

```
In [29]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_diabetes
import ipywidgets as widgets
from scipy.stats import t
from scipy.stats import norm

sns.set('notebook', font_scale=1.7)
%matplotlib inline
```

Для начала вспомним, что означают слова Ridge, Lasso и Elastic. Из курса статистики вы знаете, что такое линейная регрессия.

- **Ridge** - это модель линейной регрессии, которая минимизирует  $\|y - X\theta\|_2^2 + \alpha \cdot \|\theta\|_2^2$ , где  $y$  - истинные значения целевой переменной,  $X$  - матрица "объект-признак",  $\theta$  - параметры модели,  $\alpha$  - параметр регуляризации.
- **Lasso**-регрессия минимизирует  $\|y - X\theta\|_2^2 + \alpha \cdot \|\theta\|_1$
- **Elastic**-регрессия минимизирует  $\|y - X\theta\|_2^2 + \alpha_1 \cdot \|\theta\|_1 + \alpha_2 \cdot \|\theta\|_2^2$ . Таким образом Elastic-регрессия является компромиссом между  $l_1$  и  $l_2$  регуляризацией, позже мы в этом убедимся.

Остановимся подробнее на каждой из моделей в контексте их использования с помощью библиотеки sklearn.

## Ridge-регрессия

Рассмотрим работу линейных моделей на данных о диабете. Мы будем восстанавливать зависимость количественного показателя прогрессирования заболевания от давления, массы, пола, возраста и шести показателей сыворотки крови.

Считываем данные:

```
In [100]: X = load_diabetes()['data']  
y = load_diabetes()['target']  
columns_name = load_diabetes()['feature_names'] + ['target']  
  
data = pd.DataFrame(np.hstack((X, y.reshape(-1, 1))), columns=columns_name)
```

Описание датасета:

```
In [101]: print(load_diabetes()['DESCR'])
```

```
.. _diabetes_dataset:
```

```
Diabetes dataset
```

```
-----
```

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of  $n = 442$  diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

**\*\*Data Set Characteristics:\*\***

:Number of Instances: 442

:Number of Attributes: First 10 columns are numeric predictive values

:Target: Column 11 is a quantitative measure of disease progression one year after baseline

:Attribute Information:

- Age
- Sex
- Body mass index
- Average blood pressure
- S1
- S2
- S3
- S4
- S5
- S6

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times  $\sqrt{n\_samples}$  (i.e. the sum of squares of each column totals 1).

Source URL:

<https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html> (<https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html>)

For more information see:

Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of

Statistics (with discussion), 407–499.  
([https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle\\_2002.pdf](https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf))

```
In [102]: data.describe().round(4)
```

Out[102]:

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
count	442.0000	442.0000	442.0000	442.0000	442.0000	442.0000	442.0000	442.0000	442.0000	442.0000	442.0000
mean	-0.0000	0.0000	-0.0000	0.0000	-0.0000	0.0000	-0.0000	0.0000	-0.0000	-0.0000	152.1335
std	0.0476	0.0476	0.0476	0.0476	0.0476	0.0476	0.0476	0.0476	0.0476	0.0476	77.0930
min	-0.1072	-0.0446	-0.0903	-0.1124	-0.1268	-0.1156	-0.1023	-0.0764	-0.1261	-0.1378	25.0000
25%	-0.0373	-0.0446	-0.0342	-0.0367	-0.0342	-0.0304	-0.0351	-0.0395	-0.0332	-0.0332	87.0000
50%	0.0054	-0.0446	-0.0073	-0.0057	-0.0043	-0.0038	-0.0066	-0.0026	-0.0019	-0.0011	140.5000
75%	0.0381	0.0507	0.0312	0.0356	0.0284	0.0298	0.0293	0.0343	0.0324	0.0279	211.5000
max	0.1107	0.0507	0.1706	0.1320	0.1539	0.1988	0.1812	0.1852	0.1336	0.1356	346.0000

Как видим, данные уже центрированы, кроме целевой переменной. Центрируем ее:

```
In [103]: y = (y - np.mean(y)) / np.std(y)
```

Используем Ridge-регрессию:

```
In [104]: model = Ridge(  
    alpha=1.0, # коэффициент регуляризации, чем больше – тем сильнее регуляризация  
    fit_intercept=True, # использовать ли параметр-остаток при обучении  
    max_iter=100, # максимальное количество итераций в методе оптимизации  
    tol=0.0001, # точность решения  
    solver='auto' # используемый метод оптимизации  
)
```

Рассмотрим подробнее параметр `solver`.

Доступные параметры:

- `auto` выбирает автоматически под тип данных.
- `svd` использует сингулярное разложение матрицы для вычисления коэффициентов регрессии. Более стабильный метод для вырожденных матриц, чем метод `cholesky`.
- `cholesky` (<https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.solve.html>) использует функцию из `scipy.linalg.solve`.
- `sparse_cg` (<https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.cg.html>) использует функцию из `scipy.sparse.linalg.cg`.
- `lsqr` (<https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.sparse.linalg.lsqr.html>) использует функцию из `scipy.sparse.linalg.lsqr`.
- `sag` и `saga` - стохастический градиентный спуск. Метод `saga` заявлен как улучшенная версия метода `sag`. Данные методы являются итеративными и лучше сходятся, когда признаки находятся в одном масштабе (то есть для данных нужно выполнять некоторый препроцессинг, например, при помощи `StandartScaler` из `sklearn`).

```
In [105]: model.fit(X, y)  # обучаем модель
```

```
Out[105]: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=100, normalize=False,
               random_state=None, solver='auto', tol=0.0001)
```

```
In [106]: model.coef_  # значения параметров, полученные после обучения
```

```
Out[106]: array([ 0.38264347, -1.079853 ,  3.97829569,  2.61836869,  0.07673933,
                  -0.38329512, -1.97440417,  1.52341509,  3.41461526,  1.45286195])
```

```
In [107]: model.intercept_  # параметр-остаток
```

```
Out[107]: 3.3595630481817513e-15
```

```
In [108]: model.n_iter_  # число итераций, вернет None если solver не итеративный
```

```
In [109]: model.get_params() # возвращает настройки модели, которые использовались при обучении
```

```
Out[109]: {'alpha': 1.0,  
           'copy_X': True,  
           'fit_intercept': True,  
           'max_iter': 100,  
           'normalize': False,  
           'random_state': None,  
           'solver': 'auto',  
           'tol': 0.0001}
```

Теперь займемся визуализацией. Посмотрим, как ведут себя предсказания в зависимости от изменения параметров модели.

```
In [110]: def linear_predictor(alpha, model, count_of_objects=20):
    """
    Данная функция строит график предсказаний модели на первых count_of_objects объектах датасета.
    """

    if model == 'Ridge':
        model = Ridge(alpha=alpha, max_iter=1000, fit_intercept=True)
    elif model == 'Lasso':
        model = Lasso(alpha=alpha, max_iter=1000, fit_intercept=True)
    elif model == 'Elastic':
        model = ElasticNet(alpha=alpha, max_iter=1000, fit_intercept=True)

    model.fit(X, y)
    y_pred = model.predict(X)

    plt.figure(figsize=(14, 5))

    plt.scatter(np.arange(count_of_objects), y_pred[:count_of_objects],
                color='blue', label='Предсказания модели')
    plt.plot(np.arange(count_of_objects), y_pred[:count_of_objects],
             color='blue')
    plt.scatter(np.arange(count_of_objects), y[:count_of_objects], color='red', label='Истинные значения')

    plt.grid(ls=':')
    plt.xlabel('Номер объекта', fontsize=19)
    plt.xticks(np.arange(count_of_objects))
    plt.ylabel('Предсказание', fontsize=19)
    plt.title('Предсказания на {} объектах'.format(count_of_objects), fontsize=22)
    plt.legend(fontsize=19)
    plt.show()
```

```
In [111]: # создать виджет, но не отображать его
ip = widgets.interactive(
    linear_predictor,
    alpha=widgets.FloatSlider(min=0.0001, max=10, step=0.01, value=1),
    model='Ridge'
);

# отображаем слайдер
display(widgets.HBox(ip.children[:1]))

# отображаем вывод функции
display(ip.children[-1])
ip.update() # чтобы функция запустилась до первого изменения слайдеров

HBox(children=(FloatSlider(value=1.0, description='alpha', max=10.0, min=0.0001, step=0.01),))

Output()
```

## Lasso - регрессия

```
In [116]: model = Lasso(
    alpha=0.001, # коэффициент регуляризации, чем больше – тем сильнее регуляризация
    fit_intercept=True, # использовать ли параметр-остаток при обучении
    max_iter=100, # максимальное количество итераций в методе оптимизации
    tol=0.0001, # точность решения
    precompute=True, # использовать ли предпосчитанную матрицу Грамма для ускорения расчетов
    warm_start=False, # использовать ли результаты последнего обучения для инициализации
    positive=False # если True, то будет подбирать положительные коэффициенты
)
```

```
In [117]: model.fit(X, y) # обучаем модель
```

```
Out[117]: Lasso(alpha=0.001, copy_X=True, fit_intercept=True, max_iter=100,
    normalize=False, positive=False, precompute=True, random_state=None,
    selection='cyclic', tol=0.0001, warm_start=False)
```



```
In [118]: model.coef_ # коэффициенты
```

```
Out[118]: array([-0.          , -2.24853932,  6.7439011 ,  3.69252272, -0.96247026,  
                -0.          , -2.80411346,  0.          ,  6.45216161,  0.55224721])
```

```
In [119]: model.sparse_coef_ # другое представление коэффициентов (без нулевых)
```

```
Out[119]: <1x10 sparse matrix of type '<class 'numpy.float64'>'  
          with 7 stored elements in Compressed Sparse Row format>
```

```
In [120]: print(model.sparse_coef_)
```

```
(0, 1)      -2.248539321109048  
(0, 2)      6.743901095206777  
(0, 3)      3.692522724816055  
(0, 4)     -0.9624702646110661  
(0, 6)     -2.8041134643109387  
(0, 8)      6.4521616074692005  
(0, 9)      0.5522472107515675
```

```
In [121]: model.n_iter_ # количество итераций
```

```
Out[121]: 21
```

```
In [125]: # создать виджет, но не отображать его
ip = widgets.interactive(
    linear_predictor,
    alpha=widgets.FloatSlider(min=0.0001, max=0.05, step=0.0001, value=0.001),
    model="Lasso"
);

# отображаем слайдер
display(widgets.HBox(ip.children[:1]))

# отображаем вывод функции
display(ip.children[-1])
ip.update() # чтобы функция запустилась до первого изменения слайдеров

HBox(children=(FloatSlider(value=0.001, description='alpha', max=0.05, min=0.0001, step=0.0001),))

Output()
```

## Elastic

```
In [126]: model = ElasticNet(
    alpha=0.01, # коэффициент регуляризации, чем больше - тем сильнее регуляризация
    l1_ratio=0.6, # какое предпочтение отдавать l1, а не l2
    fit_intercept=True, # использовать ли параметр-остаток при обучении
    max_iter=100, # максимальное количество итераций в методе оптимизации
    tol=0.0001, # точность решения
    precompute=True, # использовать ли предпосчитанную матрицу Грамма для ускорения расчетов
    warm_start=False, # использовать ли результаты последнего обучения для инициализации
    positive=False # если True, то будет подбирать положительные коэффициенты
)
```

```
In [127]: model.fit(X, y) # обучаем модель
```

```
Out[127]: ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.6,
    max_iter=100, normalize=False, positive=False, precompute=True,
    random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
In [128]: model.coef_ # коэффициенты
```

```
Out[128]: array([ 0.          , -0.          ,  2.55255863,  1.45418579,  0.          ,
                  0.          , -0.96292957,  0.9192122 ,  2.24834205,  0.7722448 ])
```

```
In [129]: model.sparse_coef_ # другое представление коэффициентов (без нулевых)
```

```
Out[129]: <1x10 sparse matrix of type '<class 'numpy.float64'>'
          with 6 stored elements in Compressed Sparse Row format>
```

```
In [130]: print(model.sparse_coef_)
```

```
(0, 2)      2.552558626790454
(0, 3)      1.4541857876976718
(0, 6)      -0.962929566706065
(0, 7)      0.9192122027002174
(0, 8)      2.2483420483553225
(0, 9)      0.7722448047128204
```

```
In [131]: model.n_iter_ # количество итераций
```

```
Out[131]: 7
```

```
In [132]: # создать виджет, но не отображать его
```

```
ip = widgets.interactive(
    linear_predictor,
    alpha=widgets.FloatSlider(min=0.0001, max=0.1, step=0.001, value=0.01),
    model="Elastic"
);
```

```
# отображаем слайдер
```

```
display(widgets.HBox(ip.children[:1]))
```

```
# отображаем вывод функции
```

```
display(ip.children[-1])
```

```
ip.update() # чтобы функция запустилась до первого изменения слайдеров
```

```
HBox(children=(FloatSlider(value=0.01, description='alpha', max=0.1, min=0.0001, step=0.001),))
```

```
Output()
```

# Сравнение трех моделей

Рассмотрим графики зависимости значений параметров моделей от параметра регуляризации.

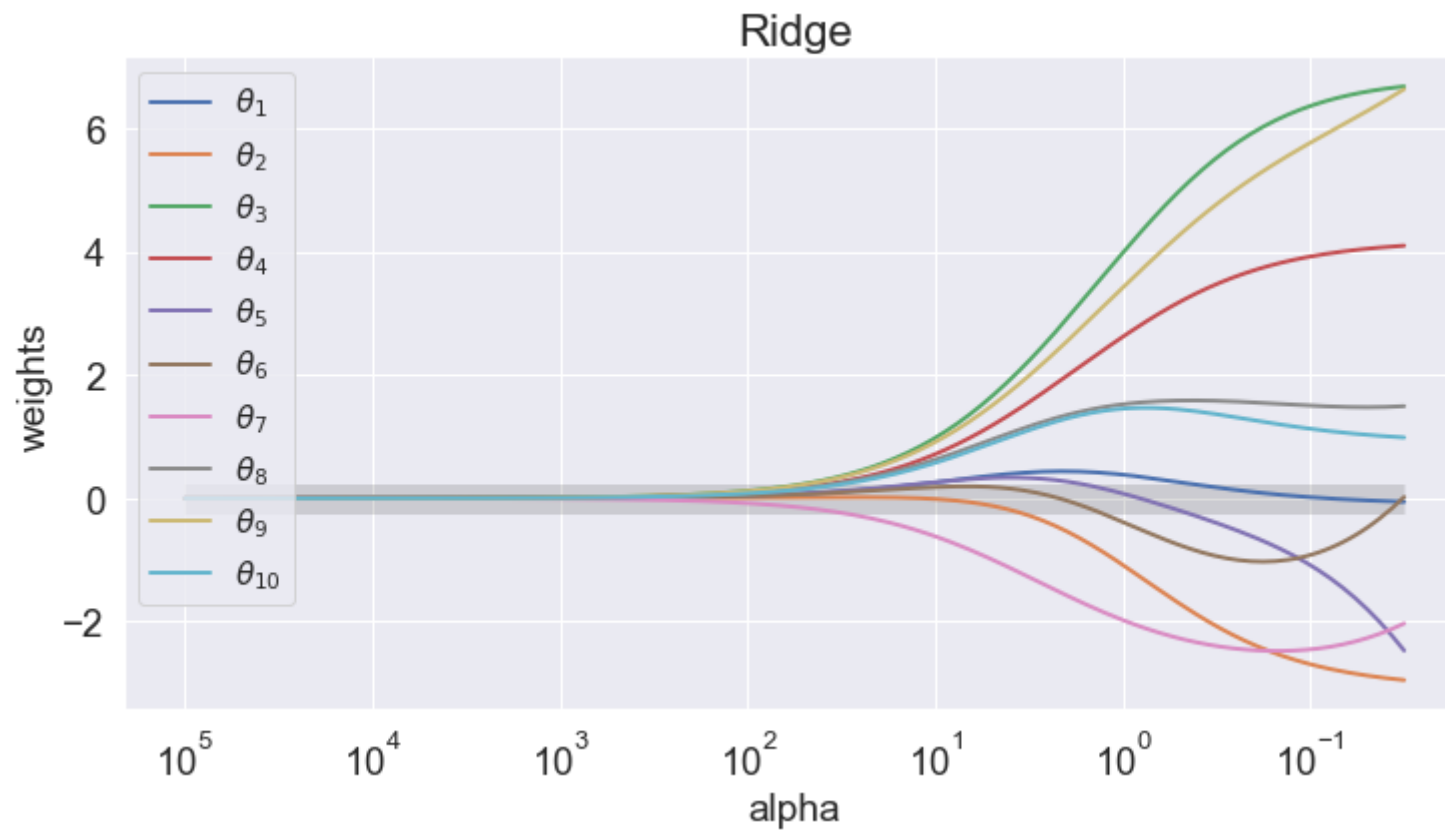
```
In [133]: def draw_track(model, X, y, log_min, log_max, num, title='', figsize=(12, 6)):
    """
    Данная функция строит график зависимости значений коэффициентов модели от параметра регуляризации
    """

    alphas = np.logspace(log_min, log_max, num)
    coefs = []
    for a in alphas:
        if 'l1_ratio' in model.get_params():
            model.set_params(alpha=a, l1_ratio=0.5)
        else:
            model.set_params(alpha=a)
        model.fit(X, y)
        coefs.append(model.coef_)

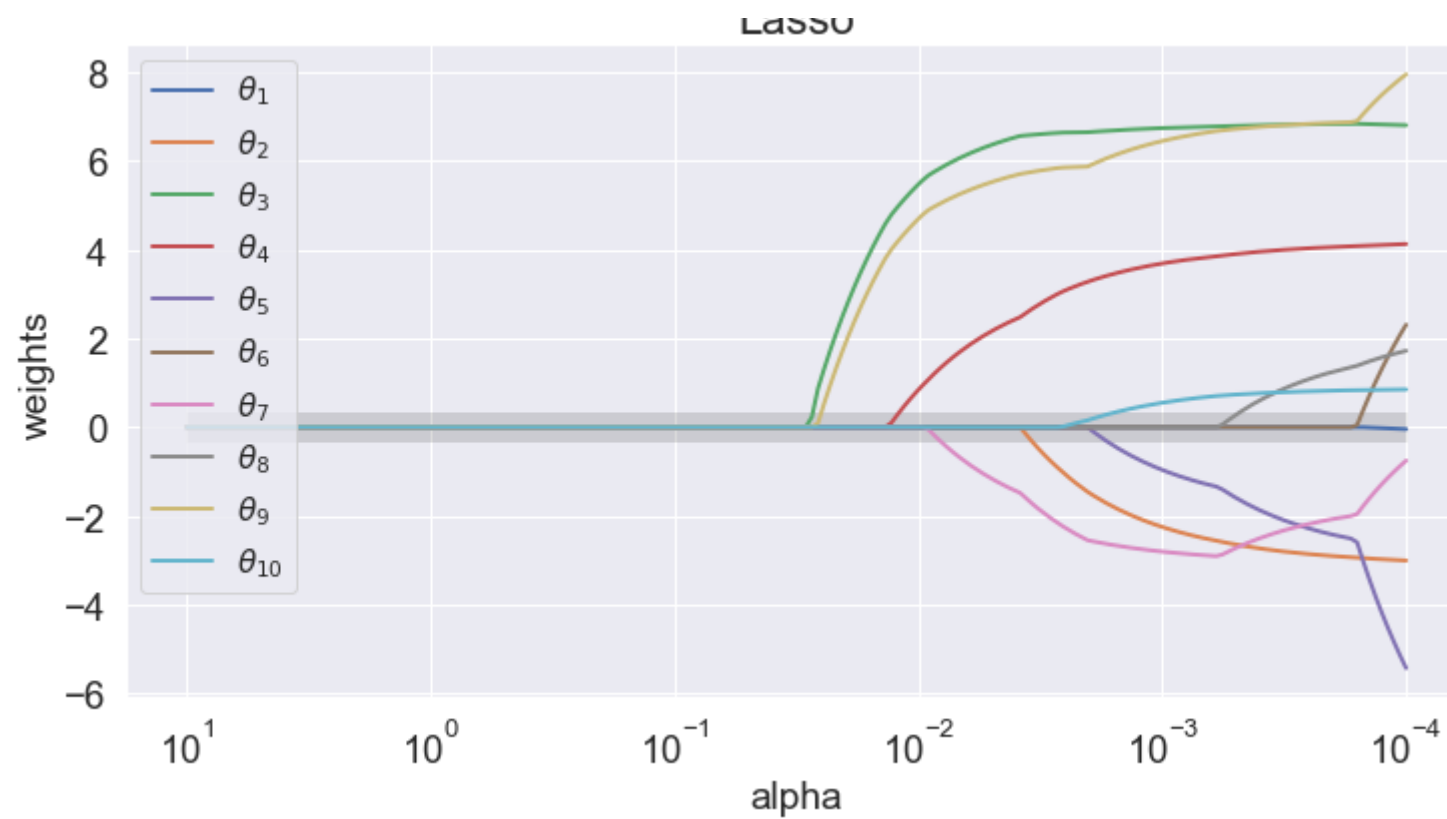
    plt.figure(figsize=figsize)
    ax = plt.gca()
    ax.hlines(0, 10 ** log_min, 10 ** log_max, linewidth=15, alpha=0.15)
    ind = 1
    for coef in np.array(coefs).T:
        label = r'$\theta_{' + str(ind) + '}$'
        ax.plot(alphas, coef, linewidth=2, label=label)
        ind += 1

    ax.set_xscale('log')
    ax.set_xlim(ax.get_xlim()[::-1]) # reverse axis
    plt.xlabel('alpha', fontsize=19)
    plt.ylabel('weights', fontsize=19)
    plt.title(title, fontsize=22)
    plt.legend(loc='upper left', fontsize=15)
    plt.axis('tight')
    plt.show()
```

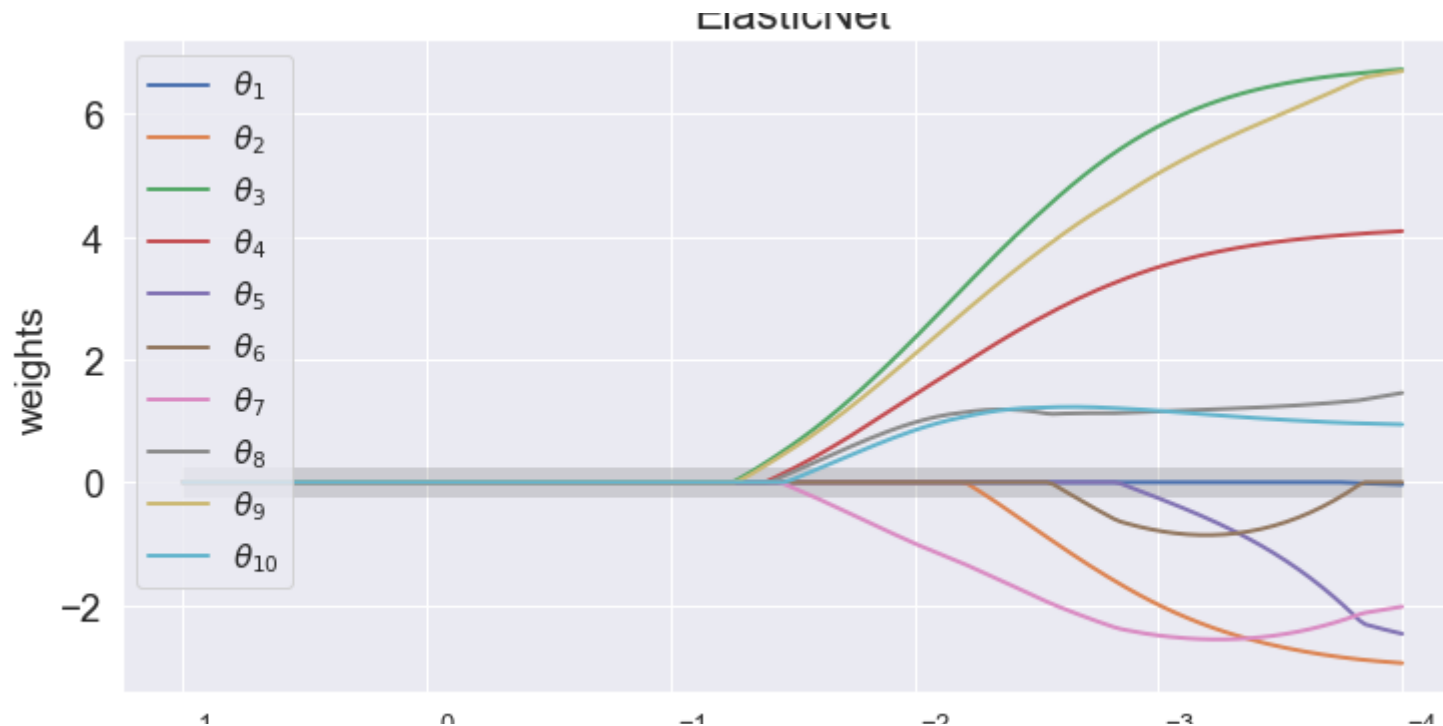
```
In [135]: draw_track(Ridge(fit_intercept=False), X, y, -1.5, 5, 200, title='Ridge')
draw_track(Lasso(fit_intercept=False), X, y, -4, 1, 200, title='Lasso')
draw_track(ElasticNet(fit_intercept=False), X, y, -4, 1, 200, title='ElasticNet')
```



Lasso



**ElasticNet**



Из графиков выше можно сделать вывод, что при высоких значениях коэффициента регуляризации модели не учатся и все коэффициенты зануляются. Также важно заметить, что Lasso при высоких значениях коэффициента регуляризации склонна занулять параметры, но в отличие от других моделей она **отбирает** определенные признаки начиная с некоторого значения коэффициента регуляризации. Другими словами, с ослаблением регуляризации Lasso присваивает ненулевые значения лишь каким-то конкретным признакам, которые считает наиболее важными.

## Вероятностный и оптимизационный подходы

На занятии вы узнали, что регрессионную модель можно рассматривать как в вероятностном, так и в оптимизационном подходе. Для вероятностного подхода при этом должны выполняться вероятностные свойства. Давайте повторим теорию про оба подхода для Ridge-регрессии:

	Вероятностный подход	Оптимизационный подход
Зависимость	$y = x^T \theta$	$y = x^T \theta$
Задача	$y = X\theta + \epsilon,$ $\epsilon \sim \mathcal{N}(0, \beta^{-1} I_n),$ $\theta \sim \mathcal{N}(0, \alpha^{-1} I_d),$ $\theta Y - ?$	$\ Y - X\theta\ _2^2 + \lambda \cdot \ \theta\ _2 \rightarrow \min$
Требования	Гауссовская линейная модель	Отсутствуют
Оценка	$\theta Y \sim \mathcal{N}(\beta \cdot \Sigma \cdot X^T Y, \Sigma),$ где $\Sigma = (\beta X^T X + \alpha I_d)^{-1},$ $\hat{\theta} = (X^T X + \frac{\alpha}{\beta} I_d)^{-1} X^T Y$	$\hat{\theta} = (X^T X + \lambda I_d)^{-1} X^T Y$
Свойства	1) Решение всегда существует 2) Так как знаем распределение, то можем строить доверительные и предсказательные интервалы 3) Так знаем распределение, то можем проверять гипотезы	1) Решение всегда существует, если $\lambda$ не равно 0

Таким образом, если данные обладают необходимыми свойствами, то мы можем построить доверительные интервалы для предсказаний модели. Вспомним, что линейную модель называют гауссовской, если  $\epsilon \sim \mathcal{N}(0, \sigma^2 I_n)$ , где  $\epsilon$  - остатки модели.

Рассмотрим данные о квартирах в Москве.

```
In [136]: flats_data = pd.read_csv('flats_moscow.txt', delimiter='\t', index_col='n')
```



```
In [137]: flats_data.head()
```

```
Out[137]:
```

	price	totsp	livesp	kitsp	dist	metrdist	walk	brick	floor	code
n										
1	81	58	40	6.0	12.5	7	1	1	1	3
2	75	44	28	6.0	13.5	7	1	0	1	6
3	128	70	42	6.0	14.5	3	1	1	1	3
4	95	61	37	6.0	13.5	7	1	0	1	1
5	330	104	60	11.0	10.5	7	0	1	1	3

Мы будем использовать признаки `price` - цена квартиры и `totsp` - площадь квартиры.

```
In [138]: plt.figure(figsize=(12, 6))
plt.scatter(flats_data['totsp'], flats_data['price'], alpha=0.25)
plt.title('Зависимость цены квартир в Москве от площади')
plt.xlabel('Площадь квартиры')
plt.ylabel('Цена квартиры')
plt.show()
```



На графике явно видна гетероскедстичность.

```
In [151]: x = flats_data[['totsp']]  
y = flats_data['price']
```

Центрируем данные.

```
In [152]: x = (x - x.mean()) / x.std()  
y = (y - y.mean()) / y.std()
```

Обучим модель и сделаем предсказание.

```
In [153]: reg_coef = 0.1  
model = Ridge(alpha=reg_coef)  
model.fit(x, y)  
y_pred = model.predict(x)
```

```
In [154]: n = x.shape[0] # количество объектов  
d = x.shape[1] # количество признаков
```

Для оценки параметра  $1/\beta$  используем оценку  $\frac{\|Y - X\theta\|^2}{n - d}$

```
In [155]: b = (n - d) / np.sum((y_pred - y) ** 2)  
a = reg_coef * b  
sigma = 1 / (b * x.T @ x + a)
```

Построим предиктивные интервалы для отклика. Для этого вспомним теорию с лекции:  
в линейной гауссовской модели  $\epsilon \sim \mathcal{N}(0, \beta^{-1} I_n)$ . Пусть  $x_0$  - новый объект.  $y_0 = x_0^T \theta + \epsilon_0$  - наблюдаемый отклик на  $x_0$ . Тогда  $y_0 | Y \sim \mathcal{N}(\beta x_0^T X^T Y, x_0^T \Sigma x_0 + \beta^{-1})$ .  
Квантили данного распределения задают границы предиктивного интервала.

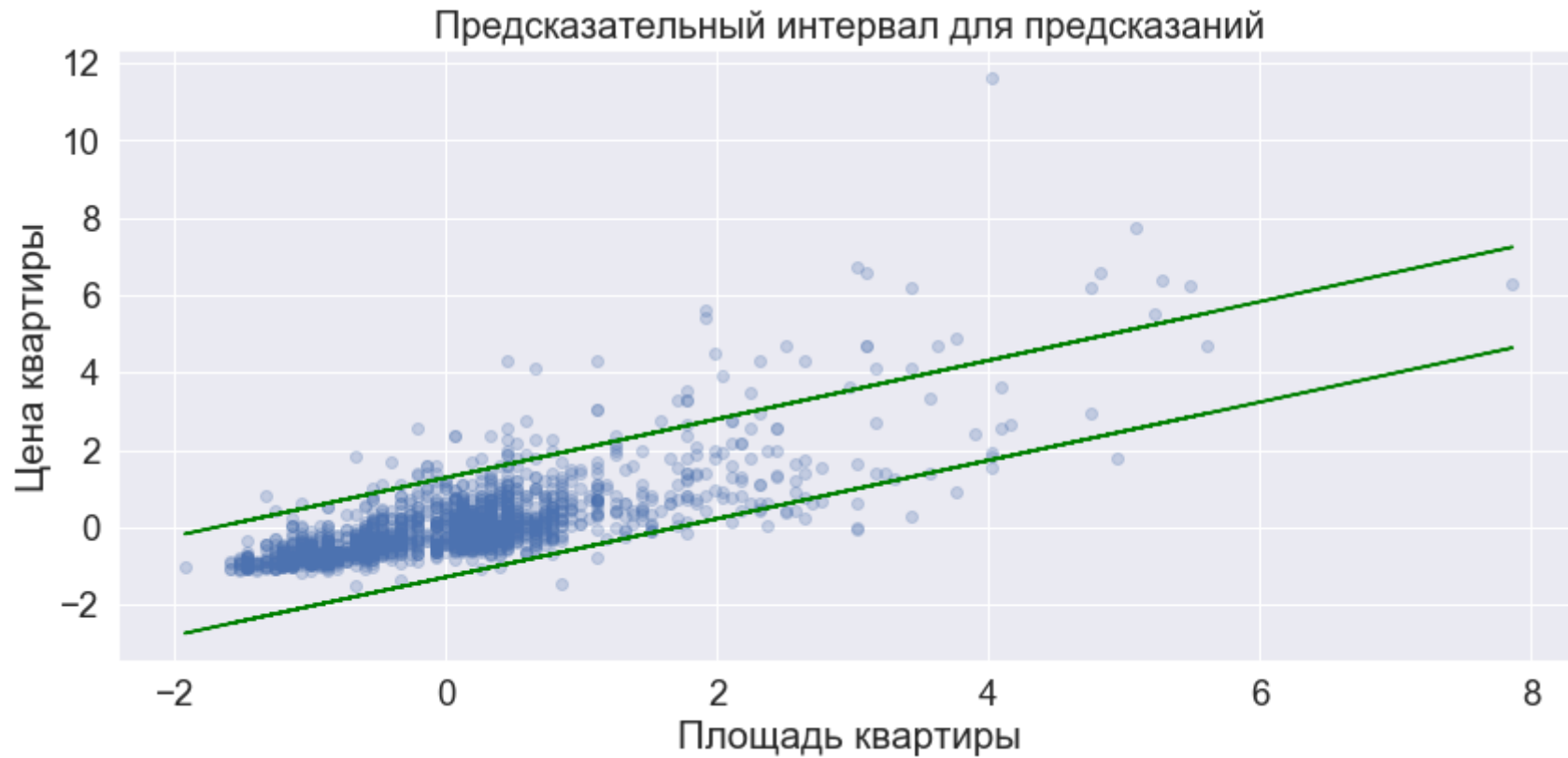
```
In [156]: left = []
          right = []
          estimations = []

          for x0 in X.values:
              x0 = x0.reshape(-1, 1)
              y0 = model.predict(x0)
              estimations.append(y0)

              distr = norm(b * x0.T @ sigma @ X.T @ y, np.sqrt(x0.T @ sigma * x0 + 1 / b))
              left.append(distr.ppf(0.975).ravel())
              right.append(distr.ppf(0.025).ravel())
```

Посмотрим график предсказаний и предиктивный интервал.

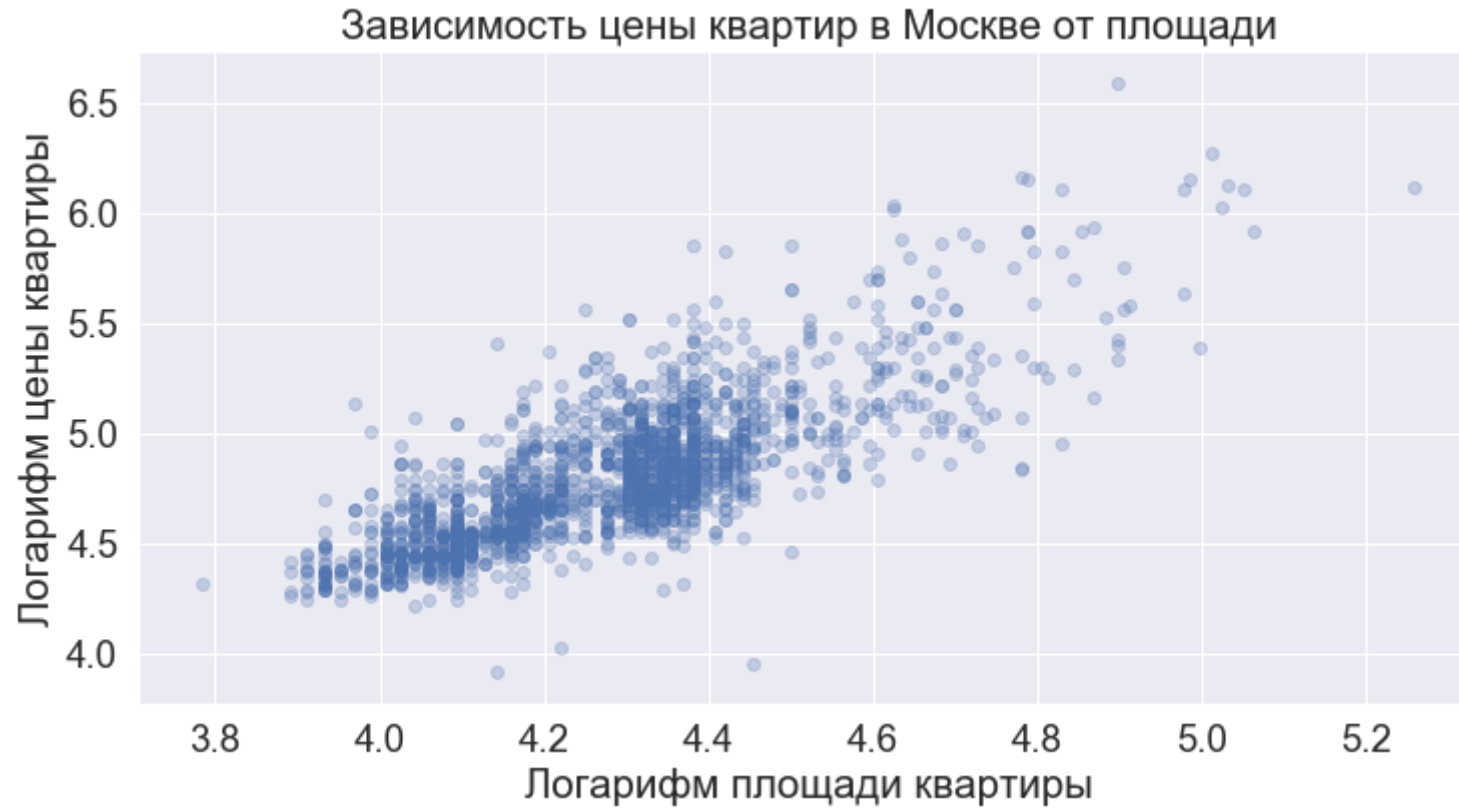
```
In [157]: plt.figure(figsize=(14,6))
plt.scatter(X['totsp'], y, alpha=0.25)
plt.plot(X['totsp'], left, color='green')
plt.plot(X['totsp'], right, color='green')
plt.title('Предсказательный интервал для предсказаний', fontsize=20)
plt.xlabel('Площадь квартиры')
plt.ylabel('Цена квартиры')
plt.show()
```



Видим, что теоретические свойства не выполнены - слишком много объектов не попадают в доверительный интервал. Это произошло из-за того, что ошибки модели гетероскедастичны.

Для решения данной проблемы попробуем посмотреть зависимость логарифма таргета от логарифма признака.

```
In [158]: plt.figure(figsize=(12, 6))
plt.scatter(np.log(flats_data['totsp']), np.log(flats_data['price']), alpha=0.25)
plt.title('Зависимость цены квартир в Москве от площади')
plt.xlabel('Логарифм площади квартиры')
plt.ylabel('Логарифм цены квартиры')
plt.show()
```



Задаем новую выборку и новый таргет - берем их логарифмы.

```
In [161]: X = np.log(flats_data[['totsp']])  
y = np.log(flats_data['price'])
```

Центрируем данные.

```
In [162]: X = (X - X.mean()) / X.std()  
y = (y - y.mean()) / y.std()
```

Обучаем модель.

```
In [163]: reg_coef = 0.1  
model = Ridge(alpha=reg_coef)  
model.fit(X, y)  
y_pred = model.predict(X)
```

Оцениваем параметры гауссовской модели.

```
In [164]: b = (n - d) / np.sum((y_pred - y) ** 2)  
a = reg_coef * b  
sigma = 1 / (b * X.T @ X + a)
```



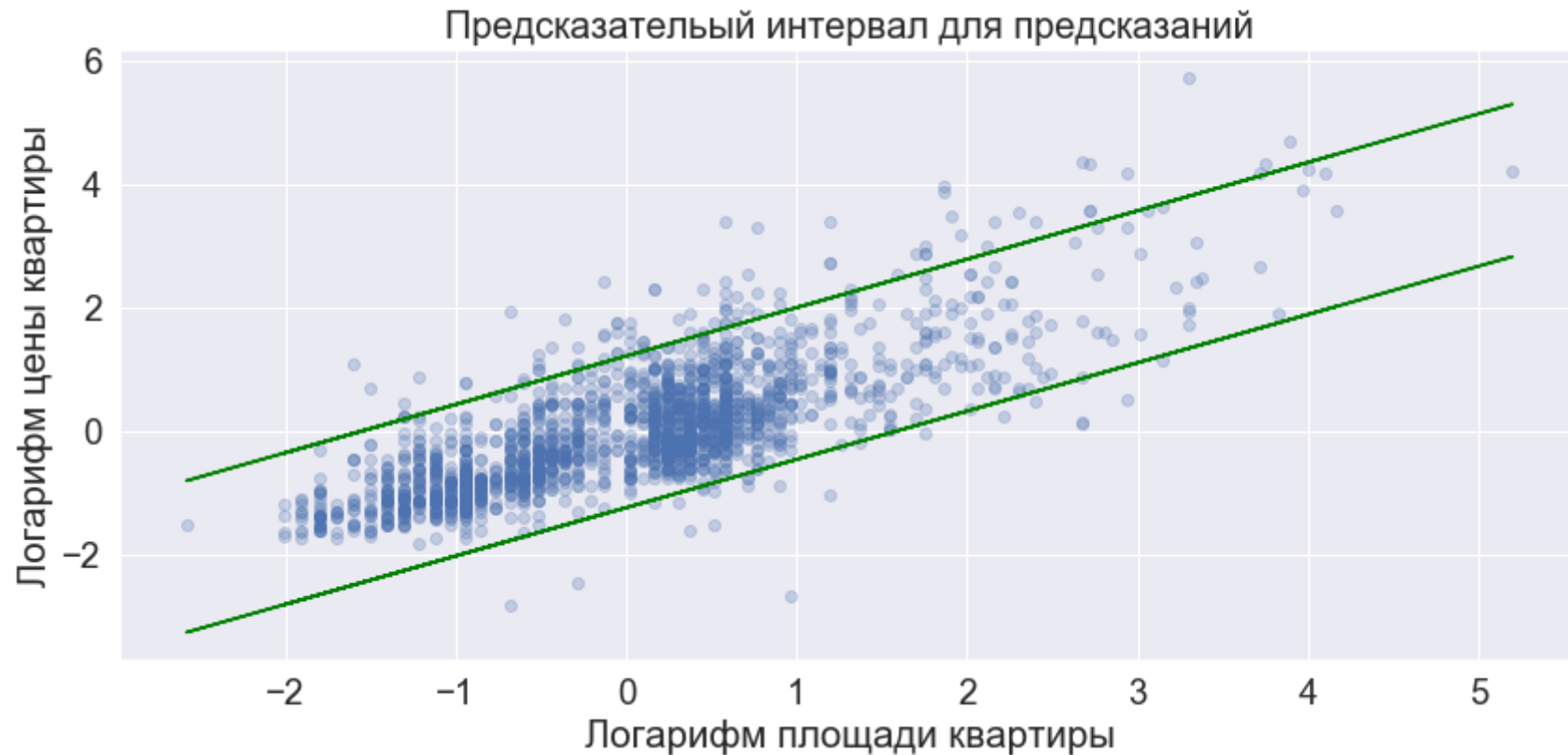
Считаем предиктивный интервал.

```
In [165]: left = []
right = []
estimations = []

for x0 in X.values:
    x0 = x0.reshape(-1, 1)
    y0 = model.predict(x0)
    estimations.append(y0)

    distr = norm(b * x0.T @ sigma @ X.T @ y, np.sqrt(x0.T @ sigma * x0 + 1 / b))
    left.append(distr.ppf(0.975).ravel())
    right.append(distr.ppf(0.025).ravel())
```

```
In [166]: plt.figure(figsize=(14,6))
plt.scatter(X['totsp'], y, alpha=0.25)
plt.plot(X['totsp'], left, color='green')
plt.plot(X['totsp'], right, color='green')
plt.title('Предсказательный интервал для предсказаний', fontsize=20)
plt.xlabel('Логарифм площади квартиры')
plt.ylabel('Логарифм цены квартиры')
plt.show()
```



Видим, что ситуация улучшилась. Таким образом, если данные гетероскедастичны, то можно попробовать рассмотреть логарифмы признаков или таргета, это может улучшить предсказания.

Теперь рассмотрим датасет о потреблении мороженого.

```
In [167]: ice_data = pd.read_csv('ice_cream.txt', delimiter='\t', index_col='date')
```

```
In [168]: ice_data.head()
```

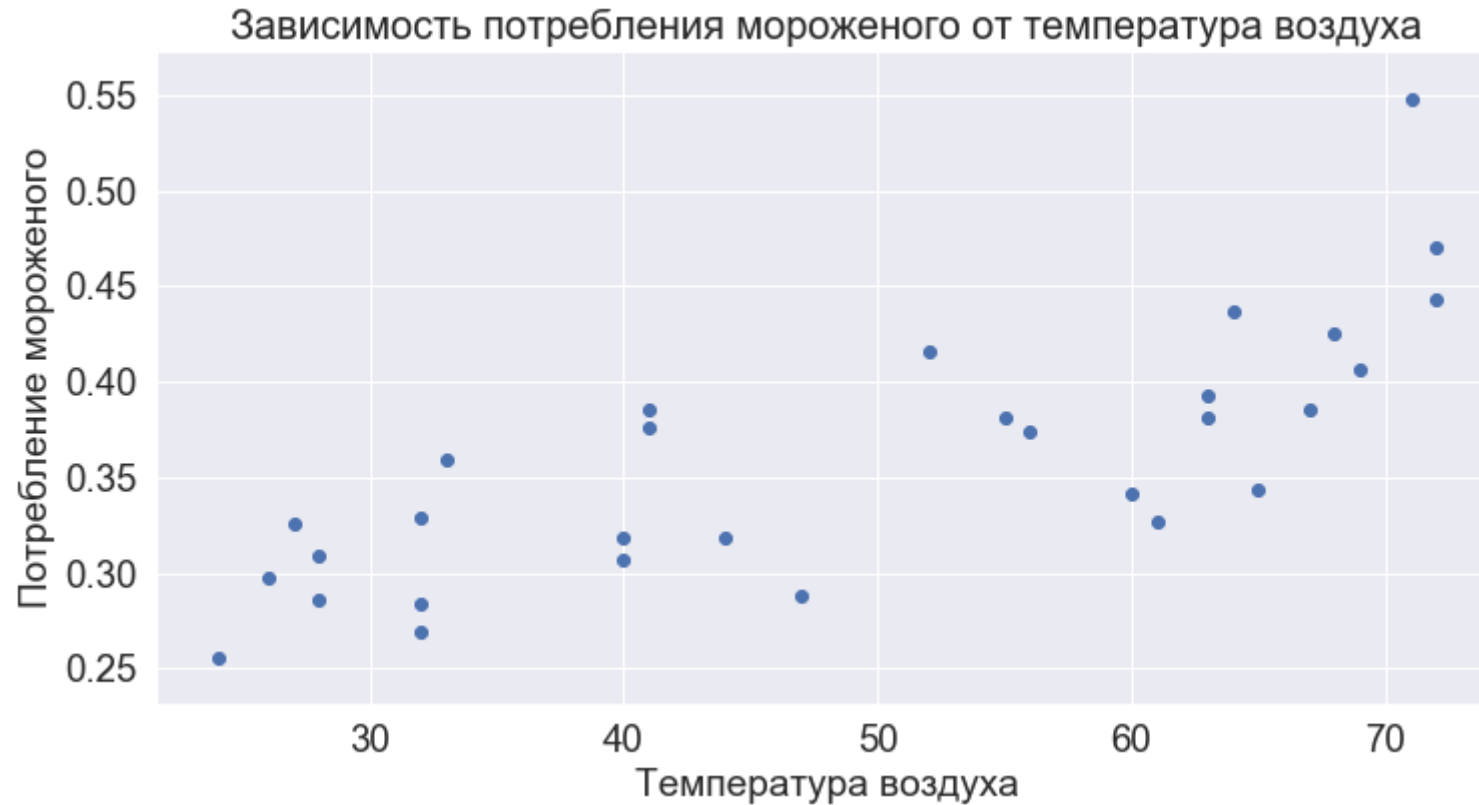
Out[168]:

	IC	price	income	temp	Lag-temp	Year
date						
1	0.386	0.270	78	41	56	0
2	0.374	0.282	79	56	63	0
3	0.393	0.277	81	63	68	0
4	0.425	0.280	80	68	69	0
5	0.406	0.272	76	69	65	0

Мы будем использовать признаки IC - потребление мороженого на человека, temp - температура воздуха в этот день.

Посмотрим, как зависит целевая переменная IC от temp .

```
In [169]: plt.figure(figsize=(12, 6))
plt.scatter(ice_data['temp'], ice_data['IC'])
plt.title('Зависимость потребления мороженого от температура воздуха', fontsize=20)
plt.xlabel('Температура воздуха', fontsize=19)
plt.ylabel('Потребление мороженого', fontsize=19)
plt.show()
```



Из графика видна гомоскедастичность.

```
In [170]: x = ice_data[['temp']]  
y = ice_data['IC']
```

Центрируем данные.

```
In [171]: x = (x - x.mean()) / x.std()  
y = (y - y.mean()) / y.std()
```

Обучаем модель и делаем предсказания.

```
In [172]: reg_coef = 0.01  
model = Ridge(alpha=reg_coef)  
model.fit(x, y)  
y_pred = model.predict(x)
```

```
In [173]: n = x.shape[0] # количество объектов  
d = x.shape[1] # количество признаков
```

Оцениваем параметры гауссовской модели.

```
In [174]: b = (n - d) / np.sum((y_pred - y) ** 2)  
a = reg_coef * b  
sigma = 1 / (b * x.T @ x + a)
```

Считаем предиктивный интервал.

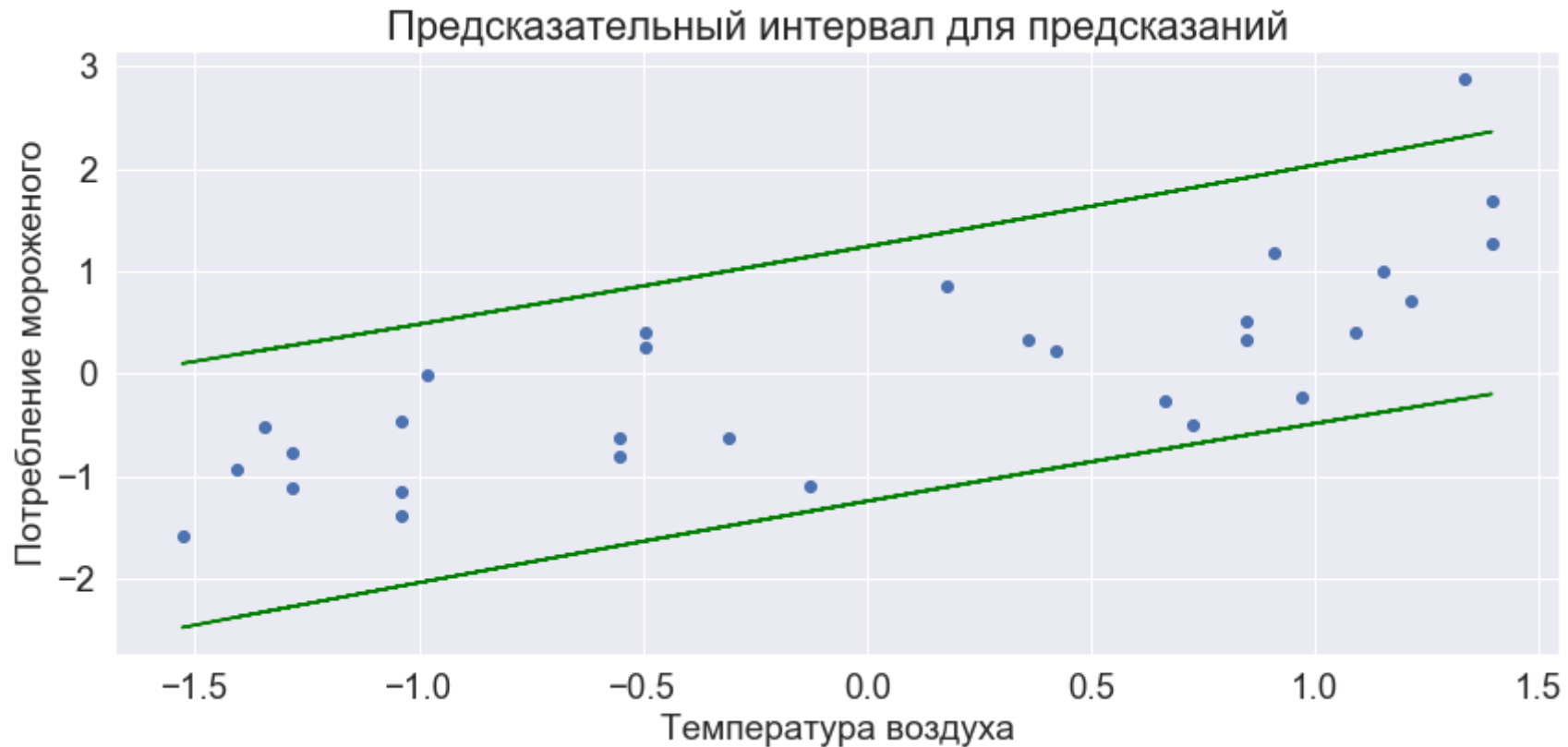
```
In [175]: left = []
right = []
estimations = []

for x0 in X.values:
    x0 = x0.reshape(-1, 1)
    y0 = model.predict(x0)
    estimations.append(y0)

    distr = norm(b * x0.T @ sigma @ X.T @ y, np.sqrt(x0.T @ sigma * x0 + 1 / b))
    left.append(distr.ppf(0.975).ravel())
    right.append(distr.ppf(0.025).ravel())
```

Строим график зависимости таргета от признака и предиктивный интервал.

```
In [176]: plt.figure(figsize=(14,6))
plt.scatter(X['temp'], y)
plt.plot(X['temp'], left, color='green')
plt.plot(X['temp'], right, color='green')
plt.title('Предсказательный интервал для предсказаний', fontsize=22)
plt.xlabel('Температура воздуха', fontsize=19)
plt.ylabel('Потребление мороженого', fontsize=19)
plt.show()
```



Видим, что почти все предсказания попали в доверительный интервал.

Таким образом, мы рассмотрели теоретические свойства для гауссовской модели и проверили их на практике в виде реализации предиктивных интервалов. Полезные следствия из гауссовской модели не работают на практике, если данные не удовлетворяют необходимым требованиям вероятностного подхода. В нашем случае, на гетероскедастичных данных не удалось построить предиктивный

интервал с заявленными теоретическими свойствами. Как вариант, улучшить ситуацию с гетероскедастичностью можно при помощи экспериментов с логарифмами признаков.