ["",
65             ", alpha = 0.1",
66             ", alpha = 10",
67             ", alpha = 100",
68             ", alpha = 1000",
69             ", alpha = 0.1",
70             ", alpha = 10",
71             ", alpha = 100",
72             ", alpha = 1000"]
73     degs = [10, 13, 20]
74     i = 0
75     for model in models:
76         for degree in degs:
77             poly_features = fit_polynomial_regression(X_train, y_train,
78                                                       degree, model)
79             train_error, y_p_t = calculate_error(model, poly_features,
80                                                  X_train, y_train)
81             control_error, y_p_c = calculate_error(model, poly_features,
82                                                    X_control, y_control)
83             print("\nhline {} & {} & {:.3f} & {:.3f} & {:.3f} & {:.3f} \\\\".
84                   format(names[i],
85                           degree,
86                           np.max(np.
87                                   abs(
88                                       y_p_t)),
89                           np.max(np.
90                                   abs(
91                                       y_p_c)),
92                           train_error,
93                           control_error))
94             i += 1
95     output_errors()
96     fig, ax = plt.subplots()
97     ax_slider_ridge = plt.axes([0.1, 0.01, 0.8, 0.03])
98     ax_slider_lasso = plt.axes([0.1, 0.9, 0.8, 0.03])
99     train_errors_lin, control_errors_lin = get_errs(LinearRegression(), X_train,

```

```

    y_train, X_control, y_control)
94 ax.plot(degrees, train_errors_lin, label='
    ')
95 ax.plot(degrees, control_errors_lin, label='
    ')
96
97 train_errors_lasso, control_errors_lasso = get_errs(Lasso(alpha=0.1),
    X_train, y_train, X_control, y_control)
98 lasso_plot_train, = ax.plot(degrees, train_errors_lasso, label='
    ')
99 lasso_plot_control, = ax.plot(degrees, control_errors_lasso, label='
    ')
100
101 train_errors_ridge, control_errors_ridge = get_errs(Ridge(alpha=0.1),
    X_train, y_train, X_control, y_control)
102 ridge_plot_train, = ax.plot(degrees, train_errors_ridge, label='
    ')
103 ridge_plot_control, = ax.plot(degrees, control_errors_ridge, label='
    ')
104
105 slider_ridge = Slider(ax_slider_ridge, 'Ridge', valmin=0, valmax=1000,
    valinit=0.1)
106 slider_ridge.on_changed(update_ridge)
107
108 slider_lasso = Slider(ax_slider_lasso, 'Lasso', valmin=0, valmax=1000,
    valinit=0.1)
109 slider_lasso.on_changed(update_lasso)
110 ax.legend()
111
112 plt.xlabel('
    ')
113 plt.ylabel('
    ')
114 plt.title('
    ')
115 plt.legend()
116 plt.show()

```

На рисунке 2.1 показана зависимость значения ошибки от степени полинома для обучающей и контрольной выборок для разных моделей.

В таблице 2.1 показано, что при применении методов регуляризации максимальный по модулю коэффициент не такой большой, как без регуляризации.

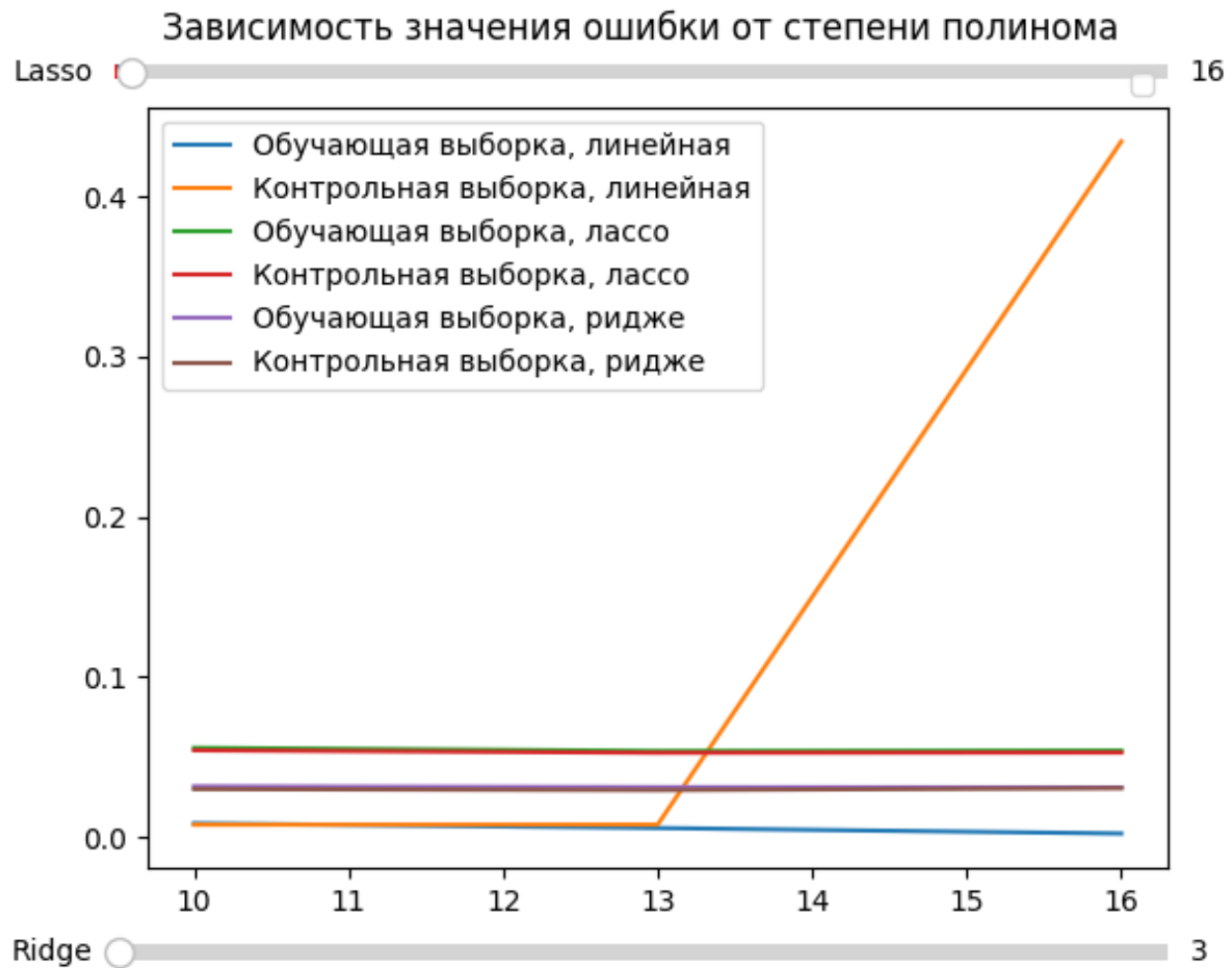


Рис. 2.1:

Таблица 2.1: Результаты работы ПО

Тип модели	Степень полинома	max модуль коэффициента полинома на обучающей выборке	max модуль коэффициента полинома на контрольной выборке	Ошибка на обучающей выборке	Ошибка на контрольной выборке
Линейная	5	0.415	0.411	0.027	0.025
Линейная	10	0.692	0.665	0.008	0.008
Линейная	20	1.000	360.683	0.000	13108.005
Лассо, alpha = 0.1	5	0.199	0.199	0.048	0.047
Лассо, alpha = 0.1	10	0.211	0.211	0.047	0.045
Лассо, alpha = 0.1	20	0.217	0.217	0.046	0.044
Лассо, alpha = 10	5	0.141	0.141	0.057	0.055
Лассо, alpha = 10	10	0.155	0.155	0.054	0.053
Лассо, alpha = 10	20	0.177	0.177	0.052	0.051
Лассо, alpha = 100	5	0.141	0.141	0.057	0.055
Лассо, alpha = 100	10	0.141	0.141	0.057	0.055
Лассо, alpha = 100	20	0.160	0.160	0.054	0.053
Лассо, alpha = 1000	5	0.141	0.141	0.057	0.055
Лассо, alpha = 1000	10	0.141	0.141	0.057	0.055
Лассо, alpha = 1000	20	0.158	0.158	0.055	0.054
Ридж, alpha = 0.1	5	0.408	0.404	0.027	0.025
Ридж, alpha = 0.1	10	0.494	0.486	0.018	0.016
Ридж, alpha = 0.1	20	0.511	1.432	0.016	0.226
Ридж, alpha = 10	5	0.258	0.257	0.040	0.038
Ридж, alpha = 10	10	0.287	0.286	0.036	0.034
Ридж, alpha = 10	20	0.296	0.296	0.036	0.034
Ридж, alpha = 100	5	0.212	0.211	0.046	0.045
Ридж, alpha = 100	10	0.230	0.229	0.044	0.043
Ридж, alpha = 100	20	0.249	0.407	0.042	0.058
Ридж, alpha = 1000	5	0.170	0.170	0.051	0.050
Ридж, alpha = 1000	10	0.201	0.201	0.048	0.046
Ридж, alpha = 1000	20	0.220	0.220	0.046	0.046

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).