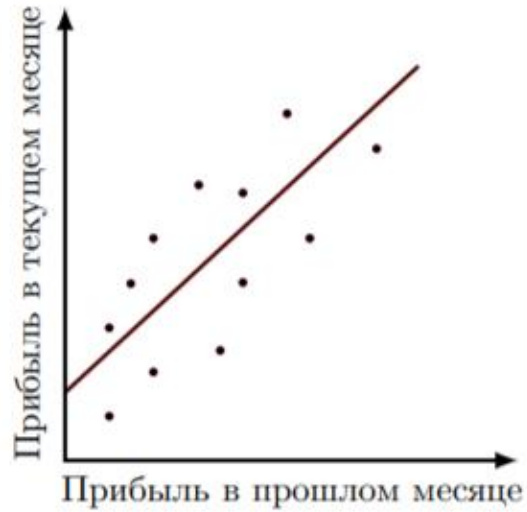
Задача регрессии

- $\mathbb{X}$ — пространство объектов
- $\mathbb{Y}$ — пространство ответов
- $x = (x^1, ..., x^d)$ — признаковое описание объекта
- $X = (x_i, y_i)_{i=1}^{\ell}$ — обучающая выборка
- $a(x)$ — алгоритм, модель
- $Q(a, X)$ — функционал ошибки алгоритма a на выборке X
- Обучение: $a(x) = \operatorname{argmin}_{a \in \mathbb{A}} Q(a, X)$

Напомним, что в задаче регрессии пространство ответов $\mathbb{Y} = \mathbb{R}$. Чтобы научиться решать задачу регрессии, необходимо задать:

- **Функционал ошибки** Q: способ измерения того, хорошо или плохо работает алгоритм на конкретной выборке.
- **Семейство алгоритмов** $\mathbb{A}$: как выглядит множество алгоритмов, из которых выбирается лучший.
- **Метод обучения**: как именно выбирается лучший алгоритм из семейства алгоритмов.

Пусть известен один признак — прибыль магазина в прошлом месяце, а предсказать необходимо прибыль магазина в следующем. Поскольку прибыль — вещественная переменная, здесь идет речь о задаче регрессии.



По этому графику можно сделать вывод о существовании зависимости между прибылью в следующем и прошлом месяцах. Если предположить, что зависимость приблизительно линейная, ее можно представить в виде прямой на этом графике. По этой прямой и можно будет предсказывать прибыль в следующем месяце, если известна прибыль в прошлом.

В целом такая модель угадывает тенденцию, то есть описывает зависимость между ответом и признаком. При этом, разумеется, она делает это не идеально, с некоторой ошибкой. Истинный ответ на каждом объекте несколько отклоняется от прогноза.

Один признак — это не очень серьезно. Гораздо сложнее и интереснее работать с многомерными выборками, которые описываются большим количеством признаков. В этом случае нарисовать выборку и понять, подходит или нет линейная модель, нельзя. Можно лишь оценить ее качество и по нему уже понять, подходит ли эта модель.

Следует отметить, что вообще нельзя придумать модель, которая идеально описывает ваши данные, то есть идеально описывает, как порождается ответ по признакам.

Далее обсудим, как выглядит семейство алгоритмов в случае с линейными моделями. Линейный алгоритм в задачах регрессии выглядит следующим образом:

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x^j,$$

где w_0 — свободный коэффициент, x^j — признаки, а w_j — их веса.

Если добавить $(d+1)$-й признак, который на каждом объекте принимает значение 1, линейный алгоритм можно будет записать в более компактной форме

$$a(x) = \sum_{j=1}^{d+1} w_j x^j = \langle w, x \rangle,$$

где используется обозначение $\langle w, x \rangle$ для скалярного произведения двух векторов.

В качестве меры ошибки не может быть выбрано отклонение от прогноза $Q(a, y) = a(x) - y$, так как в этом случае минимум функционала не будет достигаться при правильном ответе $a(x) = y$. Самый простой способ — считать модуль отклонения:

$$|a(x) - y|.$$

Но функция модуля не является гладкой функцией, и для оптимизации такого функционала неудобно использовать градиентные методы. Поэтому в качестве меры ошибки часто выбирается квадрат отклонения:

$$(a(x) - y)^2.$$

Функционал ошибки, именуемый среднеквадратичной ошибкой алгоритма, задается следующим образом:

$$Q(a, x) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2$$

В случае линейной модели его можно переписать в виде функции (поскольку теперь Q зависит от вектора, а не от функции) ошибок:

$$Q(w, x) = \frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2 .$$

Обучение модели линейной регрессии $$Q(w,x) = \frac{1}{\ell}\sum_{i=1}^{\ell}(\langle w, x_i\rangle - y_i)^2 \to \min_w.$$

Следует напомнить, что в число признаков входит также постоянный признак, равный 1 для всех объектов, что позволяет исключить постоянную составляющую в последнем соотношении.

Прежде, чем будет рассмотрена задача оптимизации этой функции, имеет смысл переписать используемые соотношения в матричной форме. Матрица «объекты–признаки» X составлена из признаковых описаний всех объектов из обучающей выборки:

$$X = \begin{pmatrix} x_{11} & \dots & x_{1d} \\ \dots & \dots & \dots \\ x_{\ell 1} & \dots & x_{\ell d} \end{pmatrix}$$

Таким образом, в ij элементе матрицы X записано значение j-го признака на i объекте обучающей выборки. Также понадобится вектор ответов y, который составлен из истинных ответов для всех объектов:

$$y = \begin{pmatrix} y_1 \\ \dots \\ y_\ell \end{pmatrix}.$$

В этом случае среднеквадратичная ошибка может быть переписана в матричном виде:

$$Q(w,X) = \frac{1}{\ell}\|Xw - y\|^2 \to \min_w.$$

Можно найти аналитическое решение задачи минимизации:

$$w_* = (X^T X)^{-1} X^T y.$$

Основные сложности при нахождении решения таким способом:

- Для нахождения решения необходимо вычислять обратную матрицу. Операция обращения матрицы требует, в случае d признаков, выполнение порядка d^3 операции, и является вычислительно сложной уже в задачах с десятком признаков.
- Численный способ нахождения обратной матрицы не может быть применен в некоторых случаях (когда матрица плохо обусловлена).

Другой, несколько более удобный, способ найти решение — использовать численные методы оптимизации.

Несложно показать, что среднеквадратическая ошибка — это выпуклая и гладкая функция. Выпуклость гарантирует существование лишь одного минимума, а гладкость — существование вектора градиента в каждой точке. Это позволяет использовать метод градиентного спуска.

При использовании метода градиентного спуска необходимо указать начальное приближение. Есть много подходов к тому, как это сделать, в том числе инициализировать случайными числами (не очень большими). Самый простой способ это сделать — инициализировать значения всех весов равными нулю:

$$w^0 = 0.$$

На каждой следующей итерации, $t = 1, 2, 3, ...$, из приближения, полученного в предыдущей итерации w^{t-1}, вычитается вектор градиента в соответствующей точке w^{t-1}, умноженный на некоторый коэффициент η_t, называемый шагом:

$$w^t = w^{t-1} - \eta_t \nabla Q(w^{t-1}, X).$$

Остановить итерации следует, когда наступает сходимость. Сходимость можно определять по-разному. В данном случае разумно определить сходимость следующим образом: итерации следует завершить, если разница двух последовательных приближений не слишком велика:

$$\|w^t - w^{t-1}\| < \varepsilon.$$

Метрики качества в задачах регрессии

Первая метрика, о которой уже шла речь — среднеквадратичная ошибка:

$$MSE(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2 .$$

Такой функционал легко оптимизировать, используя, например, метод градиентного спуска.

Этот функционал сильно штрафует за большие ошибки, так как отклонения возводятся в квадрат. Это приводит к тому, что штраф на выбросе будет очень сильным, и алгоритм будет настраиваться на выбросы. Другими словами, алгоритм будет настраиваться на такие объекты, на которые не имеет смысл настраиваться.

Похожий на предыдущий функционал качества — средняя абсолютная ошибка:

$$MAE(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} |a(x_i) - y_i| .$$

Этот функционал сложнее минимизировать, так как у модуля производная не существует в нуле. Но у такого функционала больше устойчивость к выбросам, так как штраф за сильное отклонение гораздо меньше.

Коэффициент детерминации $R^2(a, X)$:

$$R^2(a, X) = 1 - \frac{\sum_{i=1}^{\ell} \big(a(x_i) - y_i\big)^2}{\sum_{i=1}^{\ell} (y_i - \bar{y})}, \qquad \bar{y} = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i,$$

позволяет интерпретировать значение среднеквадратичной ошибки. Этот коэффициент показывает, какую долю дисперсии (разнообразия ответов) во всем целевом векторе y модель смогла объяснить.

Для разумных моделей коэффициент детерминации лежит в следующих пределах:

$$0 \leq R^2 \leq 1,$$

причем случай $R^2 = 1$ соответствует случаю идеальной модели, $R^2 = 0$ — модели на уровне оптимальной «константной», а $R^2 < 0$ — модели хуже «константной» (такие алгоритмы никогда не нужно рассматривать). Оптимальным константым алгоритмом называется такой алгоритм, который возвращает всегда среднее значение ответов $\bar{y}$ для объектов обучающей выборки.

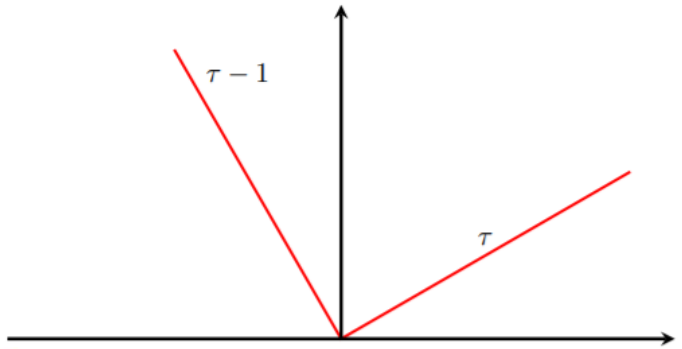
До этого рассматривались симметричные модели, то есть такие, которые штрафуют как за недопрогноз, так и за перепрогноз. Но существуют такие задачи, в которых эти ошибки имеют разную цену.

Пусть, например, требуется оценить спрос на ноутбуки. В этом случае заниженный прогноз приведет к потере лояльности покупателей и потенциальной прибыли (будет закуплено недостаточное количество ноутбуков), а завышенный — только к не очень большим дополнительным расходам на хранение непроданных ноутбуков. Чтобы учесть это, функция потерь должна быть несимметричной и сильнее штрафовать за недопрогноз, чем за перепрогноз.

В таких случаях хорошо подходит квантильная ошибка или квантильная функция потерь:

$$\rho_\tau(a, X) = \frac{1}{\ell}\sum_{i=1}^{\ell}\Big((\tau-1)\big[y_i < a(x_i)\big] + \tau\big[y_i \geq a(x_i)\big]\Big)\,(y_i - a(x_i))\,.$$

Параметр $\tau \in [0, 1]$ определяет то, за что нужно штрафовать сильнее — за недопрогноз или перепрогноз. Если τ ближе к 1, штраф будет больше за недопрогноз, а если, наоборот, ближе к 0 — за перепрогноз.



Чтобы разобраться, почему такая функция потерь называется квантильной, нужно разобраться с ее вероятностным смыслом. Пусть один и тот же объект x с одним и тем же признаковым описанием повторяется в выборке n раз, но на каждом из повторов — свой ответ $y_1, ..., y_n$.

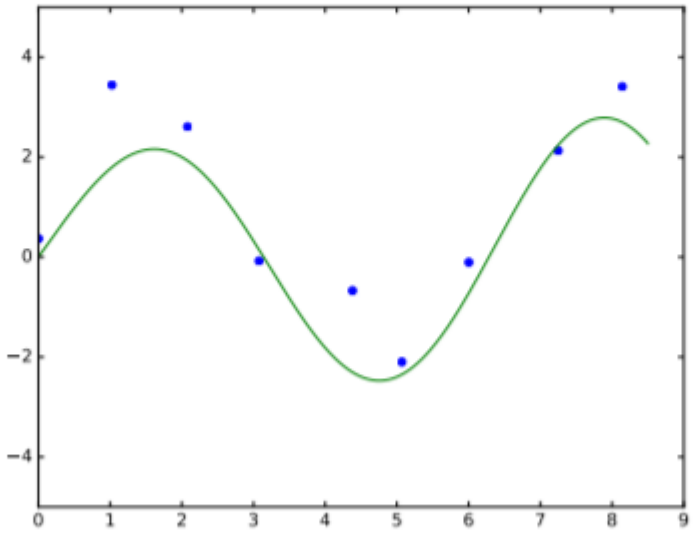
Такое может возникнуть при измерении роста человека. Измерения роста одного и того же человека могут отличаться ввиду ошибки прибора, а также зависеть от самого человека (может сгорбиться или выпрямиться).

При этом алгоритм должен для одного и того же признакового описания возвращать одинаковый прогноз. Другими словами, необходимо решить, какой прогноз оптимален для x с точки зрения различных функционалов ошибки.

Оказывается, что если используется квадратичный функционал ошибки, то наиболее оптимальным прогнозом будет средний ответ на объектах, если абсолютный, то медиана ответов. Если же будет использоваться квантильная функция потерь, наиболее оптимальным прогнозом, будет τ-квантиль. В этом и состоит вероятностный смысл квантильной ошибки.

Проблемы переобучения и недообучения в задачах линейной регрессии

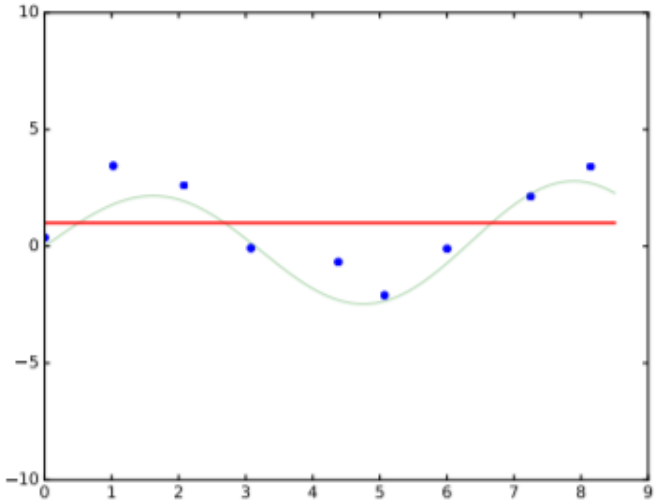
Глубже понять проблему переобучения можно на данном примере. На следующем графике изображена истинная зависимость и объекты обучающей выборки:



Истинная зависимость (зеленая линия) и элементы обучающей выборки (изображены синими точками)

Видно, что истинная зависимость является нелинейной и имеет два экстремума.

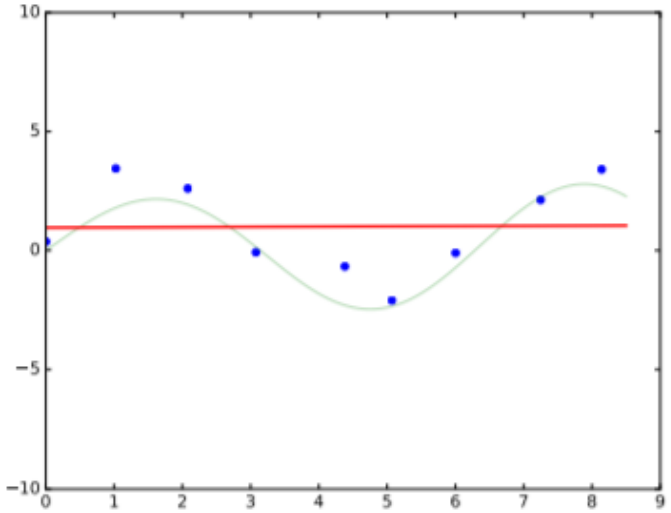
В модели $a(x) = w_0$, после того, как она будет настроена под данные, на графике получается некоторая горизотальная кривая, которая довольно плохо обобщает информацию об объектах из выборки.



Модель $a(x) = w_0$.

Имеет место недообучение. Хороший алгоритм не был построен, поскольку семейство алгоритмов слишком мало и с его помощью невозможно уловить закономерность.

В линейной регрессии используется семейство алгоритмов $a(x) = w_0 + w_1 x$.



Модель $a(x) = w_0 + w_1 x$.

В этом случае также будет иметь место недообучение. Получилось лучше, но прямая тоже плохо описывает данные.

Если семейство алгоритмов — множество многочленов 4-ей степени:

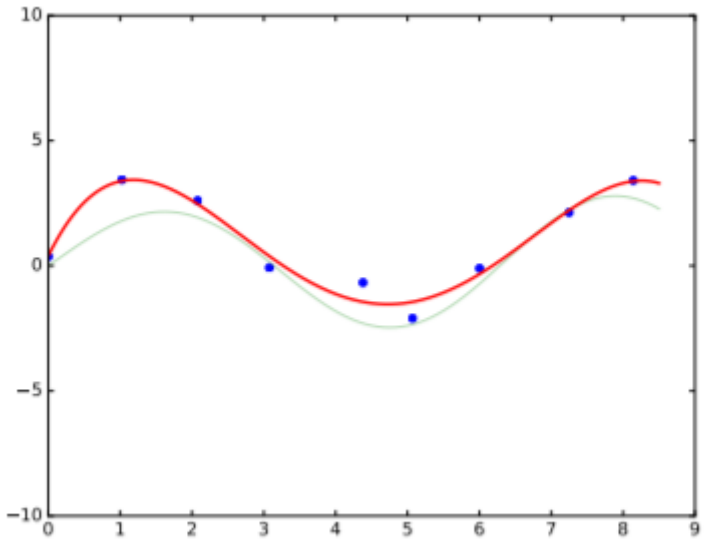
$$a(x) = w_0 + w_1 x + w_2 x^2 + ... + w_4 x^4,$$

то после обучения получившаяся кривая будет достаточно хорошо описывать и обучающую выборку, и истинную зависимость.

Если семейство алгоритмов — множество многочленов 4-ей степени:

$$a(x) = w_0 + w_1 x + w_2 x^2 + ... + w_4 x^4,$$

то после обучения получившаяся кривая будет достаточно хорошо описывать и обучающую выборку, и истинную зависимость.



Модель $a(x) = w_0 + w_1 x + w_2 x^2 + ... + w_4 x^4$.

В таком случае качество алгоритма хорошее, но нет идеального совпадения. Встает вопрос, а можно ли добиться совпадения увеличением сложности алгоритма.

При использовании многочленов 9-ой степени уже имеет место переобучение.

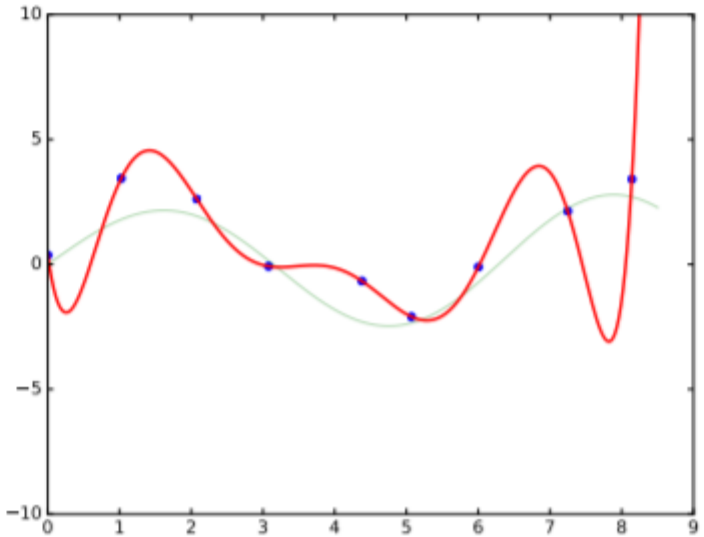


Рис. 3.5: Модель $a(x) = w_0 + w_1 x + w_2 x^2 + ... + w_9 x^9$.

Восстановленная зависимость дает идеальные ответы на всех объектах обучающей выборки, но при этом в любой другой точке сильно отличается от истинной зависимости. Такая ситуация называется переобучением. Алгоритм слишком сильно подогнался под обучающую выборку ценой того, что он будет давать плохие ответы на новых точках.

Пример линейной регрессии в Sklearn

1. Импорт библиотек

```
from matplotlib.colors import ListedColormap
from sklearn import model_selection, datasets, linear_model, metrics

import numpy as np
```

```
%pylab inline
```

2. Генерация данных

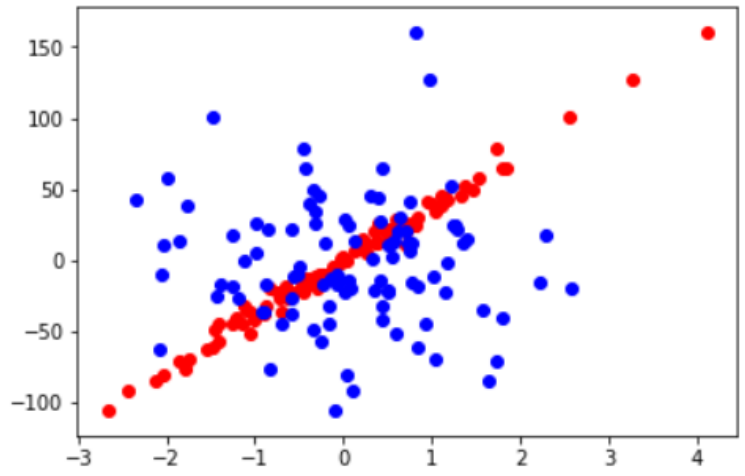
```
data, target, coef = datasets.make_regression(n_features = 2, n_informative = 1, n_targets = 1,
                                              noise = 5., coef = True, random_state = 2)
```

Создадим датасет из двух признаков, один из них информативный

Добавим немного шума

```
pylab.scatter(data[:,0], target, color = 'r')
pylab.scatter(data[:,1], target, color = 'b')
```

```
<matplotlib.collections.PathCollection at 0x7f5bd0269f60>
```



Какой из двух признаков является информативным?
1 признак – красные точки
2 признак – синие точки

```
train_data, test_data, train_labels, test_labels = model_selection.train_test_split(data, target,
                                                                                    test_size = 0.3)
```

LinearRegression

```
linear_regressor = linear_model.LinearRegression()
linear_regressor.fit(train_data, train_labels)
predictions = linear_regressor.predict(test_data)
```

1) Создаем экземпляр класса LinearRegression из модуля sklearn.linear_model,
2) Обучаем модель с помощью метода fit на обучающих данных,
3) Предсказываем значения выходного параметра на тестовой выборке с помощью predict

```
print(test_labels)
```

Истинные значения

```
[  64.19559505   23.87701013   27.83791274   -4.38652971  -19.36956003
   19.66406455   51.87072011  -40.84204295  -19.16792315  -44.51417742
  -22.32195021   28.15553021  -71.3715844   -22.64686884   13.02656201
  -24.77820218  -16.30914909  -16.79027112  -76.75213382  -13.26392817
   45.05465366   22.2276832  -105.77758163  -11.18242389   41.95683853
   25.24428409  -18.57607726  -12.98848753   24.82763821  -91.477377  ]
```

```
print(predictions)
```

Предсказанные значения

```
[  70.46040865   32.04198182   23.12210782   -5.57200284  -32.7300377
   26.65605518   52.97502966  -46.56718755  -10.53466845  -55.78515619
  -22.82420468   22.46701796  -71.60183332  -17.92152423   18.56038195
  -28.05019969  -12.28925698  -15.33841264  -70.69031042  -16.91157379
   42.1997639    17.96762582 -103.56655493  -13.75547179   42.76296885
   14.30473012  -25.35764508  -17.06870759   31.06714771  -94.73130759]
```

```
metrics.mean_absolute_error(test_labels, predictions)
```

Вычисляем среднюю абсолютную ошибку

```
4.885018353200092
```

```
linear_scoring = model_selection.cross_val_score(linear_regressor, data, target, scoring = 'neg_mean_absolute_error',
                                                 cv = 10)
print('mean: {}, std: {}'.format(linear_scoring.mean(), linear_scoring.std()))
```

```
mean: -4.070071498779695, std: 1.0737104492890204
```

* Откуда минус?

```
scorer = metrics.make_scorer(metrics.mean_absolute_error, greater_is_better = True)
```

```
linear_scoring = model_selection.cross_val_score(linear_regressor, data, target, scoring=scorer,
                                                 cv = 10)
print('mean: {}, std: {}'.format(linear_scoring.mean(), linear_scoring.std()))
```

```
mean: 4.070071498779695, std: 1.0737104492890204
```

А это что за чудеса?

В задачах регрессии для оценки качества алгоритмов используются те же методы, что и в задачах классификации. Например, можно использовать кросс-валидацию.

```
linear_regressor.coef_
```

```
array([38.62680483,  0.76650446])
```

```
# в лекции не указано, что в уравнении обученной модели также участвует свободный член
linear_regressor.intercept_
```

```
-0.7704209865438617
```

Уравнение регрессии

```
print("y = {:.2f}*x1 + {:.2f}*x2 + {:.2f}".format(linear_regressor.coef_[0],
                                                  linear_regressor.coef_[1],
                                                  linear_regressor.intercept_))
```

```
y = 38.63*x1 + 0.77*x2 + -0.77
```

* Как вы уже знаете, кросс-валидация часто применяется для подбора параметров, а при подборе параметров используется максимизация метрики. В данном случае мы используем метрику, которая увеличивается, когда модель становится хуже, поэтому метрика умножается на -1.
Но в рассматриваемой задаче мы не подбираем параметры, поэтому избавимся от минуса, создав свой собственный scorer.