

\mathbb{X} — множество объектов, $\mathbb{Y} = \mathbb{R}$ — множество ответов

$X = \{(x_1, y_1), (x_2, y_2), \dots, (x_\ell, y_\ell)\}$ — обучающая выборка

$x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$

Считаем, что есть неизвестная зависимость $y : \mathbb{X} \rightarrow \mathbb{Y}$, которую нужно восстановить по известным $y_i = y(x_i)$.

Модель зависимости: $a(x) = f(x, w)$, где $w \in \mathbb{R}^p$ — параметр, f — некоторое выбранное нами семейство функций.

Предполагаем, что $y(x) \approx a(x)$ при некотором w .

Задача: найти w .

Метод наименьших квадратов (МНК)

Выбираем w , чтобы минимизировать остаточную сумму квадратов (residual sum of squares, RSS), то есть решаем задачу оптимизации:

$$Q(w, X) = \sum_{i=1}^{\ell} (f(x_i, w) - y_i)^2 \rightarrow \min_w$$

Пусть

$$f(x, w) = \sum_{k=1}^d w_k x^k + w_0 = \langle w, x \rangle + w_0$$

$y = w_1 x^1 + w_0$ — прямая на плоскости;

$y = w_2 x^2 + w_1 x^1 + w_0$ — плоскость в трёхмерном пространстве;

$y = \langle w, x \rangle + w_0$ — гиперплоскость в d -мерном пространстве;

Трюк: добавляем константный признак

$$\mathbf{x} \rightsquigarrow \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \qquad \mathbf{w} \rightsquigarrow \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

$$\langle w, x \rangle + w_0 \rightsquigarrow \langle \tilde{w}, \tilde{x} \rangle$$

Линейная регрессия + МНК = Ordinary Least Squares (OLS)

$$Q(w, X) = \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2 \rightarrow \min_w$$

X — матрица «объекты \times признаки», y — целевой вектор:

$$Q(w, X) = \|Xw - y\|_2^2 \rightarrow \min_w$$

$$\nabla_w Q(w, X) = 2X^T(Xw - y)$$

$$\nabla_w Q(w, X) = 0 \quad \Rightarrow \quad \boxed{w = (X^T X)^{-1} X^T y}$$

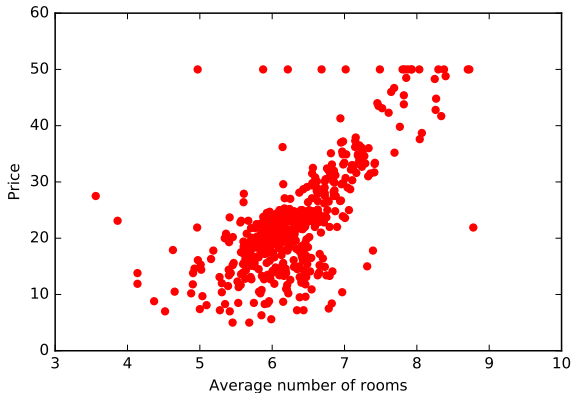
В реальной жизни такой формулой никто не пользуется, но это уже совсем другая история...

Далее будем исследовать средние цены на дома разных районов Бостона:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
boston = load_boston()
print(boston.DESCR)
```

Среднее количество комнат

```
plt.scatter(boston.data[:, 5], boston.target, color='r')  
plt.xlabel('Average number of rooms')  
plt.ylabel('Price')
```



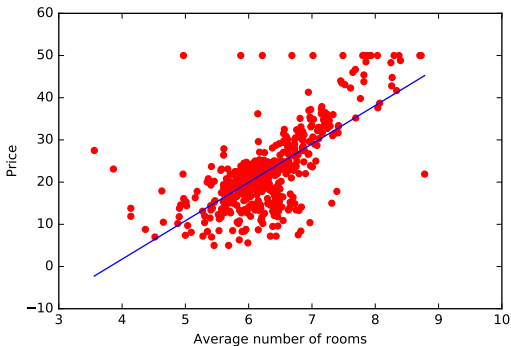
```
LinearRegression(self, fit_intercept=True,  
                 normalize=False, copy_X=True, n_jobs=1)
```

```
from sklearn.linear_model import LinearRegression
```

```
lr = LinearRegression()  
lr.fit(boston.data[:, 5].reshape(-1, 1), boston.target)  
# Строим равномерную сетку из 100 точек:  
grid = np.linspace(boston.data[:, 5].min(),  
                  boston.data[:, 5].max(), 100)  
# Строим предсказание в точках этой сетки:  
best_line = lr.predict(grid.reshape(-1, 1))
```


Результат работы одномерной регрессии

```
plt.scatter(boston.data[:, 5], boston.target, color='r')  
plt.plot(grid, best_line)  
plt.xlabel('Average number of rooms')  
plt.ylabel('Price')
```



Как измерить качество? – I

Среднеквадратичная ошибка (Mean Squared Error):

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

```
from sklearn.metrics import mean_squared_error

y_pred = lr.predict(boston.data[:, 5].reshape(-1, 1))
# Ошибка на обучающей выборке:
MSE = mean_squared_error(boston.target, y_pred)
print("MSE =", MSE)
```

Оказалось, что $\text{MSE} = 43.6$.

Хорошо это или плохо?

Как измерить качество? – II

Можно ввести «нормированный» MSE — так называемый коэффициент детерминации или R^2 :

$$R^2 = 1 - \frac{\sum_{i=1}^{\ell} (a(x_i) - y_i)^2}{\sum_{i=1}^{\ell} (y_i - \bar{y})^2}, \quad \text{где} \quad \bar{y} = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i.$$

$R^2 \approx 1$ — отлично, $R^2 \approx 0$ — плохо, $R^2 < 0$ — ужасно!

```
from sklearn.metrics import r2_score  
r2 = r2_score(boston.target, y_pred)
```

Оказалось, что $R^2 = 0.48$.

Mean Absolute Error:

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

Root Mean Squared Error:

$$\text{RMSE} = \sqrt{\text{MSE}}$$

И так далее...

Нормировка признаков

Отнормируем признаки так, чтобы среднее значения каждого было равно нулю, а среднеквадратичное отклонение — единице:

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()  
scaler.fit(boston.data)  
X = scaler.transform(boston.data)
```

```
# Проверим, что всё действительно работает:
```

```
print("new mean =", np.mean(X, axis=0))  
print("new std =", np.std(X, axis=0))
```

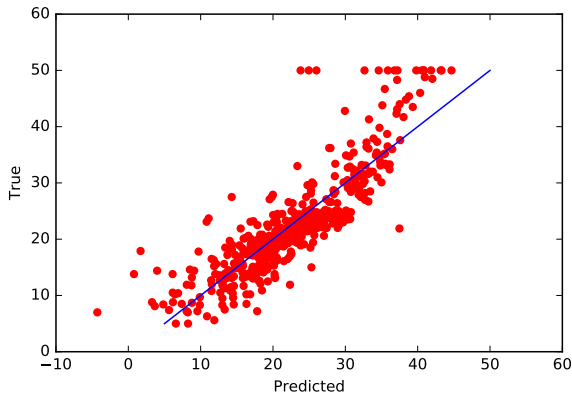
Обучим линейную регрессию на всех признаках:

```
y = boston.target  
lr.fit(X, y)
```

Построим график того, насколько предсказание отличается от фактической цены:

```
y_pred = lr.predict(X)  
plt.scatter(y_pred, y, color='r')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.plot([y.min(), y.max()], [[y.min()], [y.max()]])
```

Многомерная регрессия



$$\text{MSE} = 21.9$$

$$R^2 = 0.74$$

Если признаки отнормированные, то коэффициенты регрессии отвечают за «важность признаков».

```
idx = np.argsort(-np.abs(lr.coef_))  
idx * np.sign(lr.coef_[idx]).astype(int)  
-12,  -7,   5,   8,  -9, -10,  -4,   1,   0,  11,   3,   2,   6
```

0. CRIM per capita crime rate by town
1. ZN proportion of residential land zoned for lots over 25,000 sq.ft.
2. INDUS proportion of non-retail business acres per town
3. CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
4. NOX nitric oxides concentration (parts per 10 million)
5. RM average number of rooms per dwelling
6. AGE proportion of owner-occupied units built prior to 1940
7. DIS weighted distances to five Boston employment centres
8. RAD index of accessibility to radial highways
9. TAX full-value property-tax rate per \$10,000
10. PTRATIO pupil-teacher ratio by town
11. B $1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town
12. LSTAT % lower status of the population

Что делать с категориальными признаками?

В этом датасете категориальных признаков нет, но в жизни они встречаются.

В sklearn'е это:

```
sklearn.preprocessing.OneHotEncoder(n_values='auto',  
    categorical_features='all',  
    dtype=<class 'float'>,  
    sparse=True,  
    handle_unknown='error')
```

Признак, который принимает n значений, превращаем в вектор длины n , состоящий из одной единицы и $n - 1$ нуля.

В pandas это функция `pd.get_dummies(...)`

5-fold Cross Validation

Обучаемся на 80% выборки, оцениваем на оставшихся 20%.
Получим более адекватную оценку ошибки на новых данных.

```
from sklearn.cross_validation import KFold
kf = KFold(X.shape[0], n_folds=5)
y_pred = np.zeros(y.shape)
for train, test in kf:
    lr.fit(X[train], y[train])
    y_pred[test] = lr.predict(X[test])

print("r2 =", r2_score(y, y_pred))
print("MSE =", mean_squared_error(y, y_pred))
print("RMSE =", np.sqrt(mean_squared_error(y, y_pred)))
```

$R^2 = 0.56$, $MSE = 37.17$, $RMSE = 6.10$

А не переобучаемся ли мы?

Наблюдение: очень большие веса признаков говорят о переобучении.

Идея: «штрафовать» большие веса признаков.

Гребневая (Ridge Regression) или L_2 -регрессия

Пусть штраф — это квадрат 2-нормы весов, то есть

$$\|w\|_2^2 = w_1^2 + w_2^2 + \dots + w_d^2$$

Задача:

$$Q(X, w) = \|Xw - y\|_2^2 + \alpha\|w\|_2^2 \rightarrow \min_w,$$

где α — это параметр.

Решение:

$$w = (X^T X + \alpha I)^{-1} X^T y$$

L_1 -регрессия или LASSO (Least Absolute Shrinkage and Selection Operator) ¹

Пусть штраф — это 1-норма весов, то есть

$$\|w\|_1 = |w_1| + |w_2| + \dots + |w_d|$$

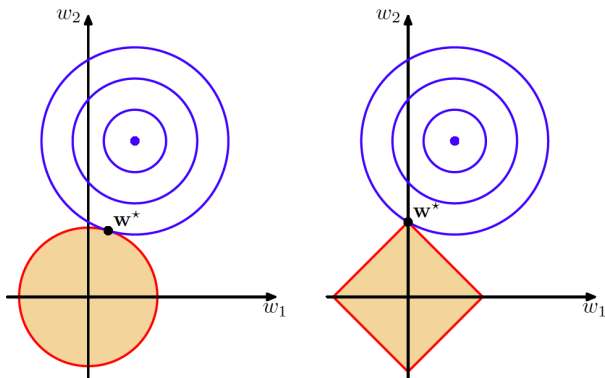
Задача:

$$Q(X, w) = \|Xw - y\|_2^2 + \alpha \|w\|_1 \rightarrow \min_w,$$

Аналитического решения нет, считается итерационными методами оптимизации.

¹Придумана в 1996 году

Геометрия L_1 и L_2 регуляризаторов



На картинке одно из объяснений, почему L_1 -регуляризатор называется разреживающим.

Ridge regression в scikit-learn

```
from sklearn.linear_model import Ridge
# Сначала возьмём альфа с потолка:
lr = Ridge(alpha=0.1)
kf = KFold(X.shape[0], n_folds=5)
y_pred = np.zeros(y.shape)
for train, test in kf:
    lr.fit(X[train], y[train])
    y_pred[test] = lr.predict(X[test])

print("r2 =", r2_score(y, y_pred))
print("MSE =", mean_squared_error(y, y_pred))
print("RMSE =", np.sqrt(mean_squared_error(y, y_pred)))
```

$$R^2 = 0.615, \text{MSE} = 32.472, \text{RMSE} = 5.698$$

Получили решение с лучшей обобщающей способностью!

	kNN Regression	Ridge Regression
Параметры	вся обучающая выборка ($\ell \times d$)	$(d + 1)$ -мерный вектор весов w
Гиперпараметры	метрика ρ , число соседей k	Коэффициент регуляризации α
Обучение (fit)	запоминаем всю обучающую выборку (нет обучения)	$\ Xw - y\ _2^2 + \alpha \ w\ _2^2 \rightarrow \min_w$
Построение прогноза (predict)	$a(x) = \frac{\sum_{i=1}^k w_i y_{(i)}}{\sum_{i=1}^k w_i}$	$a(x) = \langle w, x \rangle$

Ridge regression: как подбирать α ?

Для выбора α для гребневой регрессии в scikit-learn написана эффективная процедура Leave-One-Out кросс-валидации²

```
from sklearn.linear_model import RidgeCV
```

```
# Переберём 100 точек от 1e-4 до 10 на логарифмической шкале:
```

```
cv = RidgeCV(alphas=np.logspace(-4, 1, 100, base=10))
```

```
cv.fit(X, y)
```

```
cv.alpha_
```

Вывод: L_2 -регрессия имеет лучшую обобщающую способность, чем OLS, и очень быстро обучается.

²Обучаем алгоритм на всей выборке без одного объекта, а потом предсказываем ответ на этом объекте. И так для каждого объекта. А потом выбираем лучший результат.

LASSO в sklearn

```
from sklearn.linear_model import Lasso

# Опять возьмём alpha с потолка
lr = Lasso(alpha=0.01)
kf = KFold(X.shape[0], n_folds=5)
y_pred = np.zeros(y.shape)
for train, test in kf:
    lr.fit(X[train], y[train])
    y_pred[test] = lr.predict(X[test])

print("r2 =", r2_score(y, y_pred))
print("MSE =", mean_squared_error(y, y_pred))
print("RMSE =", np.sqrt(mean_squared_error(y, y_pred)))
```

$R^2 = 0.576$, $MSE = 35.750$, $RMSE = 5.980$

Обобщающая способность чуть лучше, но в этот раз наугад взятый коэффициент оказался не так хорош.

LASSO: как подбирать α ?

Так же:

```
from sklearn.linear_model import LassoCV

cv = LassoCV(alphas=np.logspace(-4, 1, 100, base=10), n_jobs=-1)
cv.fit(X, y)
cv.alpha_
```

Библиотека scikit-learn хороша одинаковым интерфейсом к разным функциям.

LASSO: разреживание признаков – I

```
lr = Lasso(alpha=0.01)
lr.fit(X, y)
lr.coef_

array([ -6.91894958,   6.86587554,  -0.          ,  13.43981379,
        -22.31001393,  66.85016996,  -0.          , -35.17691333,
         0.          ,  -0.          , -39.84958238,  14.92602188,
        -83.60763445])
```

Отбросили признаки:

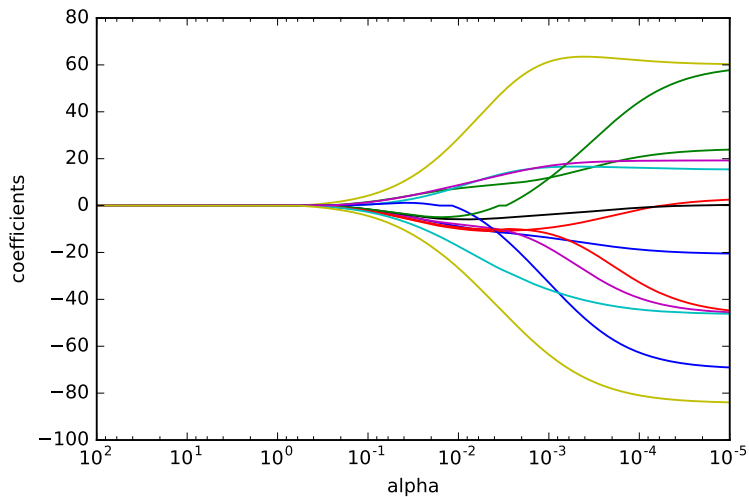
- proportion of non-retail business acres per town
- proportion of owner-occupied units built prior to 1940
- index of accessibility to radial highways
- full-value property-tax rate per \$10,000

LASSO: отбор признаков – II

```
lr = Lasso()
alphas, coefs, _ = lr.path(X, y, alphas=np.logspace(-5, 2, 1000))

fig, ax = plt.subplots()
ax.plot(alphas, coefs.T)
# Логарифмическая шкала:
ax.set_xscale('log')
ax.set_xlim(alphas.max(), alphas.min())
plt.xlabel('alpha')
plt.ylabel('coefficients')
```

LASSO: отбор признаков – III



$$Q(X, w) = \|Xw - y\|_2^2 + \alpha_1 \|w\|_1 + \alpha_2 \|w\|_2^2 \rightarrow \min_w$$

Функции с таким же интерфейсом:

```
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import ElasticNetCV
```

Используется реже, потому что два гиперпараметра сложнее настраивать, чем один.

Вопросы?