Лекция 7:
оценка и выбор моделей

Параметрические модели

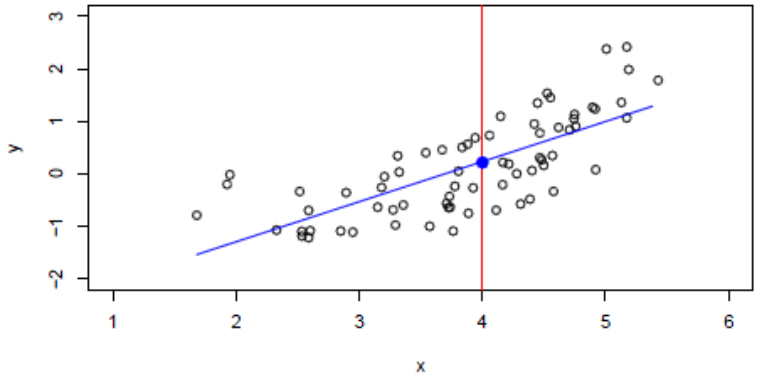
Линейная модель представляет собой важный пример параметрической модели:

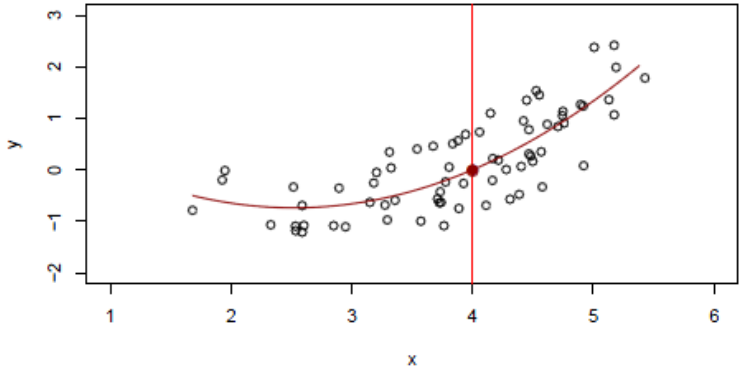
$$f_L(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots \beta_p X_p.$$

- Линейная модель определяется *p + 1* параметрами

$$\beta_0, \beta_1, \ldots, \beta_p.$$

- Мы оцениваем параметры на основе подгонки модели на обучающем наборе данных.
- Хотя такие модели *почти никогда* не показывают очень хорошую точность, но служат хорошей и интерпретируемой аппроксимацией неизвестной истинной функции *f(X)*.





Линейная регрессия

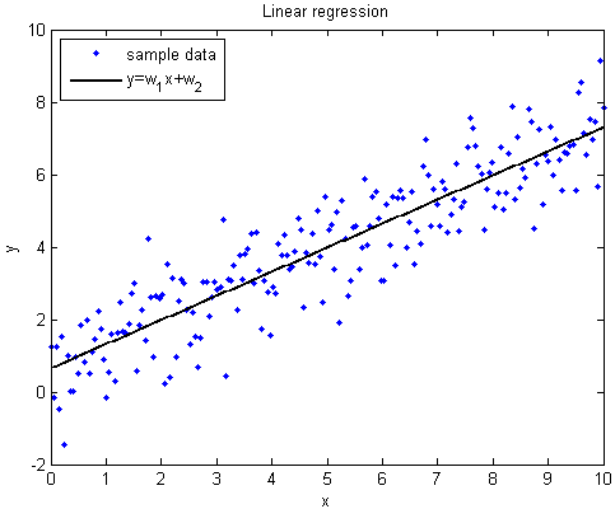
- Задача регрессии:

$$y(x_1,...,x_p) = E(Y \mid X_1 = x_1,..., X_p = x_p)$$

- Уравнение линейной регрессии:

$$f(X) = b_0 + \sum_{j=1}^{p} X_j b_j + \varepsilon$$

  - ε = N(0,σ²) - шум
  - Y –отклик
  - X=(X_1,…, X_p) - регрессоры (предикторы)
  - b – параметры модели



- Линеаризируемые регрессии:
  - Степенная ,Экспоненциальная
  - Гиперболическая, и другие

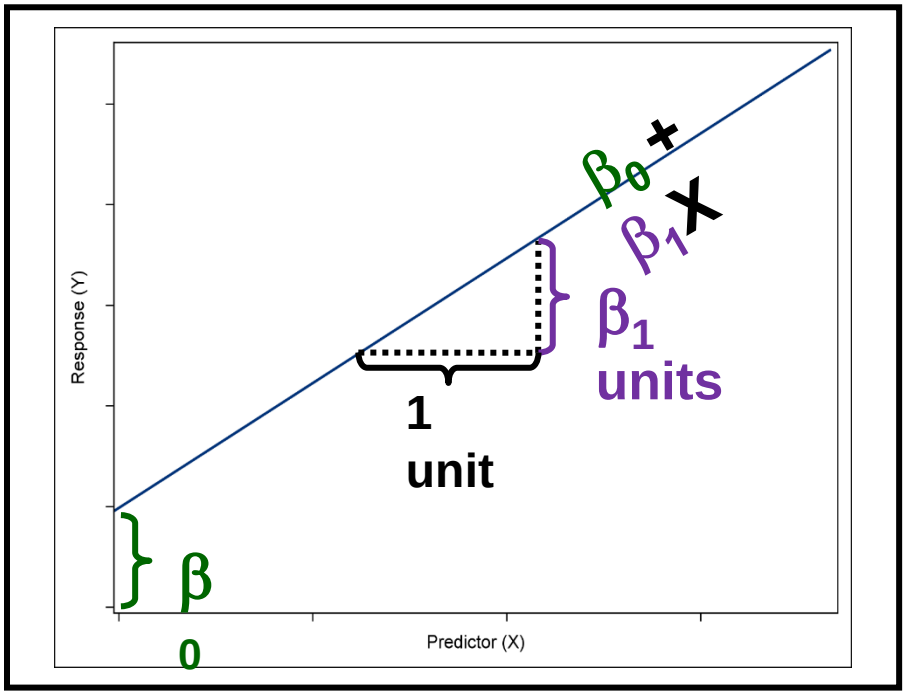
$$y = a x_1^{b_1} x_2^{b_2} ... x_p^{b_p} \varepsilon,$$

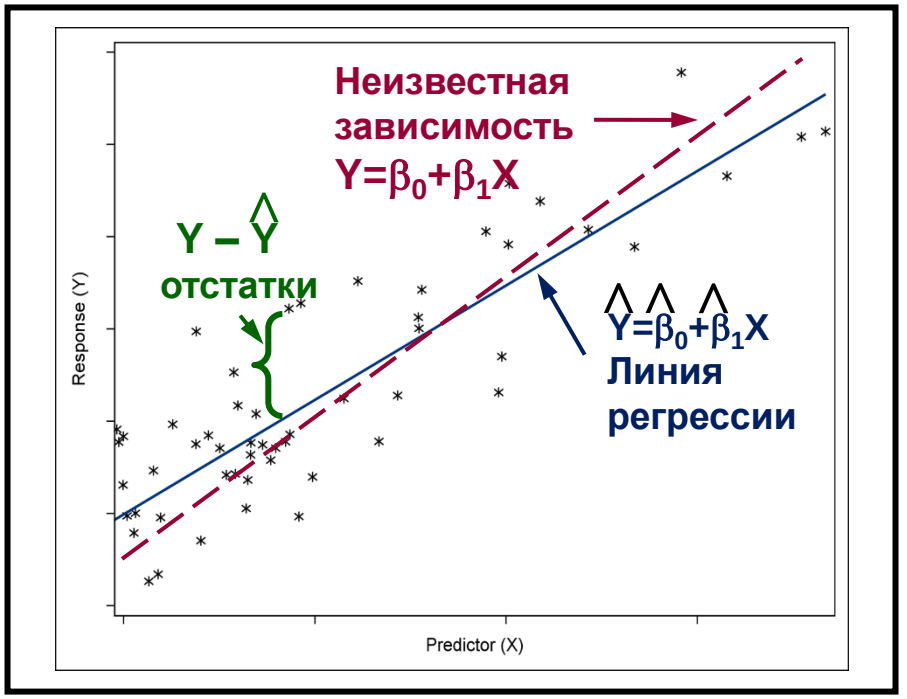
$$y = e^{a+b_1x_1+b_2x_2+...+b_px_p+\varepsilon},$$

$$y = (a + b_1x_1 + b_2x_2 + ...b_px_p + \varepsilon)^{-1}$$

- Цель регрессионного анализа:
  - Определение наличия связи между переменными и характера этой связи (подбор уравнения)
  - Предсказание значения зависимой переменной с помощью независимой(-ых)
  - Определение вклада отдельных независимых переменных в вариацию зависимой

Простая линейная регрессия

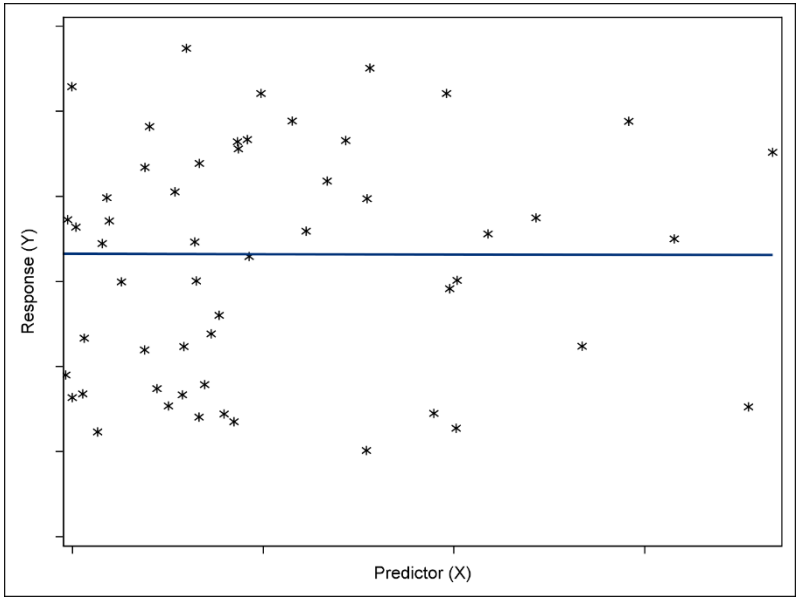


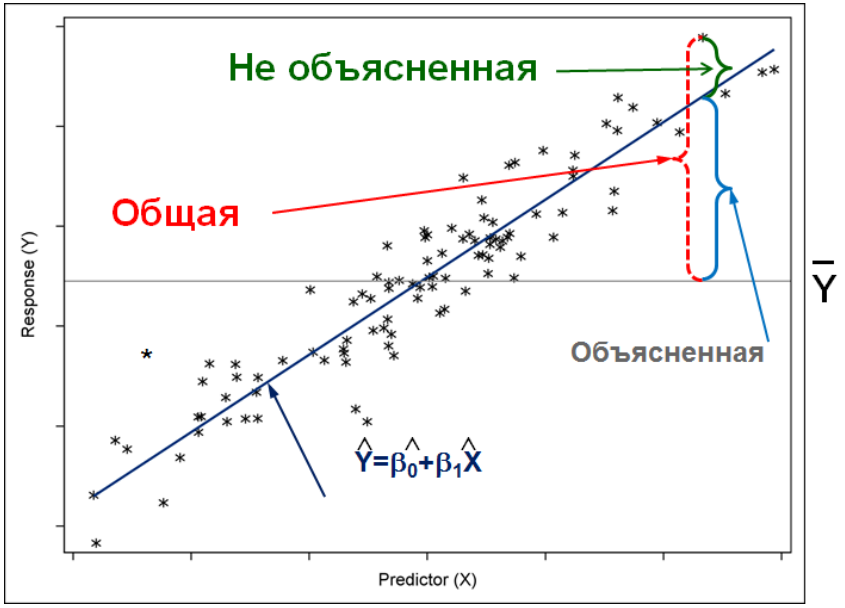


Предположения:

- Независимость наблюдений
- Выбранное уравнение регрессии (например, линейное) соответсвует истинной зависимости в данных
- Нормальность ошибки (с константной дисперсией по всем наблюдениям)

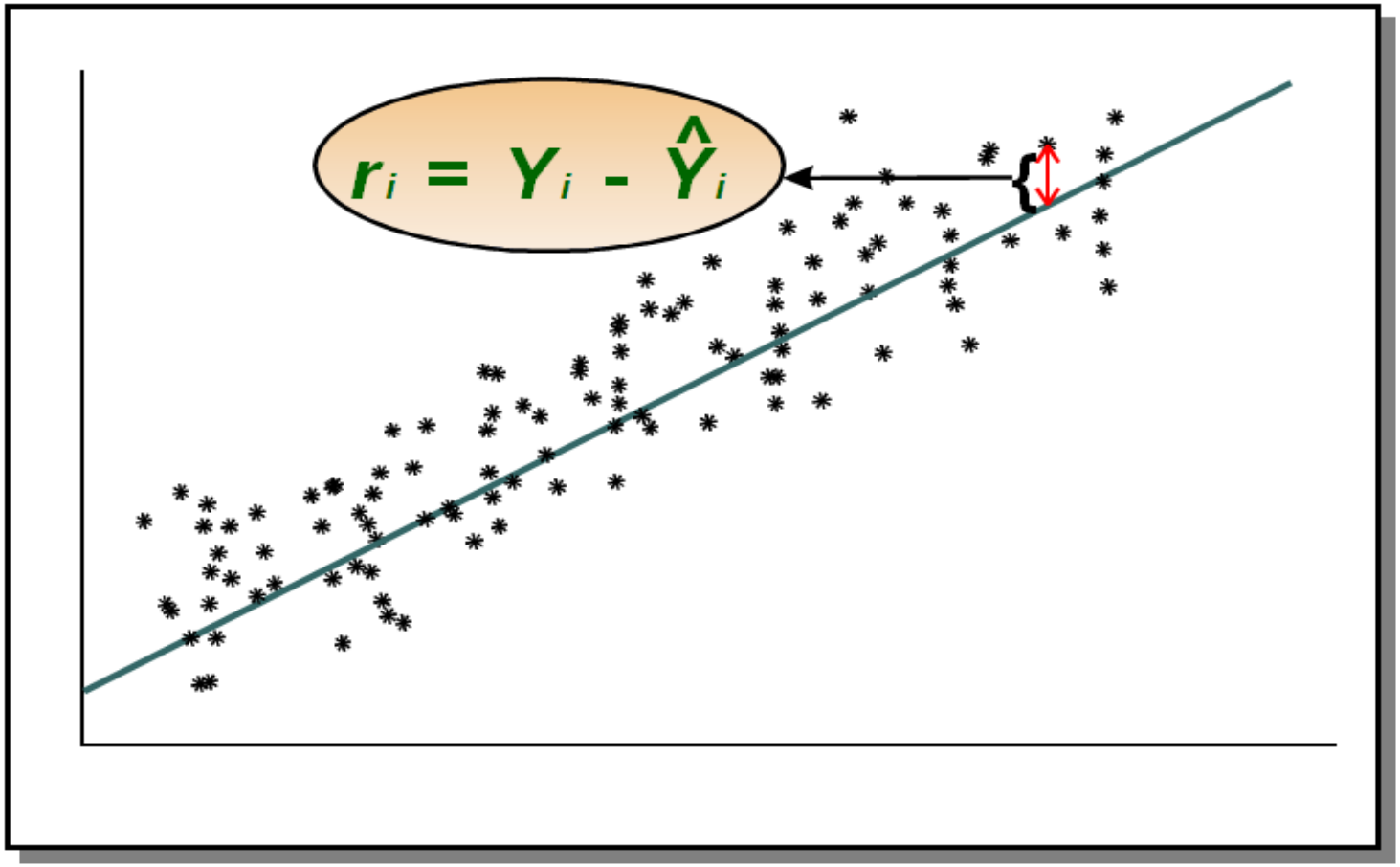
Базовая модель (Нулевая гипотеза)





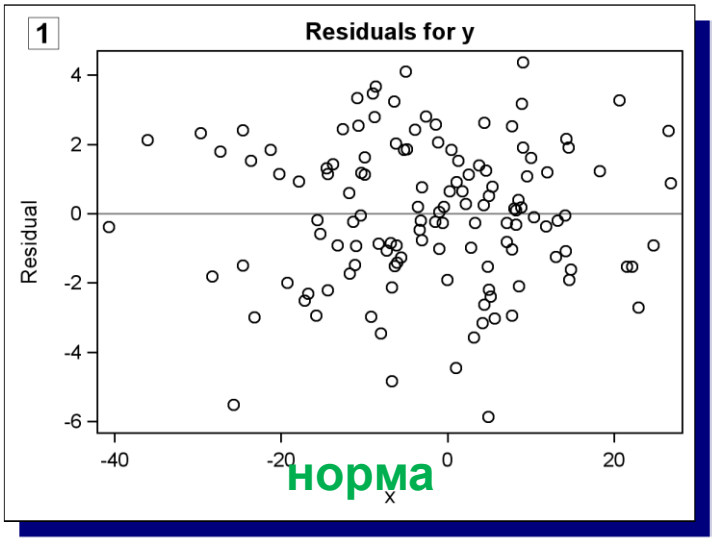
- Нулевая гипотеза:
  - Регрессионная модель приближает наблюдаемые данные не лучше базовой модели – константы ($\beta_1=0$)
- Альтернативная гипотеза:
  - Регрессионная модель лучше приближает наблюдаемые данные чем базовая модель – константа ($\beta_1\neq0$)

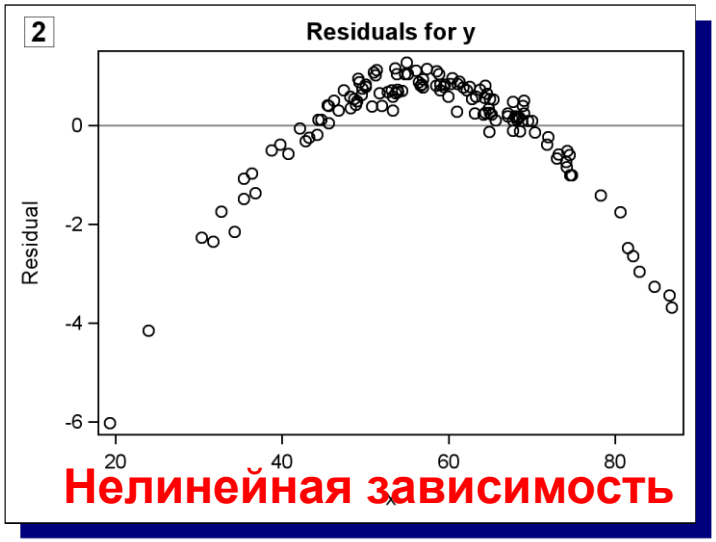
Проверка предположений модели с помощью графиков остатков

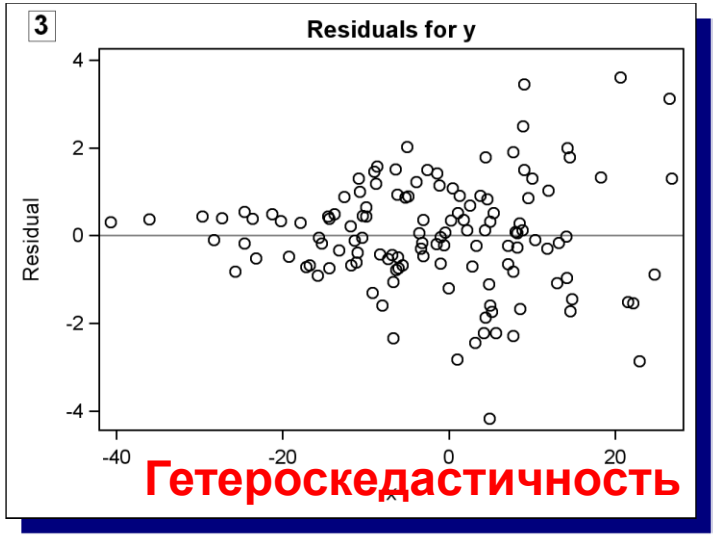


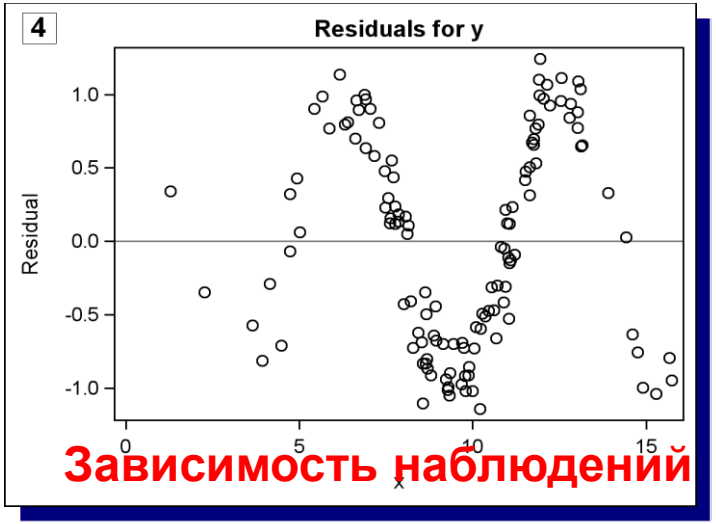
Графики: как остатки зависят от прогноза, от отклика, от предикторов

Графики остатков







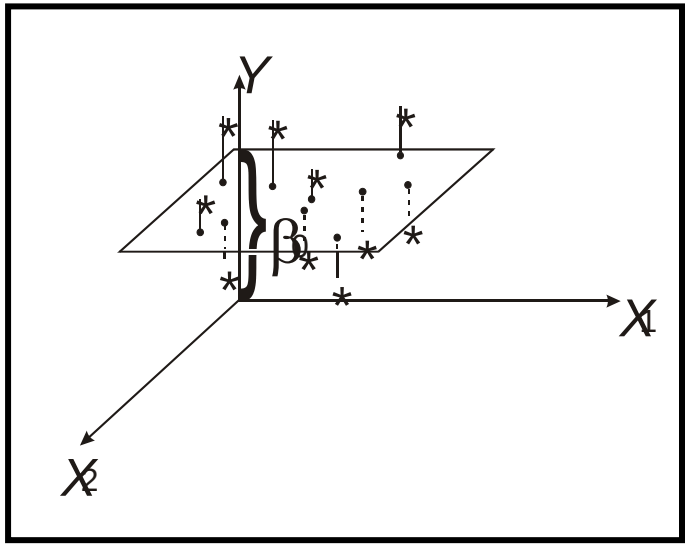


Множественная линейная регрессия

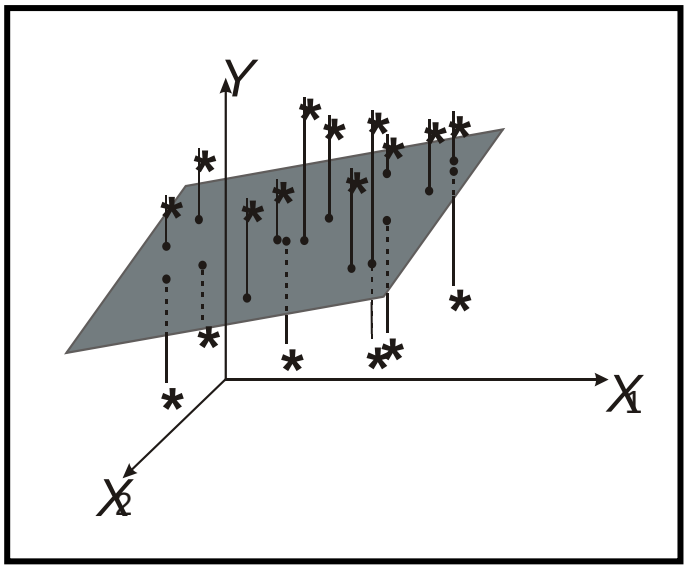
- Пример линейной модели с двумя переменными

$$Y=\beta_0+\beta_1X_1+\beta_2X_2+\varepsilon, \text{ где}$$

Y – отклик, X_1 и X_2 предикторы, ε - ошибка, β_0, β_1, и β_2-параметры (неизвестные)



Нет зависимости

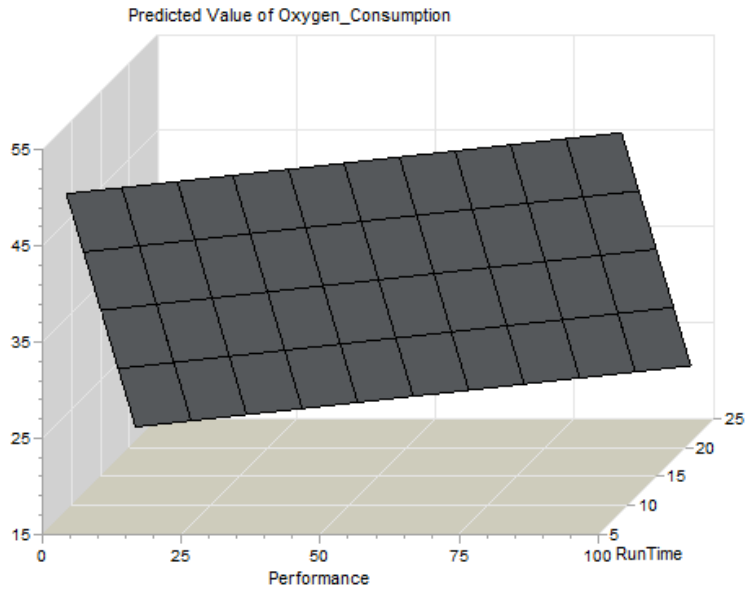


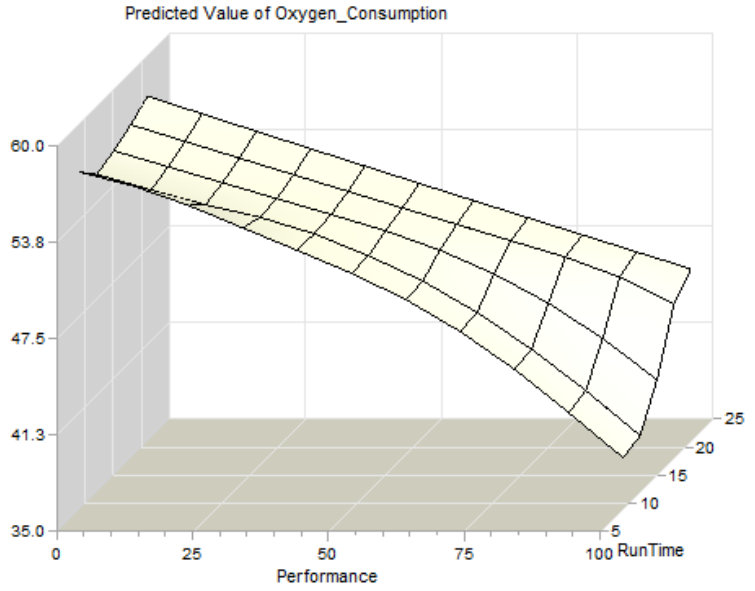
Есть зависимость

Множественная линейная регрессия

- В общем случае ищем зависимость как линейную комбинацию *k* предикторов X_1 - X_k:

$$Y=\beta_0+\beta_1X_1+...+\beta_kX_k+\varepsilon$$





$$Y=\beta_0+\beta_1X_1+\beta_2X_2+\varepsilon$$

Линейная модель с линейными эффектами

$$Y=\beta_0+\beta_1X_1+\beta_2X_1^2+\beta_3X_2+\beta_4X_2^2+\varepsilon$$

Линейная модель с нелинейными эффектами

Метод наименьших квадратов и проблема мультиколлинеарности

- Оценка ошибки = сумма регрессионных остатков (квадратичная функция потерь):

$$RSS(B) = \sum_{i=1}^{N} (y_i - f(\bar{x}_i))^2 = \sum_{i=1}^{N} (y_i - b_0 - \sum_{j=1}^{p} x_{ij} b_j)^2$$

- В матричной форме:

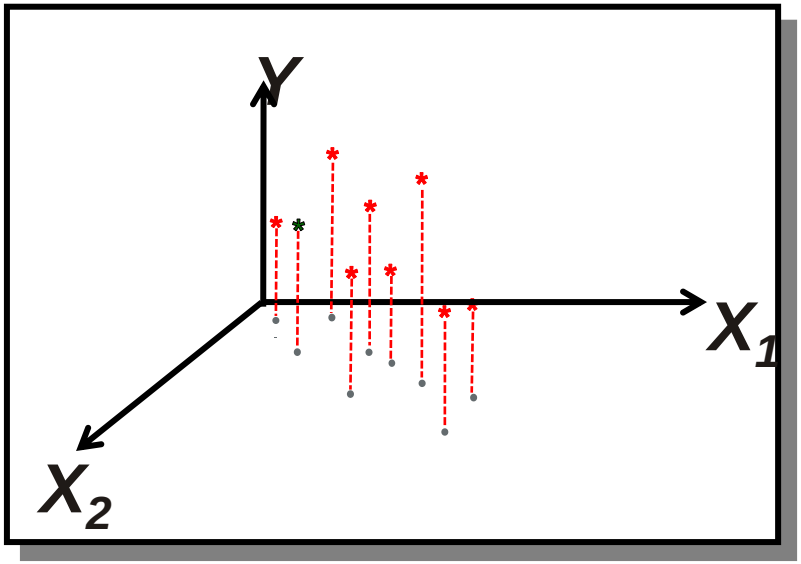
$$RSS(B) = (\bar{y} - XB)^T (\bar{y} - XB)$$

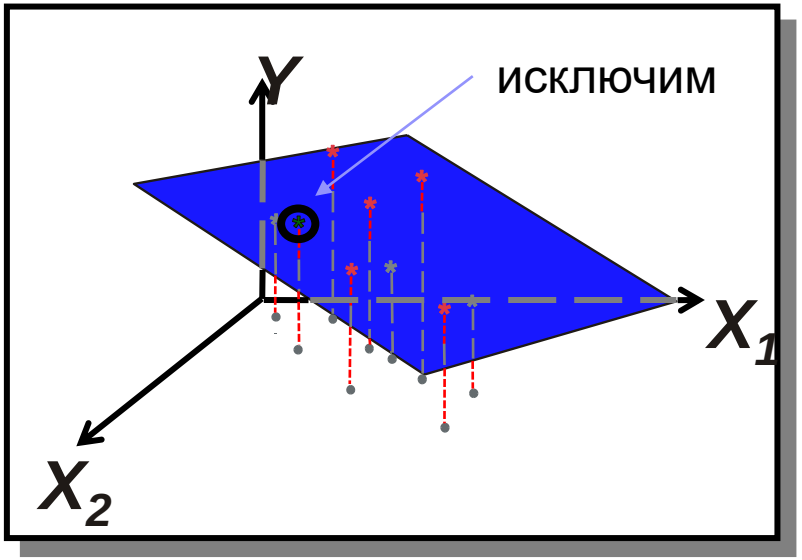
- Единственное оптимальное решение (если матрица данных не сингулярная)

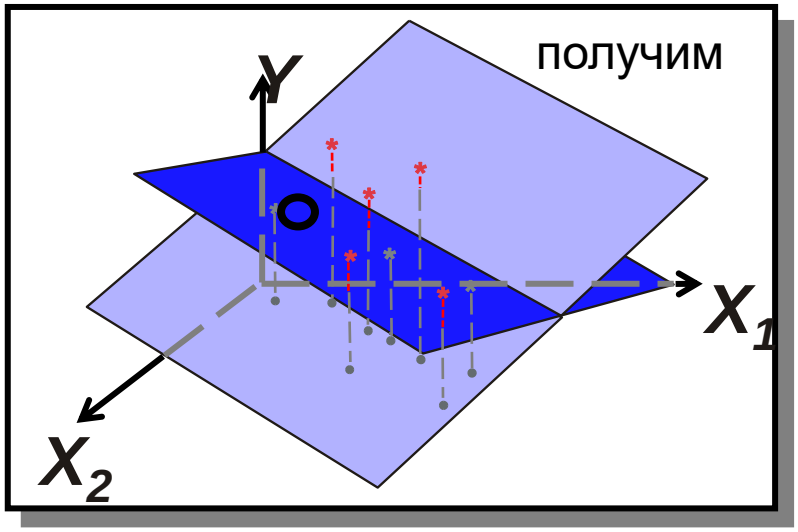
$$B = (X^T X)^{-1} X^T \bar{y}$$

- Недостатки:
  - Сингулярная матрица данных из-за коррелированных факторов
  - Большое число регрессоров – плохая точность и интерпретируемость
- Основные подходы:
  - Поиск и удаление зависимых и незначимых факторов
  - Использование «смещенных» регуляризированных моделей
  - переход к новым независимым факторам, например, с помощью метода главных компонент

Иллюстрация мультиколлинеарности







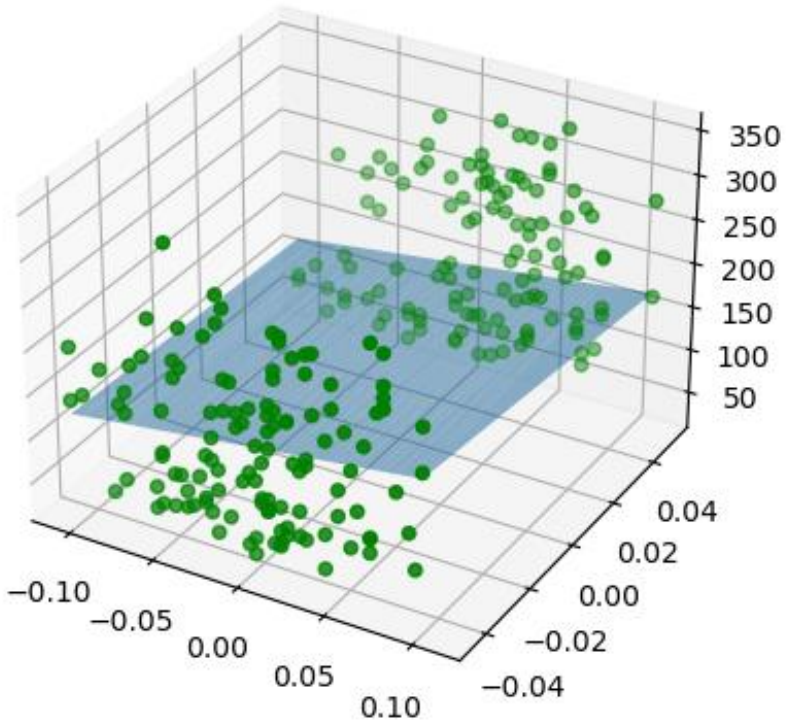
- Портятся статистики с оценкой значимости переменных
- Увеличивается вариативность оценки параметров и как следствие ошибка
- Есть тенденция к неограниченному росту коэф.

Пример (Python)

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

```
X_2d = X[:, :2]
X_2d_test = X_test[:, :2]
```

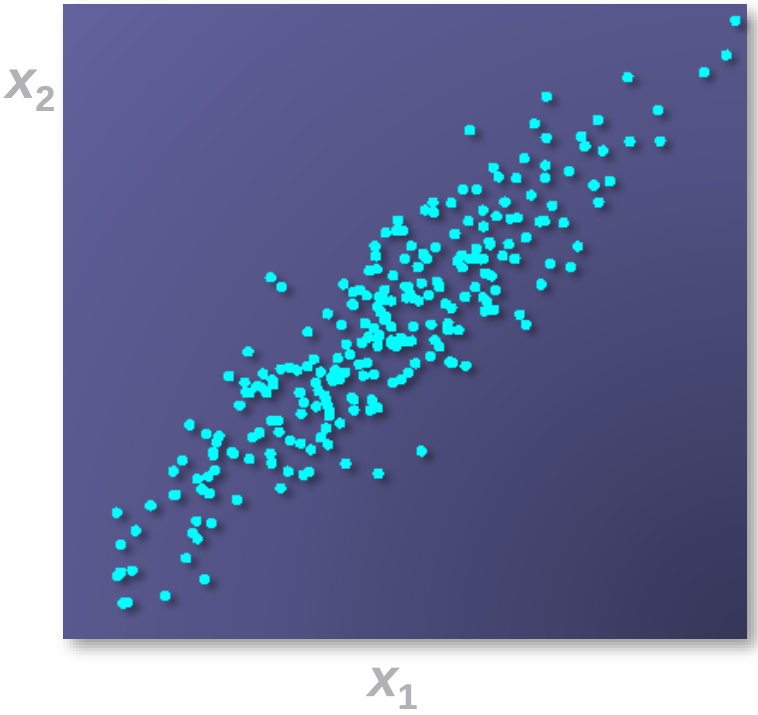
```
regr = LinearRegression()
regr.fit(X_2d, y)
pass
```



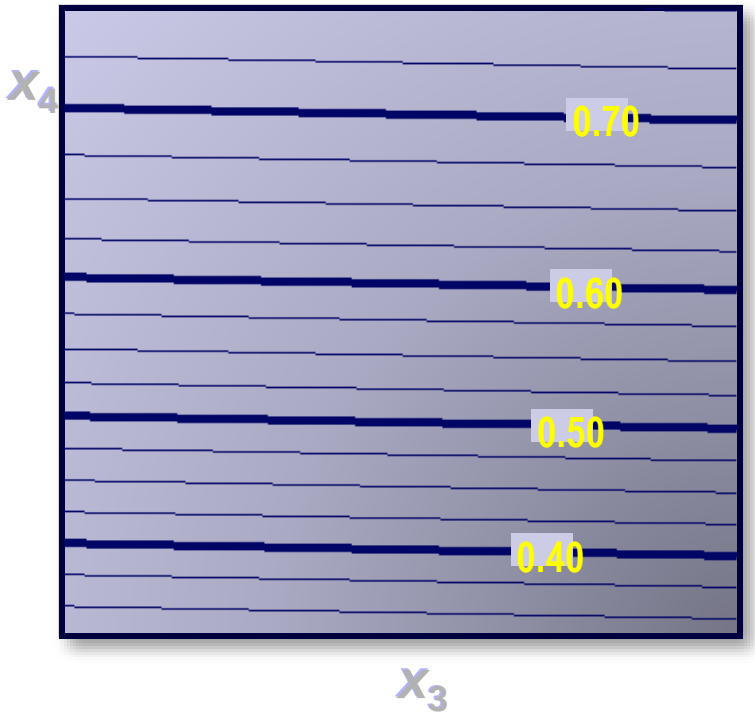

```
ax = plt.subplot(projection='3d')
ax.scatter(X_2d_test[:, 0], X_2d_test[:, 1], y_test, color="green")
ax.plot_trisurf(X_2d_test[:, 0], X_2d_test[:, 1], regr.predict(X_2d_test), alpha=0.5)
```

Проблемы входных переменных и для KNN и для МНК

Зависимость



Не релевантность отклику



Выхода два: либо преобразование либо исключение

...

Простые методы сокращения размерности

Дано: входные переменные {x1,…,xn} и выходная (числовая или бинарная) y

Задача: оставить только значимые и независимые xi

Работает в два этапа:

1. Уделяет все x_i, где $R^2(x_i)<T1$ - (*удаление незначимых)*
2. Forward stepwise регрессия $f(x_{i1},\ldots x_{ik})$ пока $R^2(f(x_{i1},\ldots xi_{ik}))-R^2(f(x_{i1},\ldots x_{ik-1}))>T2$
   - (*удаление зависимых)*

Преобразования переменных:

- Дискретизация непрерывных
- Группировка категориальных

Пример (Python)

```
from sklearn.feature_selection import SequentialFeatureSelector, VarianceThreshold
```

```
KNN = KNeighborsRegressor(n_neighbors=5)
```

```
VT = VarianceThreshold(threshold=0.002)
X_ = VT.fit_transform(X)
X.shape, X_.shape
```

```
((200, 10), (200, 9))
```

```
# KNN.score(X, y) returns R2 - performed until tol is satisfied
# direction: forward/backward - add or remove features
# for newer scikit-learn versions:
"""
selector = SequentialFeatureSelector(KNN, n_features_to_select="auto", tol=0.5, direction="backward")
"""
# for older scikit-learn versions:
selector = SequentialFeatureSelector(KNN, n_features_to_select=3, direction="backward")
selector.fit(X_, y)
X_ = selector.transform(X_)
```

Недостаток – не сохраняет значимости на каждой итерации

Возможное решение – наследуемся от модели (KNN) и добавляем в fit или score вывод своих логов

Простые методы сокращения размерности

- Unsupervised рассматривали в теме по самооганизации
- Supervised – его простая модификация:
  - X_1 - множество переменных, включенных в модель (изначально пустое),
  - X_2 - множество переменных, НЕ включенных в модель (изначально все переменные)
  - Y - отклик
  - На каждом делается перенос одной переменной x из X_2 в X_1 , такое чтобы:

$$\min \operatorname{Trace}\left(\mathbf{Y}^{\top}\left(\mathbf{I}-\mathbf{X}_1\left(\mathbf{X}_1^{\top}\mathbf{X}_1\right)^{-1}\mathbf{X}_1^{\top}\right)\mathbf{Y}\right)$$

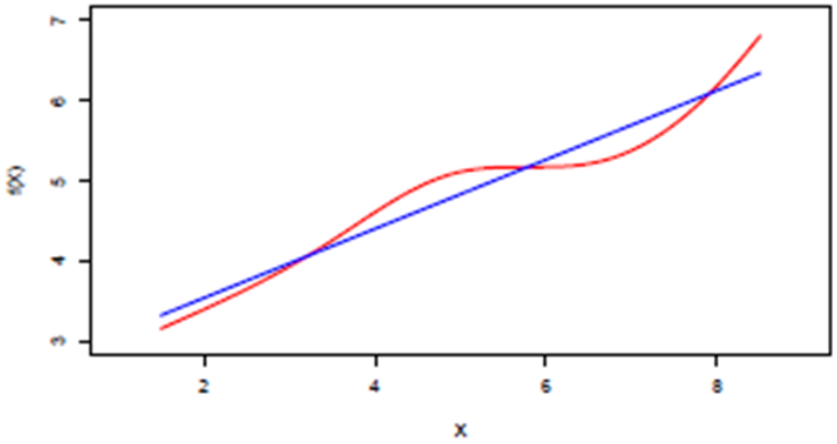
  - До тех пор, пока не отберется заданное число переменных, либо пока не будет описана заданная пропорция вариации по инф. критерию

Выбор модели и регуляризация на примере линейных моделей регрессии

- Отклик линейной модели

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon.$$

- Истинные зависимости редко бывают линейными
- Но: *интерпретируемость* и часто хорошее *качество прогнозирования*.
- Рассмотрим способы, при которых простая линейная модель может быть улучшена путем замены МНК некоторыми альтернативными методами подгонки.



Оценка точности расчета коэффициентов

- Стандартная ошибка оценки отражает, какие изменения происходят при повторной выборке

$$\mathrm{SE}(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \quad \mathrm{SE}(\hat{\beta}_0)^2 = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\right]$$

  где $\sigma^2 = \mathrm{Var}(\epsilon)$
- Эти стандартные ошибки могут быть использованы для вычисления *доверительных интервалов*.
- 95%-ый доверительный интервал определяется как диапазон значений, таких что с вероятностью 95% диапазон будет содержать истинное неизвестное значение параметра. Он имеет вид

$$\hat{\beta}_1 \pm 2 \cdot \mathrm{SE}(\hat{\beta}_1).$$

Проверка гипотез

- Стандартные ошибки также могут использоваться для *проверки гипотез* о коэффициентах.
- Наиболее распространенный вариант проверки гипотез заключается в проверке *нулевой гипотезы*

  H_0 : Нет никакой связи между *X* и *Y*

  против *альтернативной гипотезы*

  H_A : Между X и Y существует некоторая взаимосвязь.
- Математически это соответствует проверке

$$H_0 : \beta_1 = 0 \quad \text{vs} \quad H_A : \beta_1 \neq 0,$$

Проверка гипотез - продолжение

- Чтобы проверить нулевую гипотезу (о равенстве нулю коэфициента), мы вычисляем *t-статистику*

$$t = \frac{\hat{\beta}_1 - 0}{\mathrm{SE}(\hat{\beta}_1)},$$

- Она будет иметь *t*-распределение с *n-2* степенями свободы, предполагая $\beta_1 = 0$.
- Можно вычислить вероятность наблюдения любого значения, большего или равного |t|, это *p*-value.

	Coefficient	Std. Error	t-statistic	p-value
Intercept	7.0325	0.4578	15.36	< 0.0001
TV	0.0475	0.0027	17.67	< 0.0001

Оценка общей точности модели

- Мы вычисляем *стандартную ошибку невязок*

$$\mathrm{RSE} = \sqrt{\frac{1}{n-2}\mathrm{RSS}} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

где сумма квадратов невязок: $RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$

- *R-квадрат* или доля объясненной дисперсии

$$R^2 = \frac{\mathrm{TSS} - \mathrm{RSS}}{\mathrm{TSS}} = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}}$$

где $\mathrm{TSS} = \sum_{i=1}^{n}(y_i - \bar{y})^2$ - общая сумма квадратов.

- Можно показать, что в простой
- линейной регрессии $R^2 = r^2$,
- где r - корреляция между X и Y:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}.$$

$$F = \frac{(\mathrm{TSS} - \mathrm{RSS})/p}{\mathrm{RSS}/(n-p-1)} \sim F_{p,n-p-1}$$

Quantity	Value
Residual Standard Error	1.69
R^2	0.897
F-statistic	570

Интерпретация коэффициентов регрессии

- Идеальный сценарий: предикторы некоррелированы:

  - Каждый коэффициент можно оценить и тестировать отдельно.

  - Интерпретации такие как «*единичного изменение* X_j *связано с* β_j *–ым изменением в значении Y, тогда как все остальные переменные остаются фиксированными*».

- Корреляции между переменными вызывают проблемы:

  - Дисперсия всех коэффициентов имеет тенденцию к увеличению, иногда резкому

  - Интерпретации становятся непредсказуемыми - когда X_j меняется, все остальное тоже меняется.

- *«По сути, все модели ошибочны, но некоторые из них полезны»* (George Box)

Качественные (категориальные) признаки

Пример: исследовать различия в балансе кредитных карт между мужчинами и женщинами, не учитывая другие переменные.

Создается новая переменная

$$x_i = \begin{cases} 1 & \text{if } i\text{th person is female} \\ 0 & \text{if } i\text{th person is male} \end{cases}$$

Итоговая модель имеет вид:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i = \begin{cases} \beta_0 + \beta_1 + \epsilon_i & \text{if } i\text{th person is female} \\ \beta_0 + \epsilon_i & \text{if } i\text{th person is male.} \end{cases}$$

Интерпретация:

	Coefficient	Std. Error	t-statistic	p-value
Intercept	509.80	33.13	15.389	< 0.0001
gender[Female]	19.73	46.05	0.429	0.6690

Качественные признаки с более чем двумя возможными значениями

- Для признаков с несколькими возможными значениями создаются дополнительные фиктивные переменные. Например, для переменной ethnicity :

$$x_{i1} = \begin{cases} 1 & \text{if } i\text{th person is Asian} \\ 0 & \text{if } i\text{th person is not Asian,} \end{cases}$$

а вторая:

$$x_{i2} = \begin{cases} 1 & \text{if } i\text{th person is Caucasian} \\ 0 & \text{if } i\text{th person is not Caucasian.} \end{cases}$$

- Тогда обе эти переменные могут быть использованы в формуле регрессии и модель будет иметь вид

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \epsilon_i = \begin{cases} \beta_0 + \beta_1 + \epsilon_i & \text{if } i\text{th person is Asian} \\ \beta_0 + \beta_2 + \epsilon_i & \text{if } i\text{th person is Caucasian} \\ \beta_0 + \epsilon_i & \text{if } i\text{th person is AA.} \end{cases}$$

Качественные признаки с более чем двумя возможными значениями

- Число фиктивных переменных будет на единицу меньше, чем количество возможных различных значений, есть специальное *базовое значение*.

Level	D_A	D_B	D_C	D_D	D_E	D_F	D_G	D_H	D_I
A	1	0	0	0	0	0	0	0	0
B	0	1	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	0	0
D	0	0	0	1	0	0	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	0	0	1	0	0	0
G	0	0	0	0	0	0	1	0	0
H	0	0	0	0	0	0	0	1	0
I	0	0	0	0	0	0	0	0	1

	Coefficient	Std. Error	t-statistic	p-value
Intercept	531.00	46.32	11.464	< 0.0001
ethnicity[Asian]	-18.69	65.02	-0.287	0.7740
ethnicity[Caucasian]	-12.50	56.68	-0.221	0.8260

Расширения линейной модели

Удаление предположения аддитивности: *взаимодействующие* переменные и *нелинейность*

Взаимодействующие переменны:

- Раньше мы предполагали, что влияние предикторов на отклик независимо, например:

$$\widehat{\texttt{sales}} = \beta_0 + \beta_1 \times \texttt{TV} + \beta_2 \times \texttt{radio} + \beta_3 \times \texttt{newspaper}$$

- Модель с взаимосвязями имеет вид

$$\begin{aligned} \texttt{sales} &= \beta_0 + \beta_1 \times \texttt{TV} + \beta_2 \times \texttt{radio} + \beta_3 \times (\texttt{radio} \times \texttt{TV}) + \epsilon \\ &= \beta_0 + (\beta_1 + \beta_3 \times \texttt{radio}) \times \texttt{TV} + \beta_2 \times \texttt{radio} + \epsilon. \end{aligned}$$

Интерпретация примера

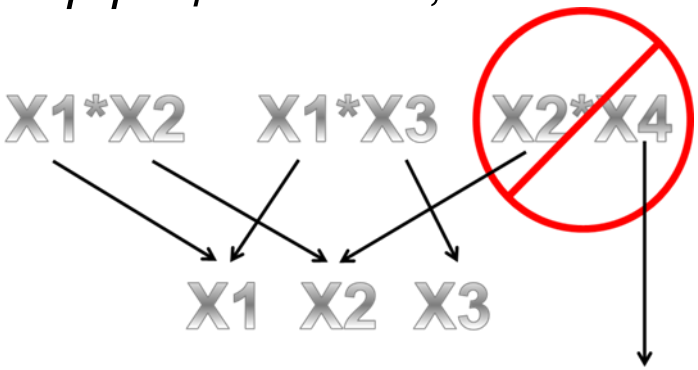
- Результаты в этой таблице показывают, что взаимосвязи важны.
- Величина *p*-value для члена TV*radio, отражающего взаимосвязь, чрезвычайно мала, что свидетельствует о наличии достоверных доказательств для гипотезы $H_A : \beta_3 \neq 0$.
- R^2 для модели с учетом взаимосвязей составляет 96.8%, по сравнению с только лишь 89.7% для модели, которая прогнозирует значение sales, используя значения TV и radio без учета взаимосвязей между ними.
- Это означает, что (96.8 – 89.7)/(100 – 89.7) = 69% дисперсии для sales, которая остается после построения аддитивной модели, объясняется членом, отражающим взаимосвязь.

	Coefficient	Std. Error	t-statistic	p-value
Intercept	6.7502	0.248	27.23	< 0.0001
TV	0.0191	0.002	12.70	< 0.0001
radio	0.0289	0.009	3.24	0.0014
TV×radio	0.0011	0.000	20.73	< 0.0001

Иерархия

- Иногда может иметь место ситуация, что член, отражающий взаимосвязь, имеет очень маленькое *p*-value, но связанные с ним базовые признаки (в примере, TV и radio) не проявляют аналогичные свойства.
- Принцип иерархии:

  Если мы включаем взаимосвязь в модель, мы должны также включать базовые признаки, даже если значения p-value, связанные с их коэффициентами, не показывают значимость.



Мотивация – улучшение интерпретируемости

Пример

	fixed acidity	volatile acidity	citric acid	sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	target_quality
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6

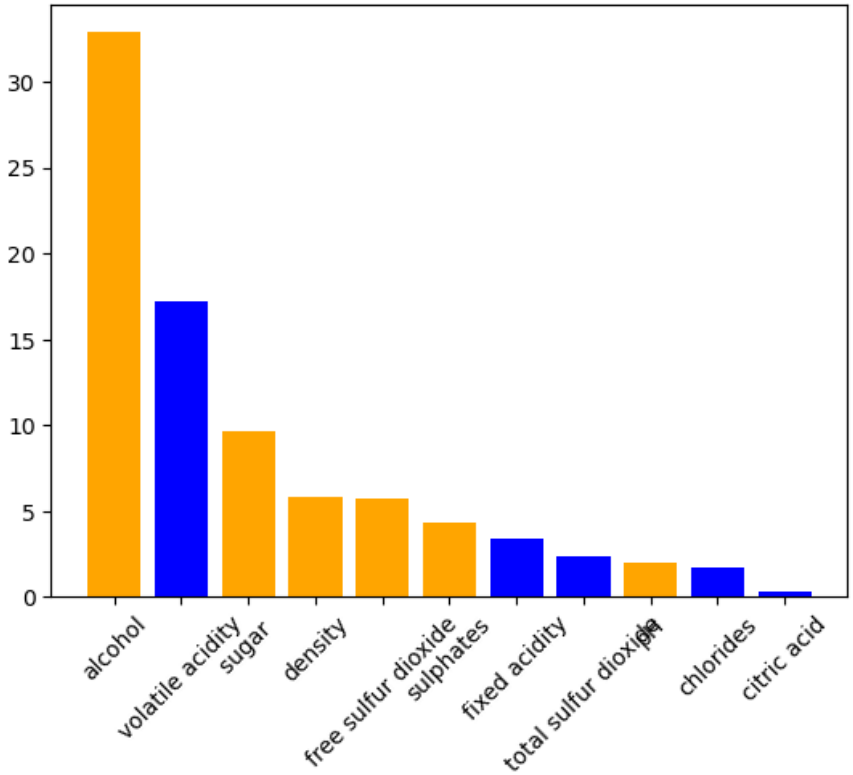
```python
import statsmodels.api as sm

y = df["target_quality"]
X = df[set(df.columns) - {"target_quality"}]

results = sm.OLS(y, X).fit()
results.summary()
```

Dep. Variable:	target_quality	**R-squared (uncentered):**	0.984
Model:	OLS	**Adj. R-squared (uncentered):**	0.984
Method:	Least Squares	**F-statistic:**	2.707e+04
Date:	Sun, 05 Mar 2023	**Prob (F-statistic):**	0.00
Time:	08:08:45	**Log-Likelihood:**	-5575.5
No. Observations:	4898	**AIC:**	1.117e+04
Df Residuals:	4887	**BIC:**	1.124e+04
Df Model:	11		

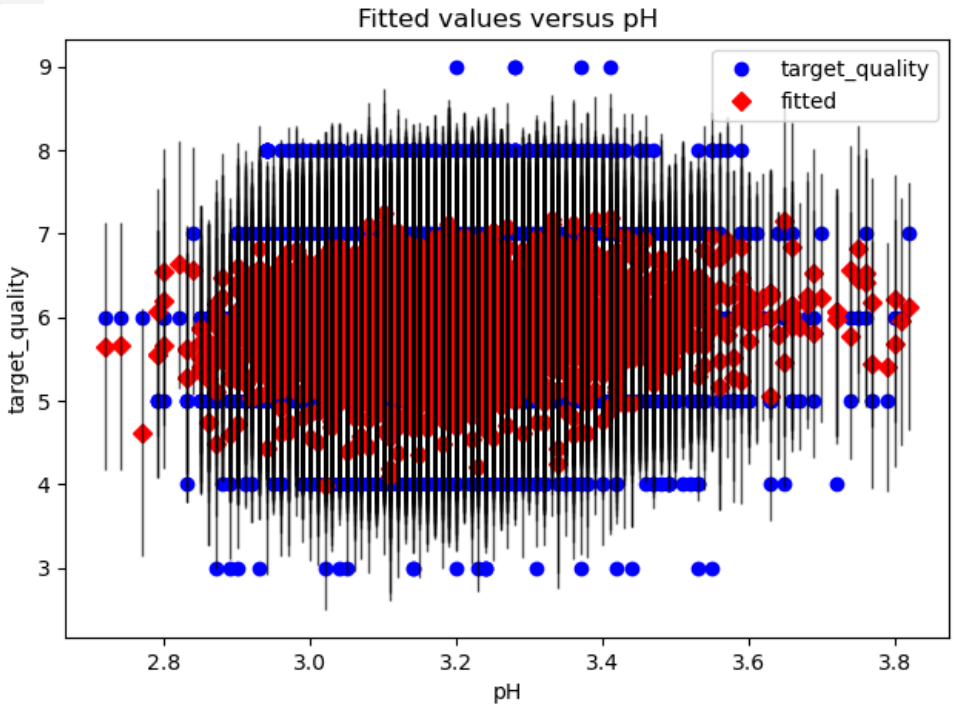
	coef	std err	t	P>|t|	[0.025	0.975]
sugar	0.0250	0.003	9.642	0.000	0.020	0.030
pH	0.1684	0.084	2.014	0.044	0.005	0.332
citric acid	-0.0293	0.096	-0.305	0.760	-0.218	0.159
volatile acidity	-1.9585	0.114	-17.196	0.000	-2.182	-1.735
density	2.0420	0.353	5.780	0.000	1.349	2.735
alcohol	0.3656	0.011	32.880	0.000	0.344	0.387
free sulfur dioxide	0.0048	0.001	5.710	0.000	0.003	0.006
sulphates	0.4165	0.097	4.279	0.000	0.226	0.607
chlorides	-0.9426	0.543	-1.736	0.083	-2.007	0.122
total sulfur dioxide	-0.0009	0.000	-2.352	0.019	-0.002	-0.000
fixed acidity	-0.0506	0.015	-3.356	0.001	-0.080	-0.021

Пример

```
res = results.tvalues
sign = (res > 0).map({True:"orange", False:"blue"})
res = abs(res).sort_values()[::-1]
sign = sign[res.index].values
plt.bar(res.index, res.values, color=sign)
plt.xticks(rotation=45)
plt.show()
```




```
fig = sm.graphics.plot_fit(results, "pH")
fig.tight_layout(pad=1.0)
```



Пример с formula

```
import statsmodels.formula.api as smf
model1 = smf.ols(formula='target_quality ~ alcohol * pH', data=df)

res1 = model1.fit()
res1.summary()
```

model1

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	16.9769	2.194	7.739	0.000	12.676	21.277
alcohol	-1.1383	0.207	-5.491	0.000	-1.545	-0.732
pH	-4.5215	0.691	-6.547	0.000	-5.875	-3.168
alcohol:pH	0.4557	0.065	6.990	0.000	0.328	0.583

Dep. Variable:	target_quality	**R-squared:**	0.200
Model:	OLS	**Adj. R-squared:**	0.199
Method:	Least Squares	**F-statistic:**	407.6
No. Observations:	4898	**AIC:**	1.162e+04
Df Residuals:	4894	**BIC:**	1.165e+04
Df Model:	3		

```
model2 = smf.ols(formula='target_quality ~ alcohol + pH', data=df)

res2 = model2.fit()
res2.summary()
```

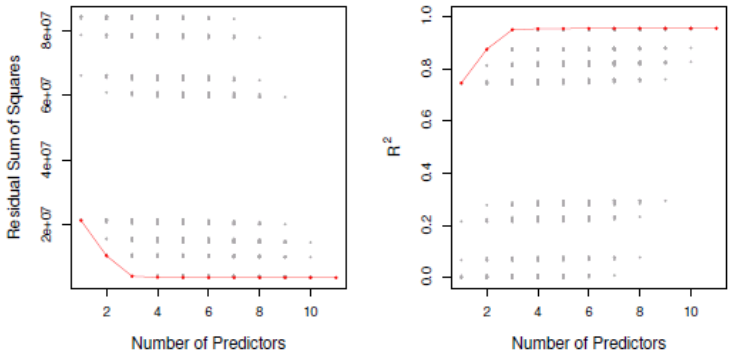
model2

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.7422	0.250	6.966	0.000	1.252	2.233
alcohol	0.3093	0.009	33.207	0.000	0.291	0.328
pH	0.2770	0.076	3.649	0.000	0.128	0.426

Dep. Variable:	target_quality	**R-squared:**	0.192
Model:	OLS	**Adj. R-squared:**	0.192
Method:	Least Squares	**F-statistic:**	581.3
No. Observations:	4898	**AIC:**	1.167e+04
Df Residuals:	4895	**BIC:**	1.169e+04
Df Model:	2		

Выбор важных переменных

- Наиболее очевидный подход называется регрессия *всех подмножеств* или регрессия *наилучших подмножеств* (МНК для всех комбинаций и выбор лучшего варианта по нектором критерию)

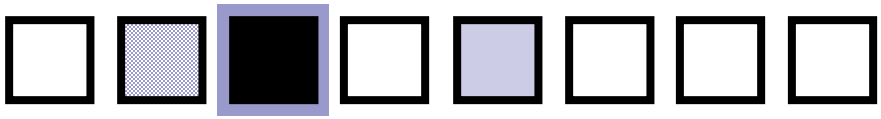


- На практие – не всегда применимо для значительного числа переменных, Поэтому существуют методы:
  1. Пошаговые методы (прямые, обратные, комбинированные)
  2. Методы с регуляризацией (штраф за сложность)
  3. Методы преобразования пространства признаков (регрессия главных компонент и PLS)

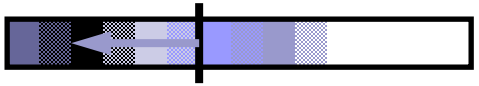
Прямой отбор

- Начинаем с *нулевой модели* содержащей только константу.
- Строим p простых линейных регрессий и добавляем к нулевой модели переменную, которая дает наименьшее значение RSS.
- Добавляем к этой модели переменную, которая дает наименьшее значение RSS среди всех моделей с двумя переменными.
- Продолжаем процесс до тех пор, пока не сработает какое-либо правило останова,
- Например, на каждом шаге рассчитывается p-value гипотезы о том, что улучшение целевой функции по сравнению с нулевой моделью существенно.

Input *p*-value

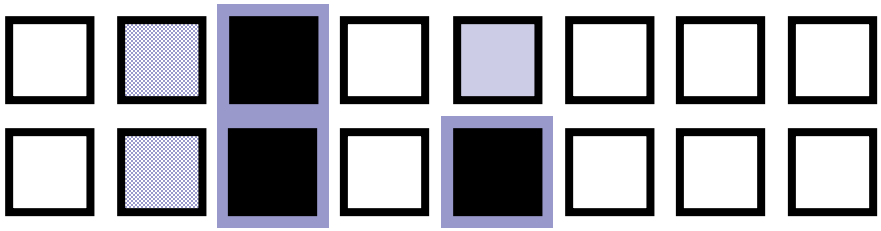


Entry Cutoff

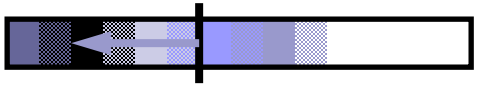


...

Input *p*-value

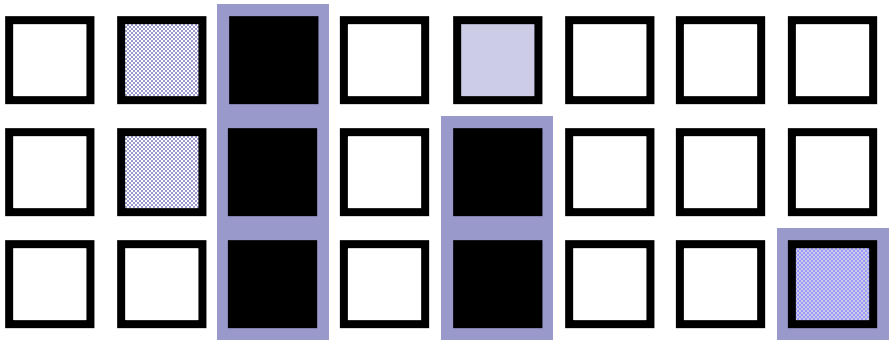


Entry Cutoff

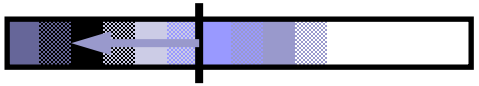


...

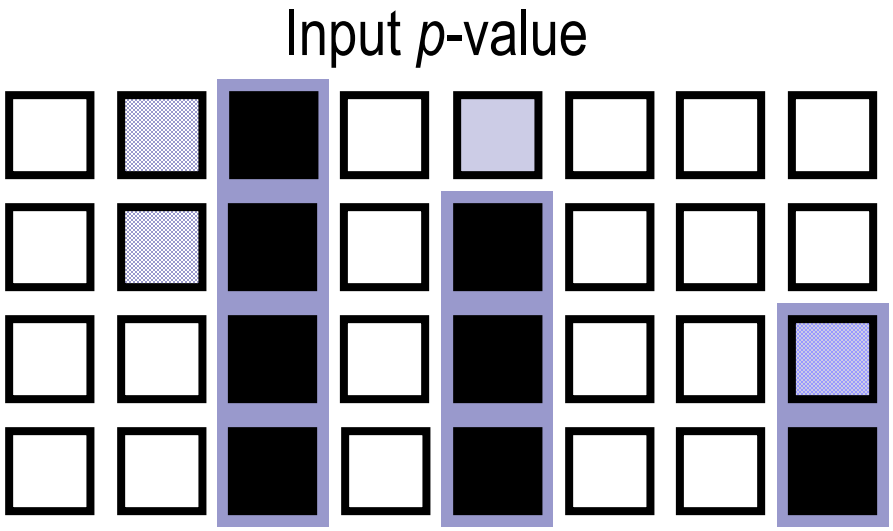
Input *p*-value

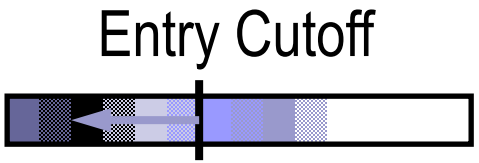


Entry Cutoff

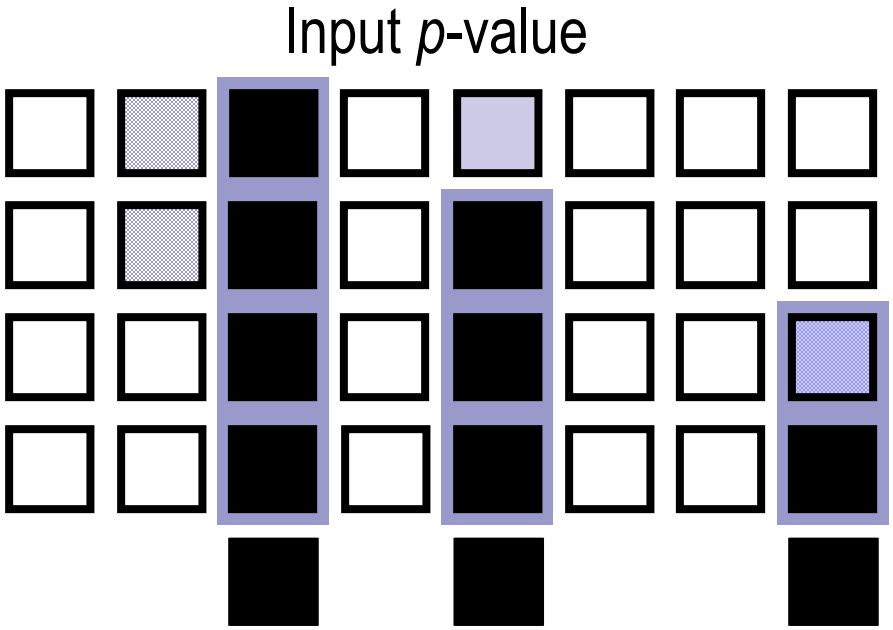


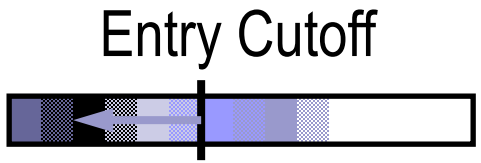
...


Input *p*-value


Entry Cutoff

...

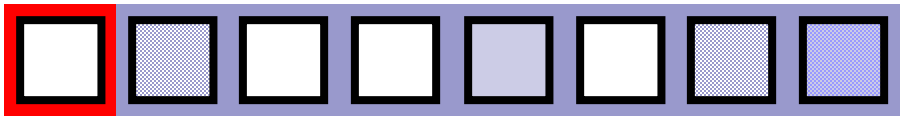

Input *p*-value


Entry Cutoff

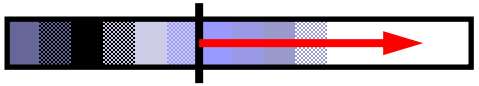
Обратный отбор

- Начинаем с модели, содержащей все переменные.
- Удаляем переменную с наибольшим *p*-value, то есть переменную, которая является наименее статистически значимой.
- Строим новую модель с (*p-1*)-ой переменными и удаляем переменную с наибольшим *p*-value.
- Продолжаем до тех пор, пока не сработает правило останова.
- Например, на каждом шаге рассчитывается *p*-value гипотезы о том, что ухудшение целевой функции по сравнению с нулевой моделью существенно.

Input *p*-value

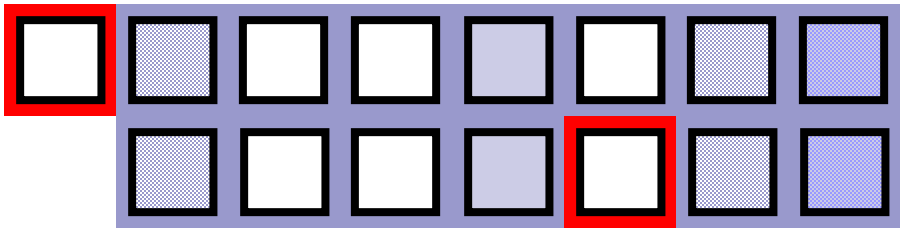


Stay Cutoff

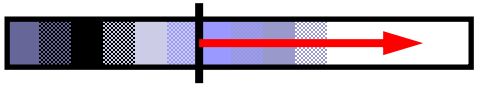


...

Input *p*-value

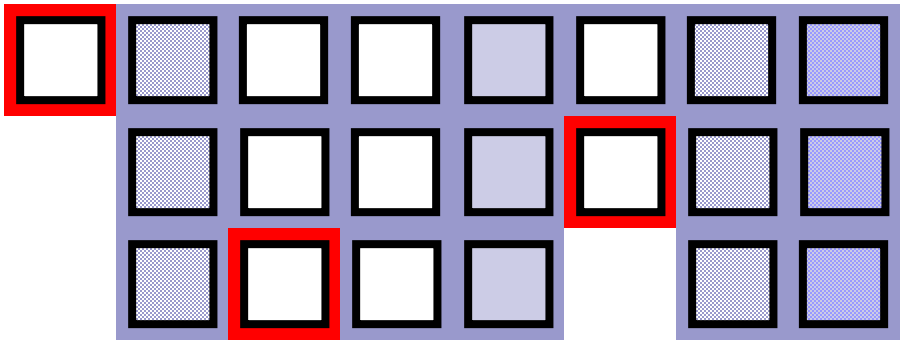


Stay Cutoff

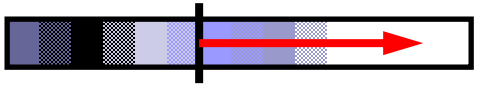


...

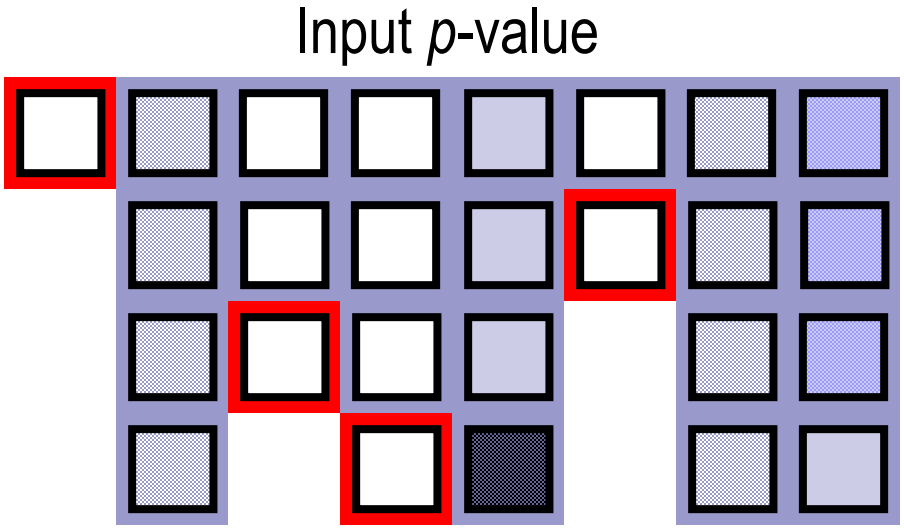
Input *p*-value

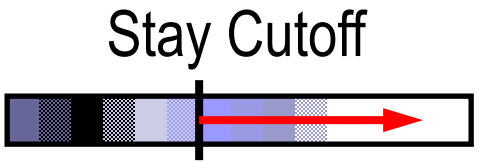


Stay Cutoff

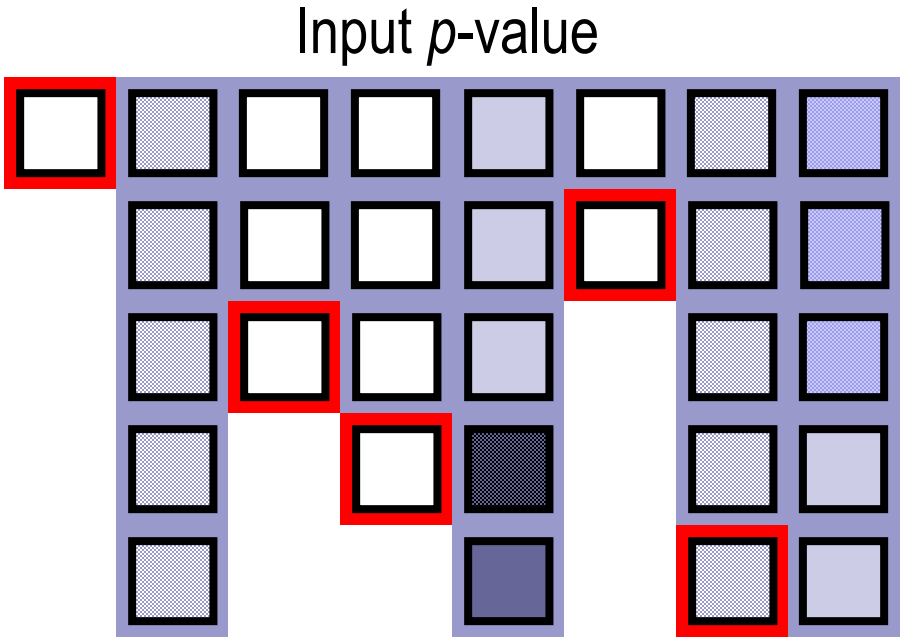


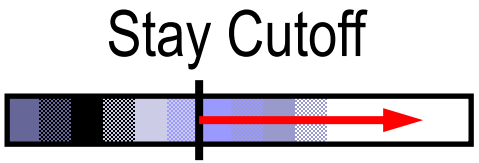
...


Input *p*-value

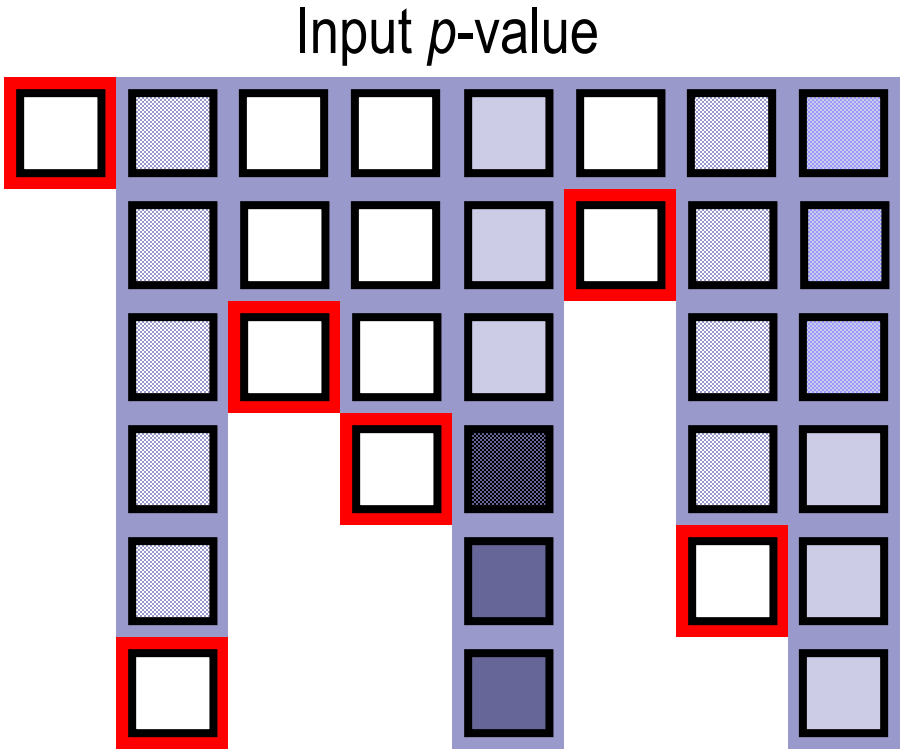

Stay Cutoff

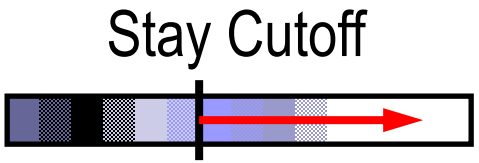
...


Input *p*-value

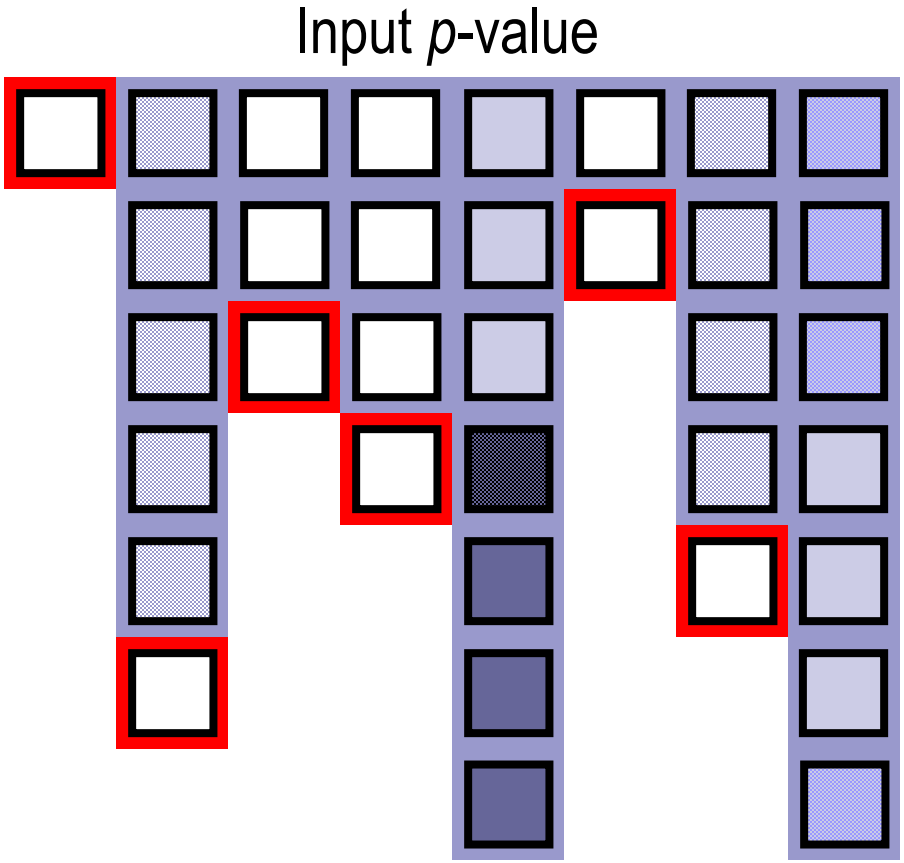

Stay Cutoff

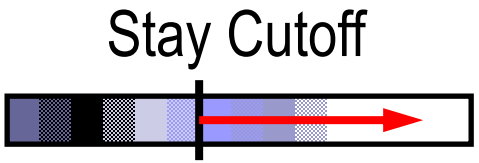
...


Input *p*-value

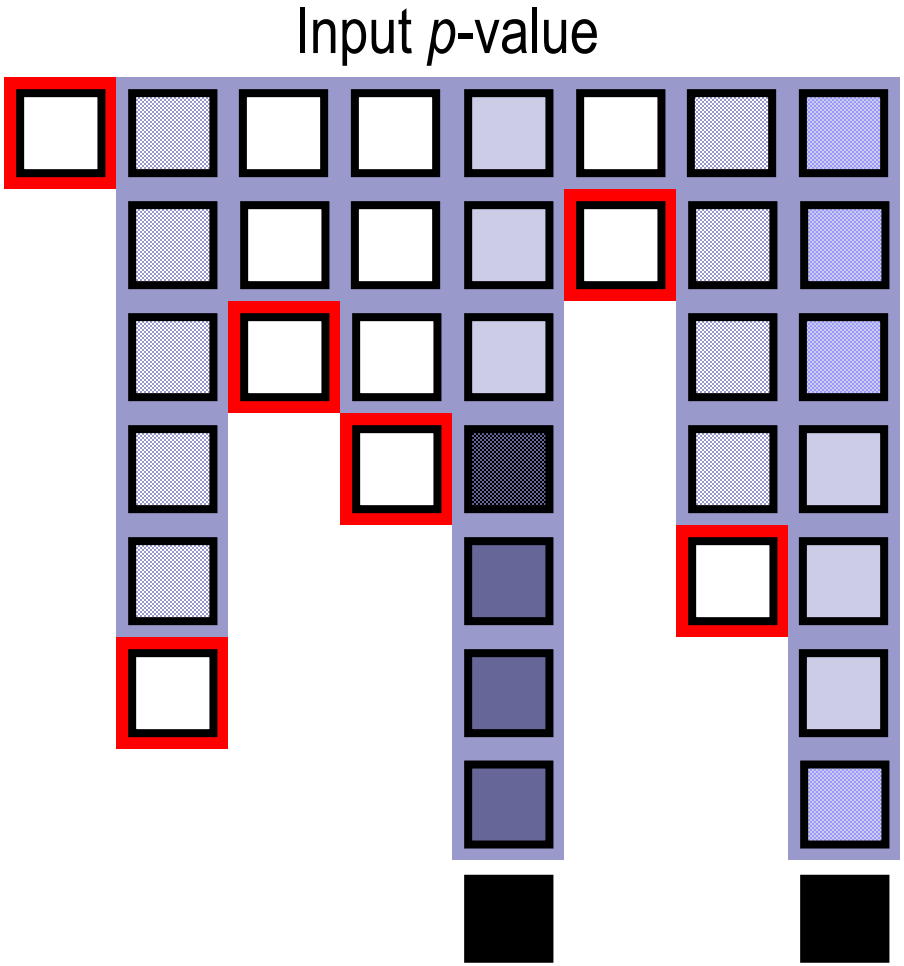

Stay Cutoff

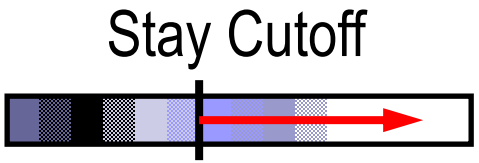
...


Input *p*-value


Stay Cutoff

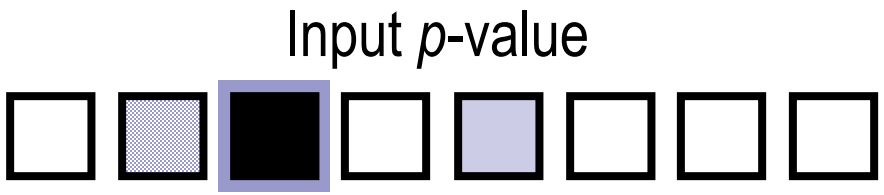
...

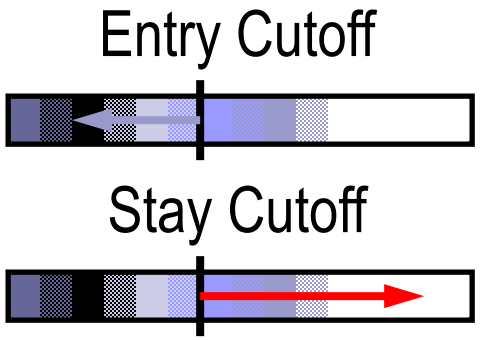

Input *p*-value


Stay Cutoff

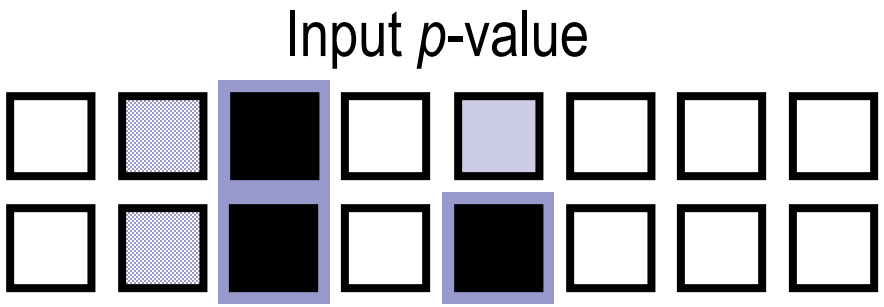
Комбинированный отбор

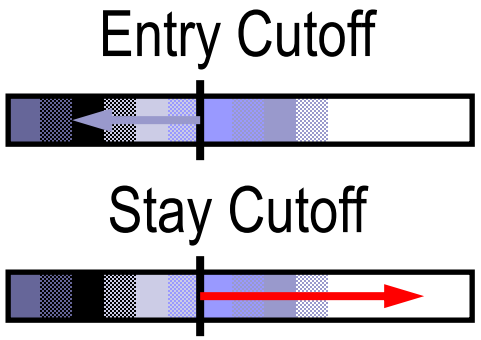
- Прямой и обратный методы – жадные (как и комбинированный) с не сильно гибким перебором
- Комбинированный метод (более гибкий перебор):
  1. Пытаемся сделать шаг вперед (сначала из нулевой модели)
  2. Затем пытаемся сделать шаг назад
  3. Продолжаем 1 и 2 до тех пор, пока не сработает правило останова и для добавления и для удаления переменных или пока не попали в «цикл» (последоватльное добавление и удаление одной и той же переменной)


Input *p*-value

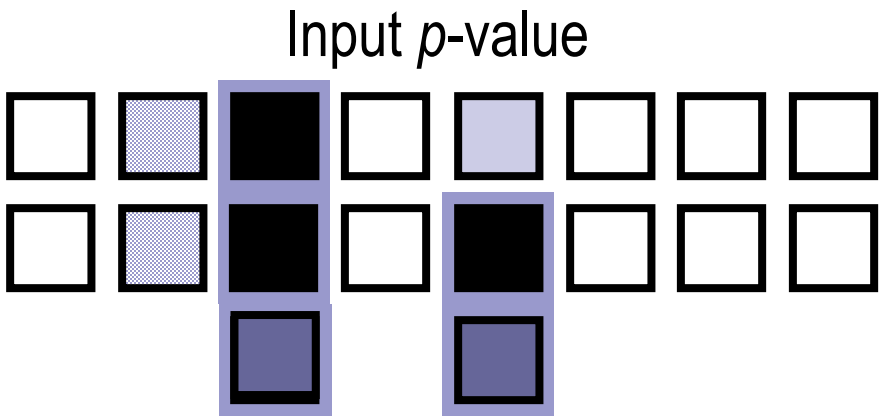

Entry Cutoff
Stay Cutoff

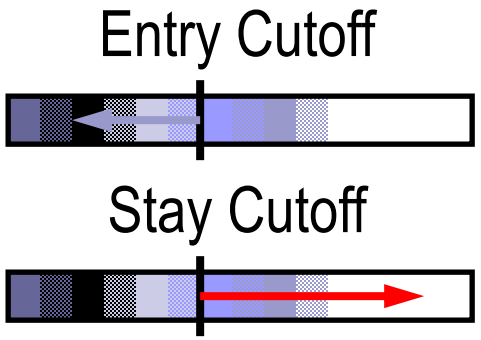
...


Input *p*-value

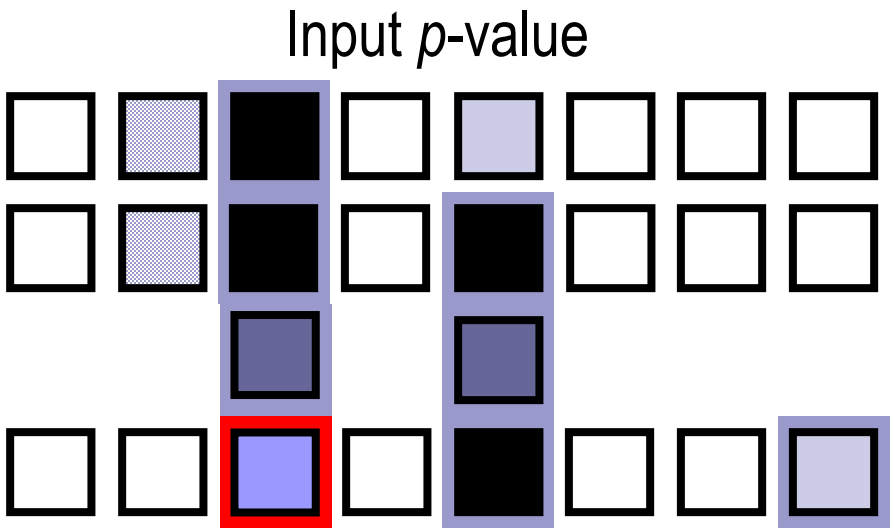

Entry Cutoff
Stay Cutoff

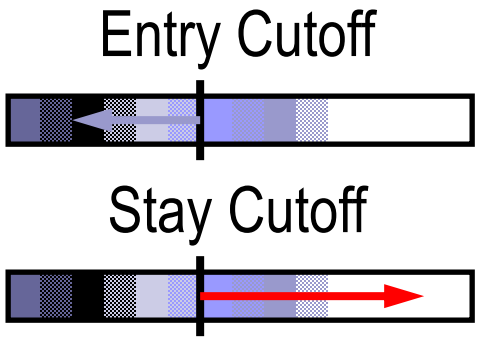
...


Input *p*-value

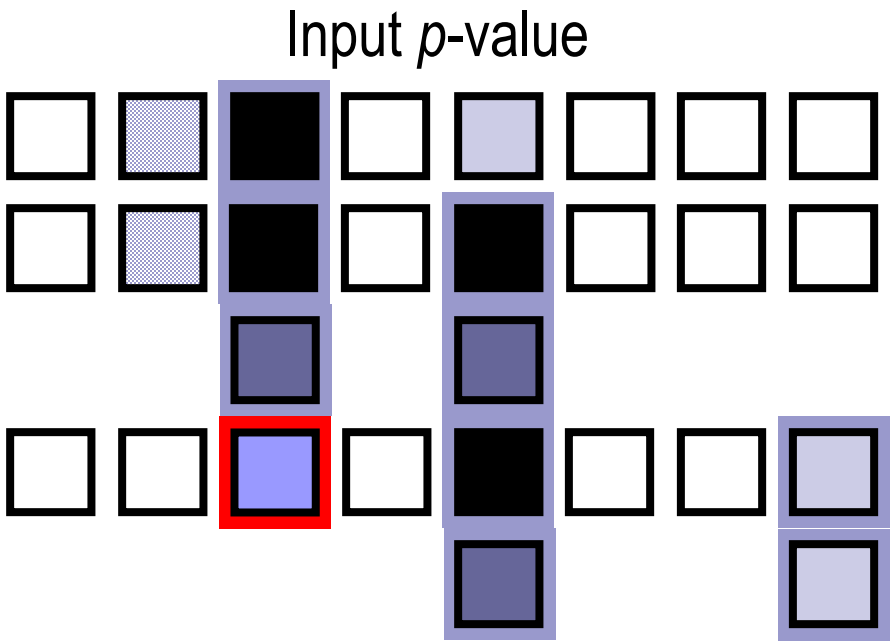

Entry Cutoff
Stay Cutoff

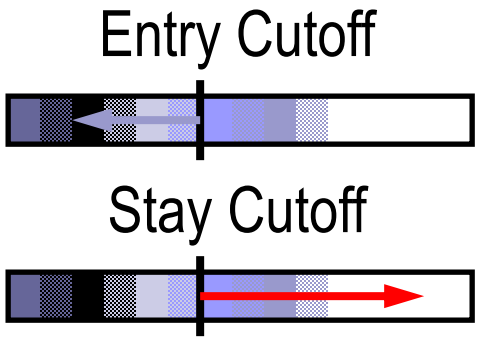
...


Input *p*-value

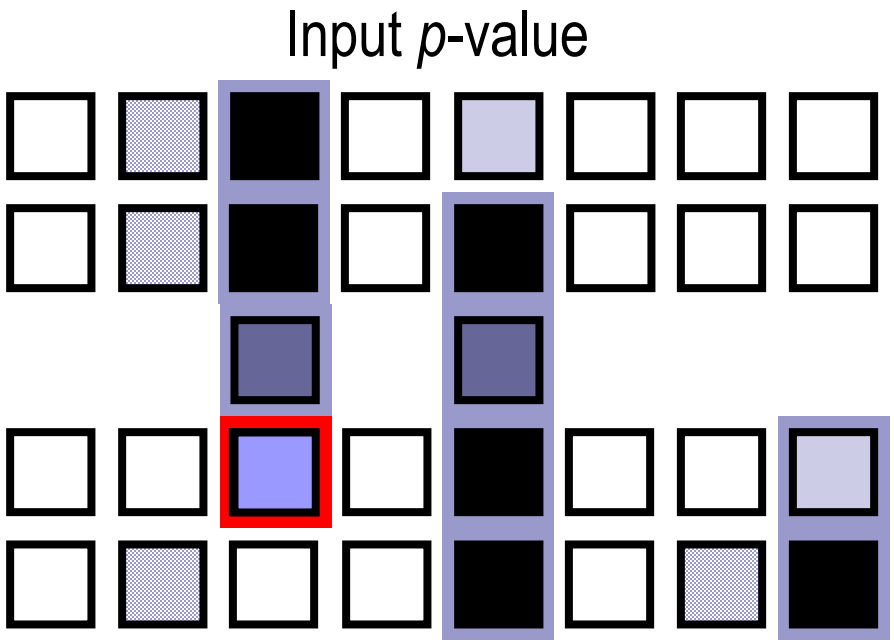

Entry Cutoff
Stay Cutoff

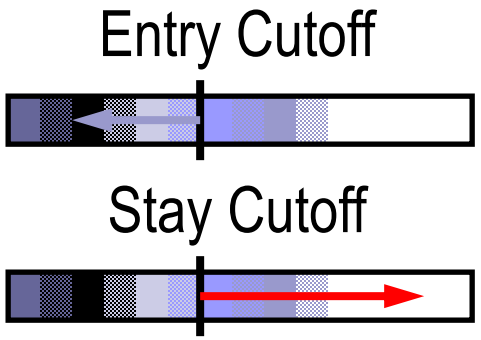
...


Input *p*-value

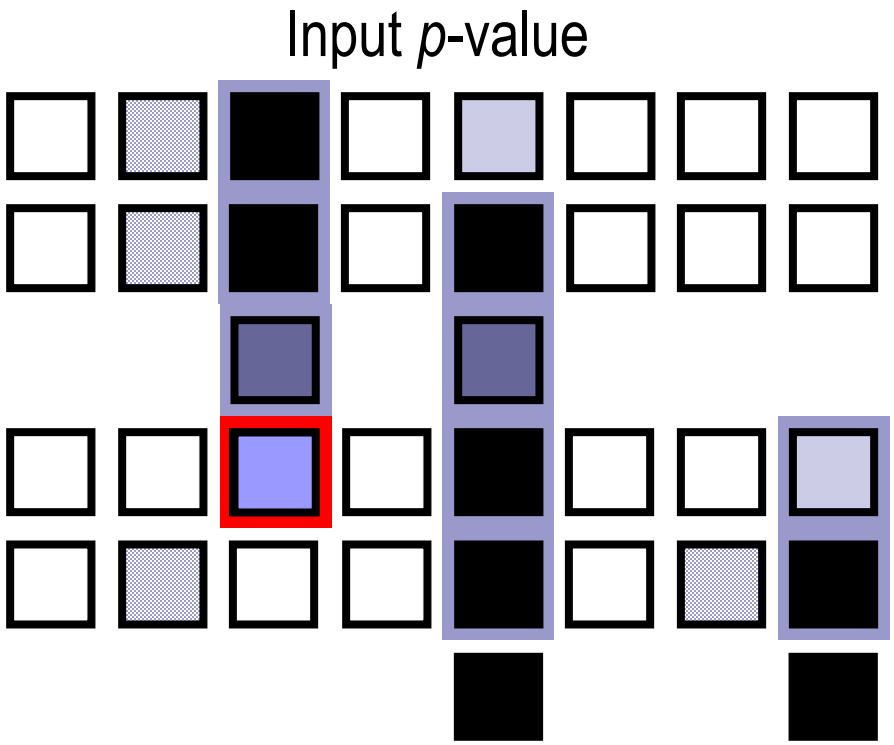

Entry Cutoff
Stay Cutoff

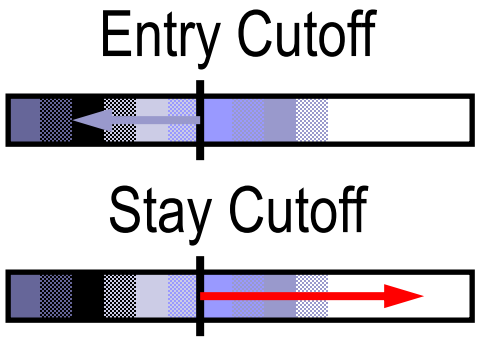
...


Input *p*-value


Entry Cutoff
Stay Cutoff

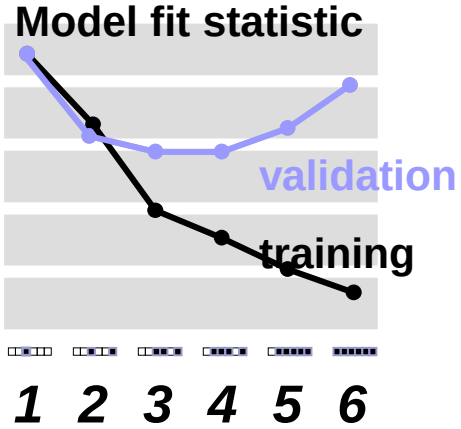
...

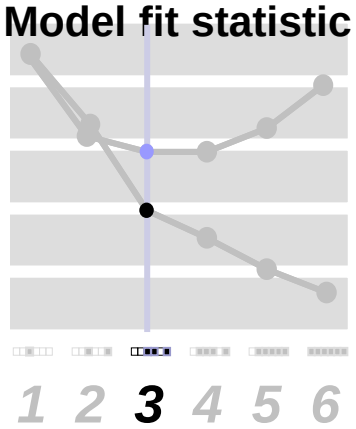

Input *p*-value


Entry Cutoff
Stay Cutoff

Пошаговая регрессия

1. Пусть M_0 – *начальная* модель, которая не содержит предикторов.
2. Для $k = 0, \ldots, p-1$: все $p-k$ моделей, которые дают улучшение по сравнению с предикторами в M_k с добавлением одного дополнительного предиктора.





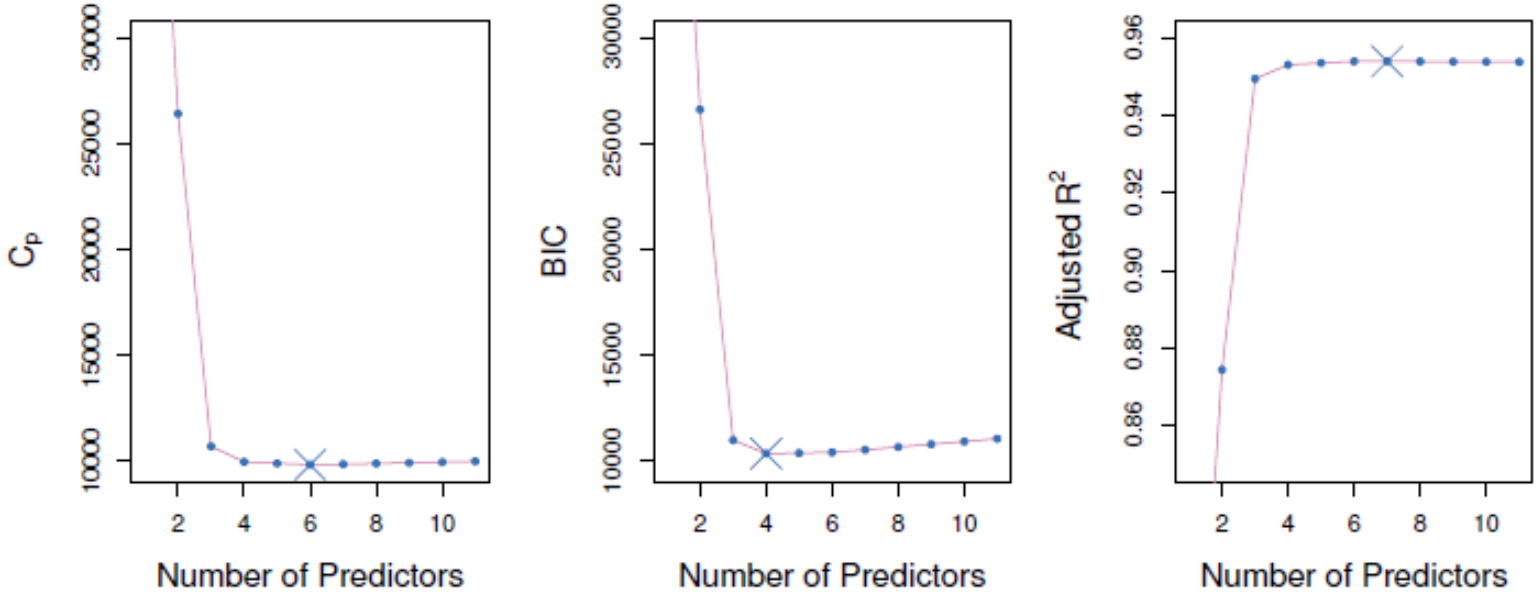
3. Выбираем одну наилучшую модель среди $M_0, \ldots, M_p$, используя валидационную или кросс-валидированную ошибку прогнозирования, C_p (AIC), BIC или скорректированный коэффициент детерминации R^2.

Выбор оптимальной модели

- Модель, содержащая все предикторы, всегда будет иметь наименьшее RSS и наибольший R^2, так как эти величины связаны с ошибкой обучения.
- Нужно выбрать модель с низкой тестовой ошибкой, а не с малой ошибкой обучения. Таким образом, RSS и R^2 не подходят для выбора лучшей модели среди набора моделей с различным количеством предикторов.
- Мы можем косвенно оценить ошибку тестирования, делая *поправку* на ошибку обучения, чтобы учесть смещения из-за *переобучения*.
- Мы можем *непосредственно* оценить погрешность тестирования, используя либо подход валидационного набора, либо подход кросс-валидации.

C_p, AIC, BIC и скорректированный коэффициент детерминации R^2

- Эти методы настраивают ошибку обучения для размера модели, и могут быть использованы для выбора среди множества моделей с различным числом переменных.
- На рисунке показаны C_p, BIC, и скорректированный R^2 для лучшей модели каждого размера.



Некоторые подробности

- *Mallow* C_p:

$$C_p = \frac{1}{n}\left(\mathrm{RSS} + 2d\hat{\sigma}^2\right),$$

- где *d* – обобщенное значение используемых параметров и $\hat{\sigma}^2$ - оценки дисперсии ошибки *ε*, связанной с каждым измерением отклика.
- Критерий *AIC*, определяемый для широкого класса моделей, рассчитывается методом максимального правдоподобия:

$$AIC = -2 \log L + 2d$$

где *L* - значение функции логарифмического правдоподобия для оцениваемой модели.

- В случае линейной модели с гауссовскими ошибками, максимальное правдоподобие и наименьшие квадраты - это одно и то же, и C_p и AIC эквивалентны.

BIC

$$\mathrm{BIC} = \frac{1}{n}\left(\mathrm{RSS} + \log(n) d\hat{\sigma}^2\right).$$

- Аналогично C_p, BIC будет небольшой для моделей с низкой ошибкой тестирования, и поэтому мы выбираем модель, которая имеет самое низкое значение BIC .
- Обратите внимание на то, что BIC заменяет $2d\hat{\sigma}^2$, используемое C_p ,на $\log(n)d\hat{\sigma}^2$, где *n* - число наблюдений.
- Так как *log n > 2* для любого *n > 7*, BIC статистики в целом сильнее штрафует модели со многими переменными, и, следовательно, приводит к выбору модели меньшего размера, чем C_p

Вычисление R^2

- Для модели наименьших квадратов с *d* переменными скорректированная R^2 статистика рассчитывается как:

$$\text{Adjusted } R^2 = 1 - \frac{\text{RSS}/(n-d-1)}{\text{TSS}/(n-1)}.$$

  где TSS – полная сумма квадратов.
- В отличие от C_p, AIC и BIC, для которых *малое* значение указывает на модель с низкой ошибкой тестирования, *большое* значение скорректированного R^2 соответствует модели с небольшой ошибкой тестирования.
- В отличие от статистики R^2, скорректированная R^2 статистика наказывает за включение ненужных переменных в модель.

Пример forward R^2

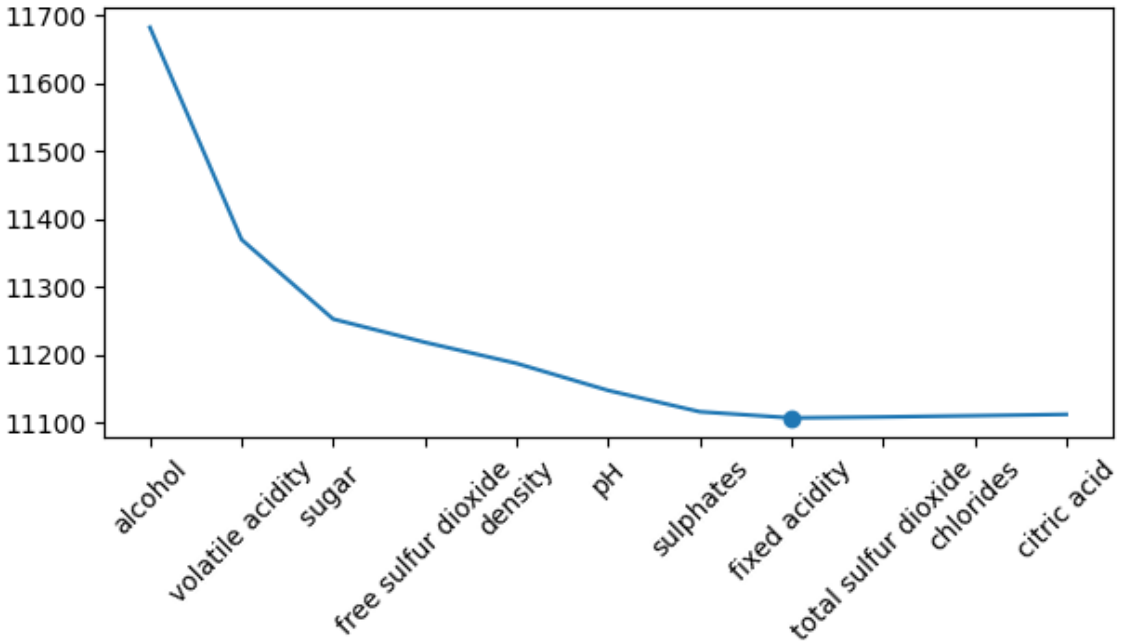
```python
def next_possible_feature(X, y, current_features,
                          by="rsquared", asc=False):
    feat_dict = {'feat': [], by:[]}
    for col in X.columns:
        if col not in (current_features):
            sub_X = X[current_features + [col]]
            sub_X = sm.add_constant(sub_X)
            model = sm.OLS(y, sub_X).fit()
            metric = getattr(model, by)
            feat_dict['feat'].append(col)
            feat_dict[by].append(metric)
    feat_df = pd.DataFrame(feat_dict)
    feat_df = feat_df.sort_values(by=[by],ascending=asc)
    best = feat_df.iloc[0].to_dict()
    return (best['feat'], best[by])
```

```python
# r^2
curr_feats = []
for i in range(len(X.columns)):
    print("="*10, "iteration:", i, "="*10)
    col, val = next_possible_feature(X, y, curr_feats,
                                     "rsquared", False)
    print("+", col, "->", val)
    curr_feats.append(col)
```

```
========== iteration: 0 ==========
+ alcohol -> 0.18972533274925663
========== iteration: 1 ==========
+ volatile acidity -> 0.2402311847533
========== iteration: 2 ==========
+ sugar  -> 0.25852615806597845
========== iteration: 3 ==========
+ free sulfur dioxide -> 0.2639942210
========== iteration: 4 ==========
+ density -> 0.2689515645655992
========== iteration: 5 ==========
+ pH -> 0.2751821150463122
========== iteration: 6 ==========
+ sulphates -> 0.2801195827165033
========== iteration: 7 ==========
+ fixed acidity -> 0.2817519637249508
========== iteration: 8 ==========
+ total sulfur dioxide -> 0.281835337
========== iteration: 9 ==========
+ chlorides -> 0.28186254437932134
========== iteration: 10 ==========
+ citric acid -> 0.2818703641332855
```

Пример forward AIC

```
# AIC
curr_feats = []
vals = []
for i in range(len(X.columns)):
    col, val = next_possible_feature(X, y, curr_feats,
                                     "aic", True)
    curr_feats.append(col)
    vals.append(val)

ind_min = np.argmin(vals)
```



Валидация и кросс-валидация

- Каждая из процедур возвращает последовательность моделей M_k, индексированная по размеру модели $k = 0, 1, 2, \ldots$. Наша задача здесь состоит в выборе $\hat{k}$. После выбора, мы определим модель $\mathcal{M}_{\hat{k}}$
- Вычислим ошибку на валидационном наборе или ошибку перекрестной проверки для каждой модели M_k, а затем выберем k , для которого полученная ошибка тестирования является наименьшей.
- Эта процедура имеет преимущество по сравнению с AIC, BIC, C_p и скорректированным R^2, которое состоит в том, что оно обеспечивает непосредственную оценку тестовой ошибки, и *не требует оценки дисперсии ошибки*
- Она также может быть использована в более широком диапазоне задач выбора модели, даже в случаях, когда трудно точно определить число степеней свободы модели (например количество предикторов в модели) или трудно оценить дисперсию ошибок

Пример SequentialFeatureSelector

```
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn import linear_model

lm = linear_model.LinearRegression()
sfs = SequentialFeatureSelector(lm,
        n_features_to_select=4,
        direction="forward",
        cv=3)
sfs.fit(X, y)
```

```
sfs.get_support()
```

```
array([False, False,  True, False,  True, False, False,  True,  True,
       False, False])
```

```
sfs.get_feature_names_out()
```

```
array(['alcohol', 'volatile acidity', 'sulphates', 'sugar '], dtype=object)
```

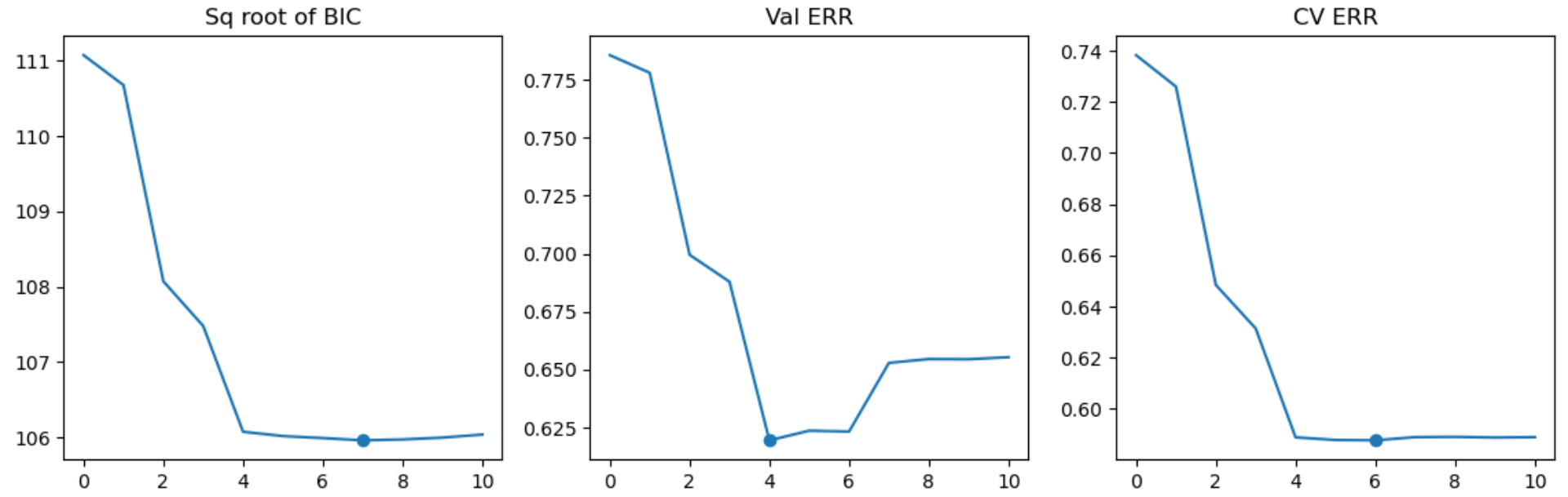
Пример

```python
# BIC
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error

def get_full_bic(X, y):
    model = sm.OLS(y, X).fit()
    return model.bic

def get_folds_error(X, y, folds):
    kf = StratifiedKFold(n_splits=folds)
    res_folds = []
    for tr_ind, tst_ind in kf.split(X, y):
        model = sm.OLS(y[tr_ind],
                       X.iloc[tr_ind]).fit()
        y_pred = model.predict(X.iloc[tst_ind])
        error = mean_squared_error(y[tst_ind], y_pred)
        res_folds.append(error)
    return np.mean(res_folds)
```

```python
full_bic = []
val_err = []
cv_err = []

range_feats = range(len(curr_feats))
for i in range_feats:
    sub_X = X[curr_feats[:i+1]]
    full_bic.append(get_full_bic(sub_X, y))
    val_err.append(get_folds_error(sub_X, y, 2))
    cv_err.append(get_folds_error(sub_X, y, 5))
```

Пример

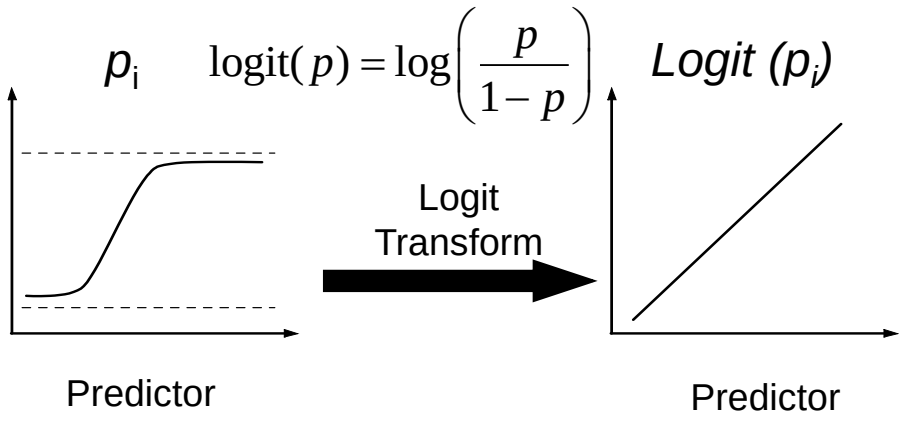
```
fig, axes = plt.subplots(ncols=3, figsize=(14, 4))

metr = [np.sqrt(full_bic), val_err, cv_err]
names = ["Sq root of BIC", "Val ERR", "CV ERR"]
for i, (l, n) in enumerate(zip(metr, names)):
    ind_min = np.argmin(l)
    axes[i].set_title(n)
    axes[i].plot(range_feats, l)
    axes[i].scatter(range_feats[ind_min], l[ind_min])
plt.show()
```



Если не нормальное распределение ошибки?

Функция связи → $g(E(y_i)) = F(\mathbf{X}, \beta)$

- Распределение отклика наблюдений принадлежит экспоненциальному семейству. $f(y \mid \theta) = h(y)c(\theta)\exp\left(t(y) \cdot W(\theta)\right)$
- Дисперсия зависимой переменной Y – функция от среднего.
- F(X,β) моделирует функцию от E(y) (link function – функция связи)
- Распределение отклика наблюдений может подсказать какую функцию связи выбрать (дальше)
- Пример (лоистическая регрессия):



$$f(y \mid p) = p^y (1-p)^{1-y} =$$
$$= (1-p) I_y \exp\left(y \log(p/(1-p))\right)$$

$$I_y = \begin{cases} 1, y \in \{0,1\} \\ 0, \textit{иначе} \end{cases}$$

Типовые функции связи для обобщенных линейных моделей

Model	Response	Distribution	Mean	Variance	Canonical Link
Linear Regression	Continuous	Normal	μ	σ^2	identity μ
Logistic regression	Dichotomous	Binomial	π	$\pi(1-\pi)/n$	logit $\log[\pi/(1-\pi)]$
Poisson Regression	Count	Poisson	λ	λ	log $\log(\lambda)$
Gamma Regression	Continuous	Gamma	μ	μ^2/ν	*inverse $1/\mu$

*часто используется функция связи LOG

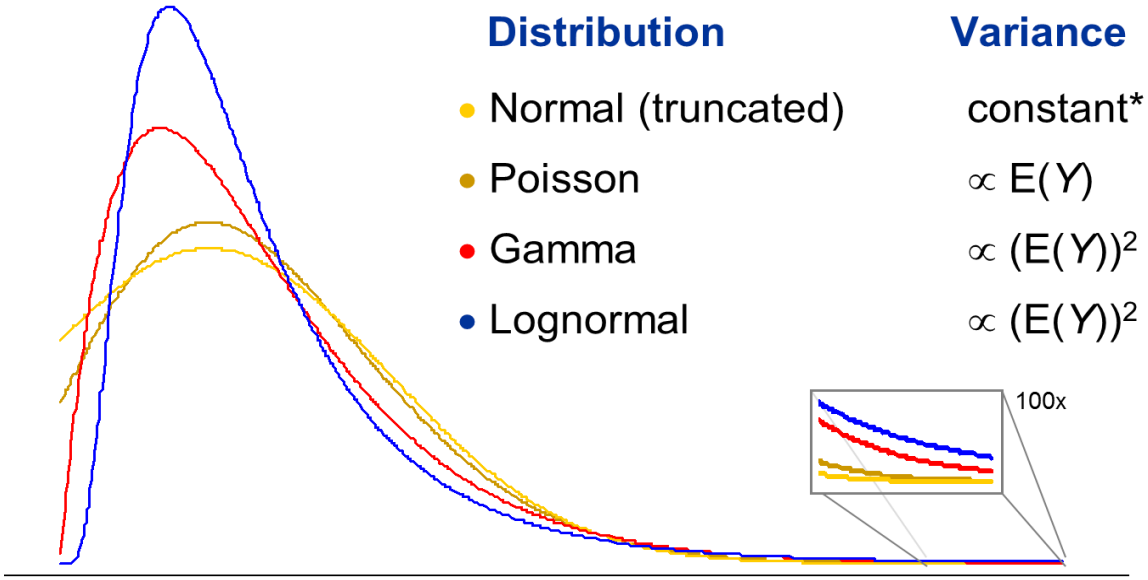
Оценка отклонения

Поиск параметров модели

- решается задача оптимизации
- max loglik с заданным распределением и функцией связи

Распределение	**Отклонение**
Normal	$Q(\mathbf{w}) = \sum (y - \mu(\mathbf{w}))^2$
Poisson	$Q(\mathbf{w}) = 2\sum \left[y \ln(y / \mu(\mathbf{w})) - (y - \mu(\mathbf{w})) \right]$
Gamma	$Q(\mathbf{w}) = 2\sum \left[-\ln(y / \mu(\mathbf{w})) + (y - \mu(\mathbf{w})) / \mu(\mathbf{w}) \right]$
Bernoulli	$Q(\mathbf{w}) = -2\sum \left[y \ln(\mu(\mathbf{w})) + (1 - y) \ln(1 - \mu(\mathbf{w})) \right]$

Выбор распределения для обощенной линейной модели



- В случае гетероскедастичности вместо линейной часто применяется гамма регрессия (с различными функциями связи)
- Гамма распределение:
  - Асиметричное распределение для **положительных** значений
  - Дисперсия пропорциональна квадрату среднего
  - Хвост «легче» чем у логнормального

Пример использования GLM

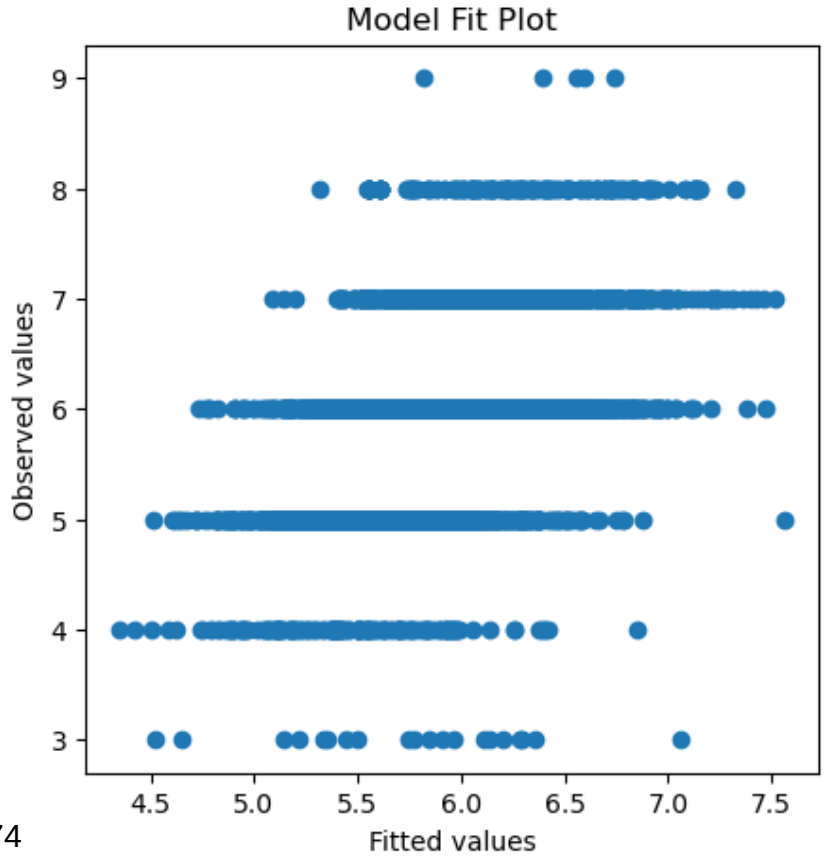
```
import statsmodels.api as sm

gamma_model = sm.GLM(y, X, family=sm.families.Gamma())
gamma_results = gamma_model.fit()
gamma_results.summary()
```

Dep. Variable:	target_quality	**No. Observations:**	4898
Model:	GLM	**Df Residuals:**	4887
Model Family:	Gamma	**Df Model:**	10
Link Function:	inverse_power	**Scale:**	0.016581
Method:	IRLS	**Log-Likelihood:**	-5595.9
Date:	Sun, 12 Mar 2023	**Deviance:**	83.215
Time:	20:31:55	**Pearson chi2:**	81.0
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.3134

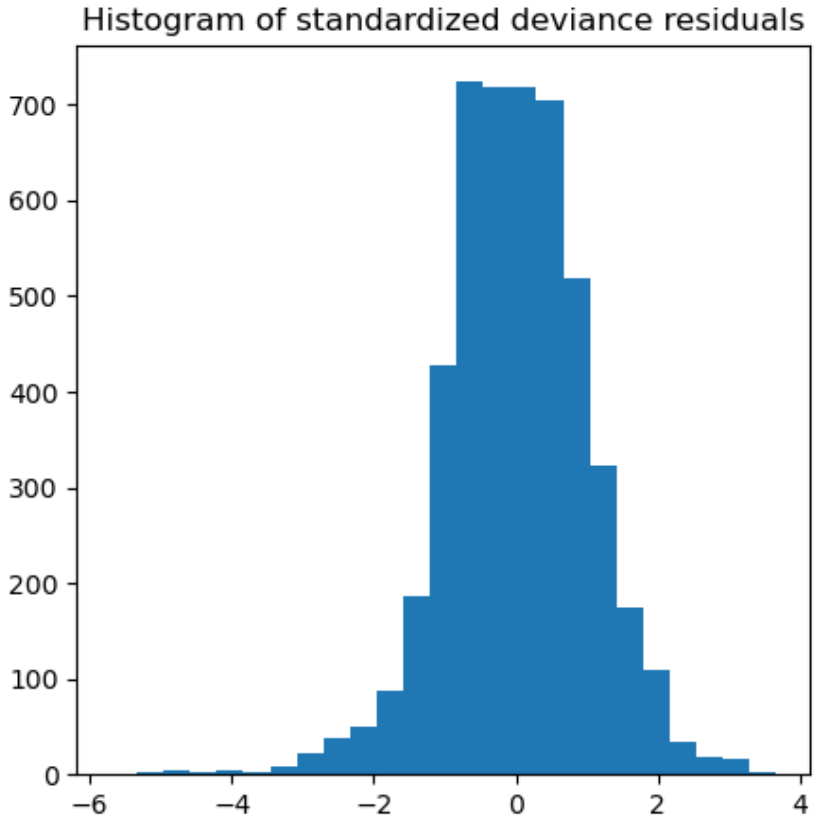
	coef	std err	z	P>|z|	[0.025	0.975]
free sulfur dioxide	-0.0001	2.41e-05	-5.598	0.000	-0.000	-8.76e-05
density	0.2860	0.010	27.808	0.000	0.266	0.306
alcohol	-0.0100	0.000	-31.482	0.000	-0.011	-0.009
pH	-0.0067	0.002	-2.768	0.006	-0.011	-0.002
volatile acidity	0.0618	0.003	17.836	0.000	0.055	0.069
total sulfur dioxide	2.369e-05	1.1e-05	2.162	0.031	2.21e-06	4.52e-05
fixed acidity	0.0010	0.000	2.372	0.018	0.000	0.002
sulphates	-0.0122	0.003	-4.496	0.000	-0.018	-0.007
sugar	-0.0009	7.57e-05	-11.640	0.000	-0.001	-0.001
chlorides	0.0367	0.017	2.149	0.032	0.003	0.070
citric acid	0.0012	0.003	0.404	0.686	-0.004	0.007

Пример использования GLM

```
fig, ax = plt.subplots()
yhat = gamma_results.predict(X)
ax.scatter(yhat, y)
```




```
fig, ax = plt.subplots()

resid = gamma_results.resid_deviance.copy()
resid_std = stats.zscore(resid)
ax.hist(resid_std, bins=25)
```



Методы регуляризации (штраф за сложность)

Регрессия Ridge и Lasso

- Методы выбора подмножества используют метод наименьших квадратов для линейной модели, которая содержит подмножество предикторов.
- В качестве альтернативы, можно построить модель содержащую все *p* предикторов с использованием методики, которая *ограничивает или регуляризирует* оценки коэффициентов, или, что эквивалентно, сводит некоторые оценки коэффициентов к нулю.
- Может быть не сразу понятно, почему такое ограничение должно улучшить построение модели, но оказывается, что сокращение количества коэффициентов может значительно уменьшить их дисперсию.

Гребневая регрессия

- Напомним, что процедура подгонки по методу наименьших квадратов оценивает коэффициенты $\beta_0, \beta_1, \ldots, \beta_p$, минимизируя:

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

- Оценки коэффициентов $\hat{\beta}^R$ ridge-регрессии напротив являются значениями, которые надо минимизировать

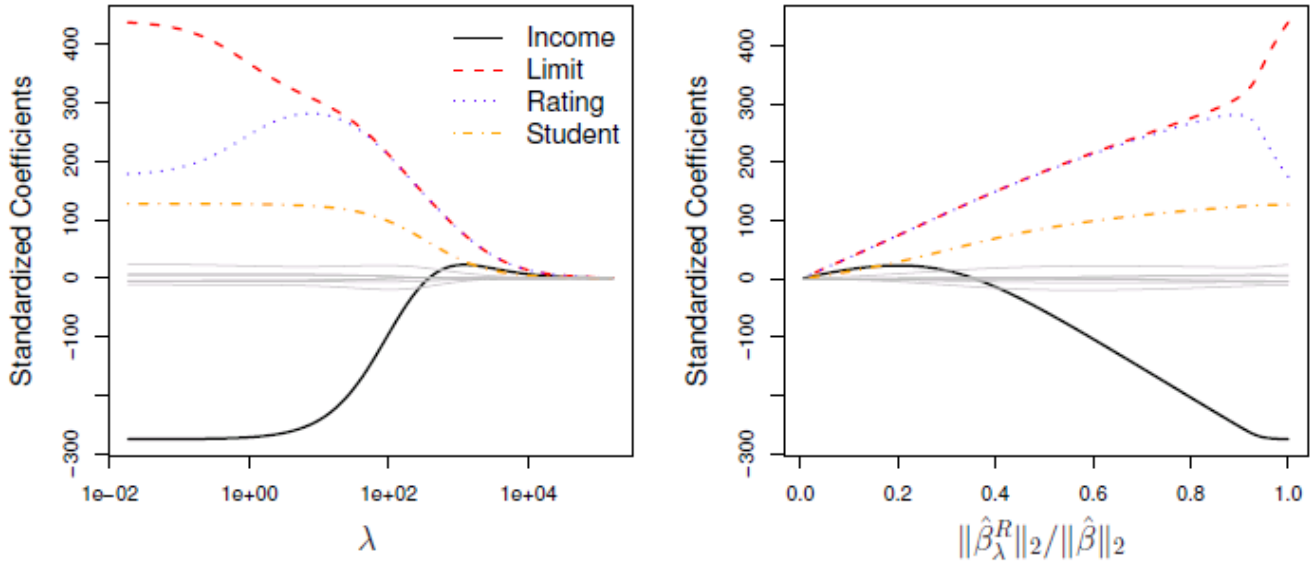
$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = \mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2,$$

где $\lambda \geq 0$ - параметр настройки, который задается (рассчитывается) независимо.

Штраф за сложность

Гребневая регрессия

- Гребневая регрессия (как и МНК) стремится найти коэффициенты, которые дают наименьшее RSS.
- Но, второй член, $\lambda \sum_j \beta_j^2$, называемый штрафом сокращения, мал при $\beta_1, \ldots, \beta_p$ близких к нулю, и поэтому имеет место эффект сведения оценок β_j к нулю. Параметр настройки λ (подбор кросс-валидацией) служит для управления относительным влиянием этих двух членов на оценки коэффициентов



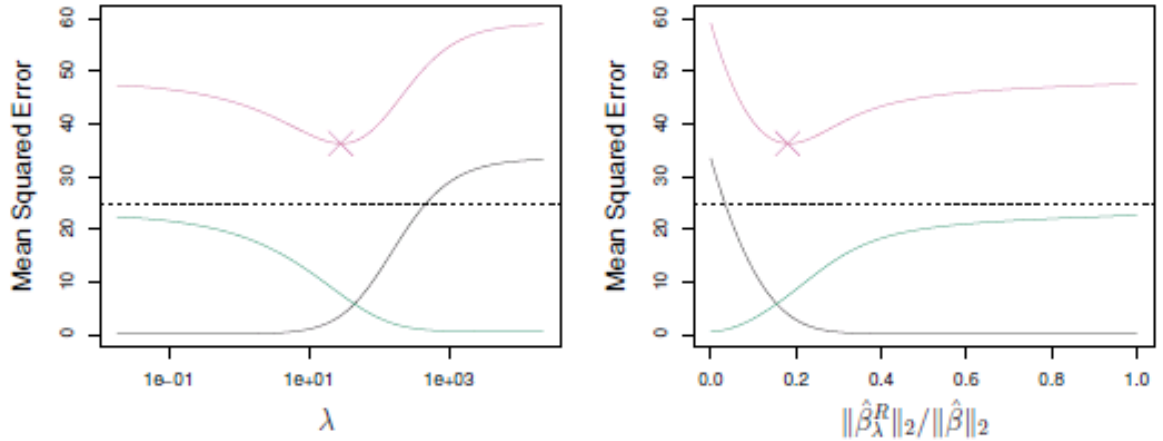
Гребневая регрессия: масштабирование предикторов

- Оценки коэффициентов стандартным методом наименьших квадратов являются масштабируемым: умножая X_j на константу *c* просто приводит к масштабированию оценок коэффициентов наименьших квадратов на коэффициент. Другими словами, независимо от того, как масштабируется *j*-ый предиктор, $X_j\hat{\beta}_j$ останется прежним.
- Оценки коэффициентов гребневой регрессии наоборот могут *существенно* измениться при умножении заданного предиктора на константу, из-за суммы квадратов коэффициентов в штрафной части целевой функции регрессии.
- Поэтому лучше всего применять гребневую-регрессию после нормирования коэффициентов на коэффициенты модели без регуляризации $\|\hat{\beta}_\lambda^R\|_2/\|\hat{\beta}\|_2$, или после *стандартизации предикторов:*

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \overline{x}_j)^2}}$$

Почему Ridge регрессия дает улучшения по сравнению с методом наименьших квадратов?

Компромисс отклонение-дисперсия



Смоделированные данные с $n = 50$ наблюдениями, $p = 45$ предикторами все имеют отличные от нуля коэффициенты. Квадратичное смещение (черное), дисперсия (зеленая) и среднеквадратичная ошибка тестирования (фиолетовая) для предикторов *ridge*-регрессии на смоделированном наборе данных, в зависимости от λ и $\|\hat{\beta}^R_\lambda\|_2/\|\hat{\beta}\|_2$. Горизонтальные пунктирные линии указывают на минимально возможное значение MSE.

Lasso

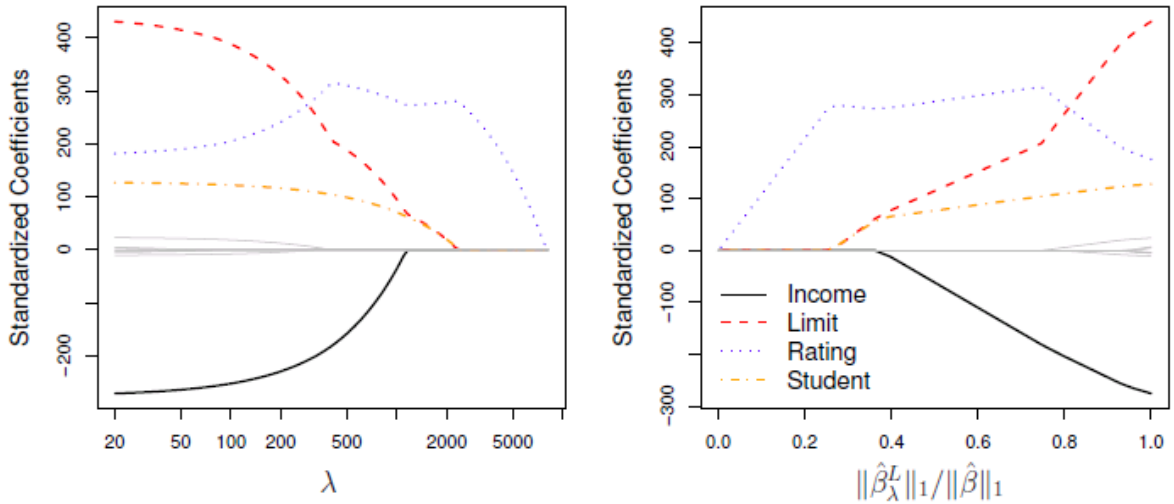
- Гребневая регрессия имеет один очевидный недостаток: в отличие от отбора подмножества, которое, как правило, выбирает модели, которые включают только подмножество переменных, гребневая регрессия будет включать в себя все *p* предикторов в конечной модели
- *Lasso* - относительно недавняя альтернатива, которая преодолевает этот недостаток. Коэффициенты lasso минимизируют величину

$$\underbrace{\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2}_{\ell_2}+\underbrace{\lambda\sum_{j=1}^{p}|\beta_j|}_{\ell_1}=\mathrm{RSS}+\underbrace{\lambda\sum_{j=1}^{p}|\beta_j|}_{\ell_1}.$$

Штраф за сложность

Lasso: продолжение

- В случае Lasso, штраф имеет эффект сведения некоторых оценок коэффициентов в точности к нулю, когда параметр настройки λ достаточно велик.
- Следовательно, так же, как выбор лучшего подмножества, lasso выполняет *отбор переменных*.
- lasso приводит к *разреженным* моделям - моделям, которые включают только подмножество переменных.
- Как и в ridge регрессии, выбирая хорошее значение λ для lasso имеет решающее значение



Выбор переменных в регрессии Lasso

Почему lasso, в отличие от ridge-регрессии, приводит к оценкам коэффициентов, которые в точности равны нулю?

Можно показать, что оценки коэффициентов lasso и ridge регрессии решают задачи

$$\underset{\beta}{\text{minimize}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

при условии $\sum_{j=1}^{p} |\beta_j| \leq s$

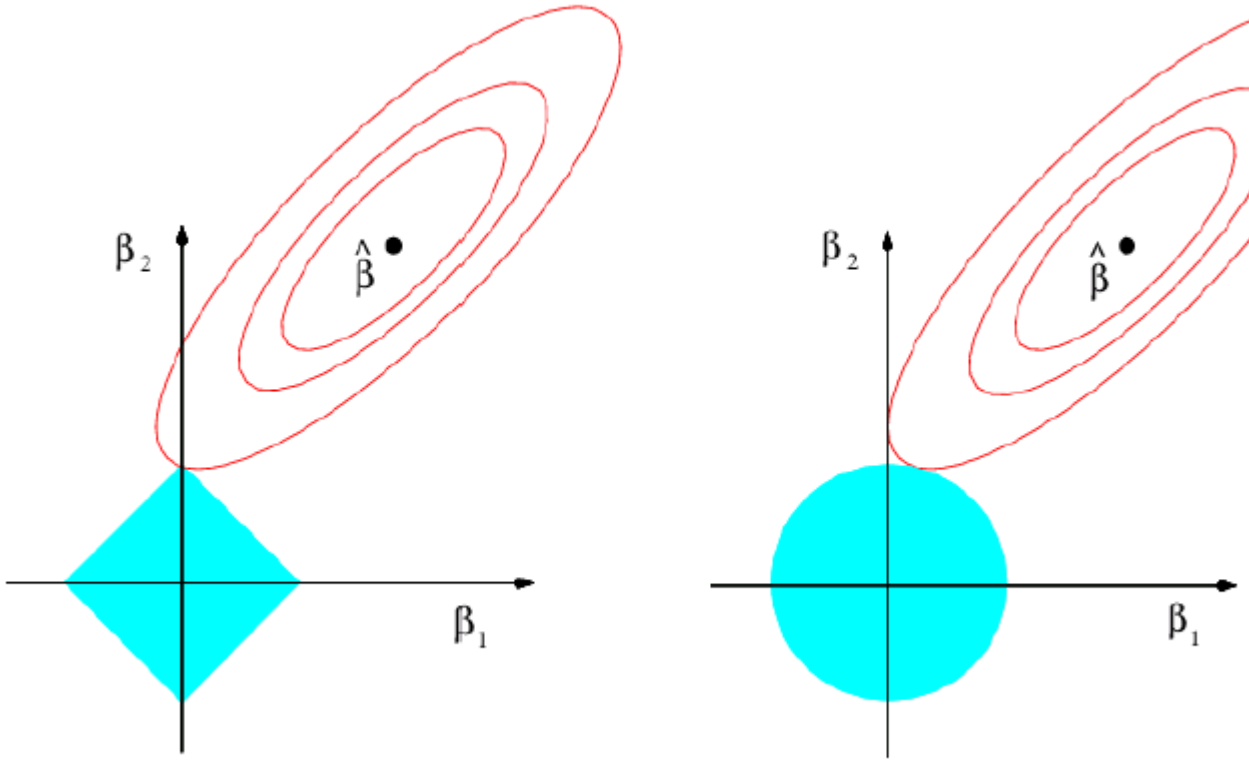
и

$$\underset{\beta}{\text{minimize}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

при условии $\sum_{j=1}^{p} \beta_j^2 \leq s,$

соответственно.

Иллюстрация регрессии Lasso

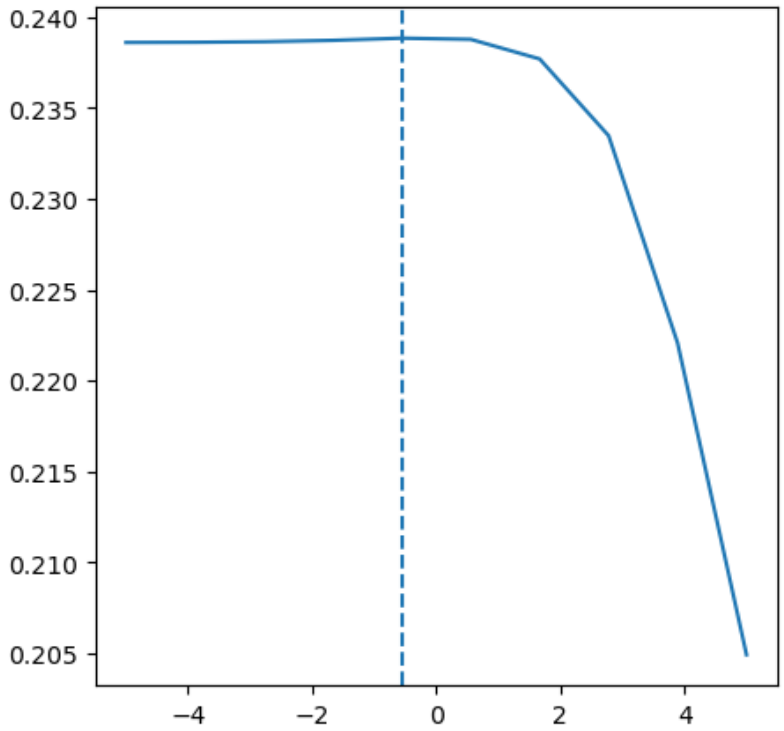


Выбор параметров настройки для Ridge регрессии и Lasso

- Что же касается выбора подмножества, для ridge-регрессии и lasso нам нужен способ определения, какая из рассматриваемых моделей лучше.
- То есть нам нужен метод выбора значения для параметра настройки λ или, что эквивалентно, значение s.
- *Перекрестная проверка* обеспечивает простой способ решения этой проблемы. Выберем сетку значений λ и вычислим ошибку кросс-валидации для каждого значения λ .
- Затем мы выбираем значение параметра настройки, для которого ошибка перекрестной проверки является наименьшей.
- И, наконец, модель перестраивается с использованием всех имеющихся объектов и выбранного значения параметра настройки.

Пример Ridge

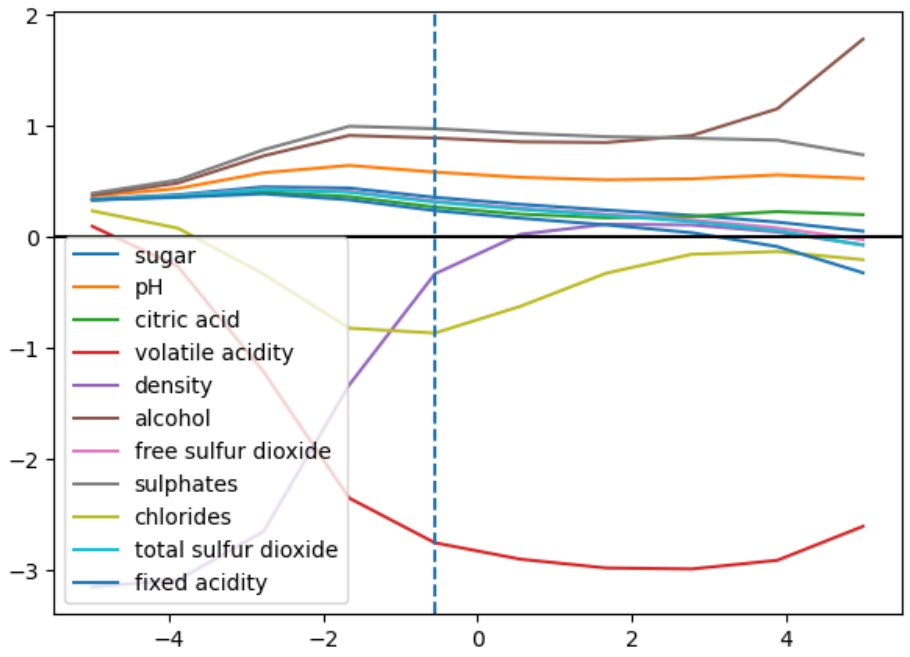
```
from sklearn.linear_model import Ridge

degree = np.linspace(-5, 5, 10)
alphas = np.exp(degree)
scores = []
for alpha in alphas:
    ridge = Ridge(alpha=alpha, fit_intercept=False)
    score = cross_val_score(ridge, X, y, cv=5,
                            scoring="r2")
    scores.append(np.mean(score))

plt.figure(figsize=(7, 5))
plt.plot(degree, scores)
best_score = np.argmax(scores)
plt.axvline(x=degree[best_score], linestyle="--")
plt.show()
```



Ошибки перекрестной проверки, которые являются результатом применения ridge регрессии для различных значений λ .

Пример Ridge

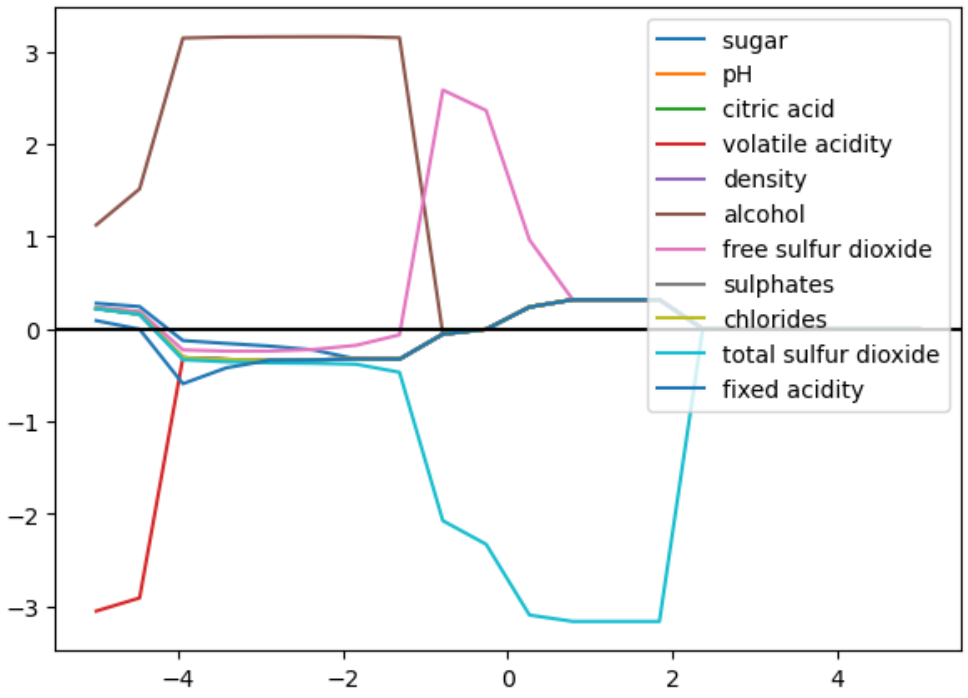
```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler

coefs = []
for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X, y)
    scaler = StandardScaler()
    coef = ridge.coef_.reshape(-1, 1)
    coef = scaler.fit_transform(coef)
    coefs.append(coef.reshape(1, -1))

plt.figure(figsize=(7, 5))
plt.plot(degree, np.vstack(coefs))
plt.legend(X.columns)
plt.axvline(x=degree[best_score], linestyle="--")
plt.axhline(y=0, color="black")
plt.show()
```



Оценки коэффициентов в зависимости от λ.
Вертикальные пунктирные линии обозначают значение λ, выбранное в результате перекрестной проверки.

Пример LASSO

```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler

coefs = []
degree = np.linspace(-5, 5, 20)
alphas = np.exp(degree)
for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X, y)
    scaler = StandardScaler()
    coef = lasso.coef_.reshape(-1, 1)
    coef = scaler.fit_transform(coef)
    coefs.append(coef.reshape(1, -1))

plt.figure(figsize=(7, 5))
plt.plot(degree, np.vstack(coefs))
plt.legend(X.columns, loc='upper right')
plt.axhline(y=0, color="black")
plt.show()
```

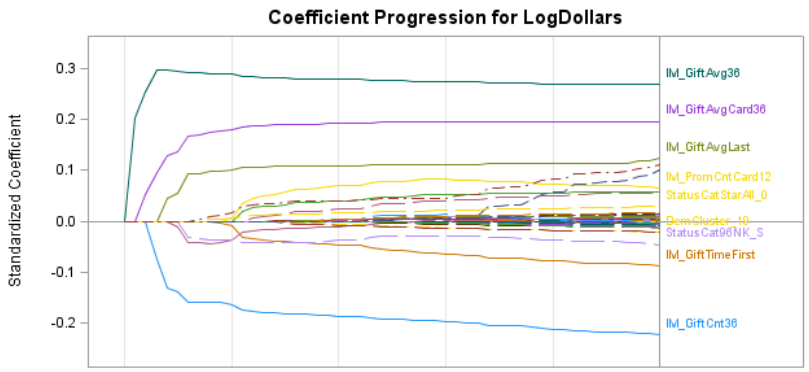


Оценки коэффициентов в зависимости от λ.
Вертикальные пунктирные линии обозначают значение λ, выбранное в результате перекрестной проверки.

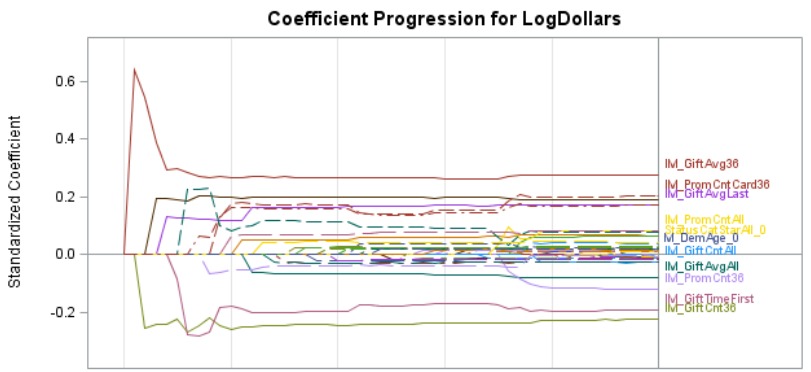
Алгоритм LAR

- Сначала все переменные стандартизуются, а отклик центрируется, и все коэффициенты равны 0.
- Находится предиктор наиболее коррелирующий с откликом и делается максимальный шаг (рассчитывается его коэффициент) по направлению этого предиктора пока не найдется другой предиктор с такой же корреляцией с остатком, как уже добавленный
- Второй предиктор добавляется в модель и делается максимальный шаг в направлении биссектрисы между добавленными предикторами, пока не найдется следующий предиктор, коррелирующий с остатком так же как уже добавленные
- Аналогично находится третий предиктор и так далее.

LAR:

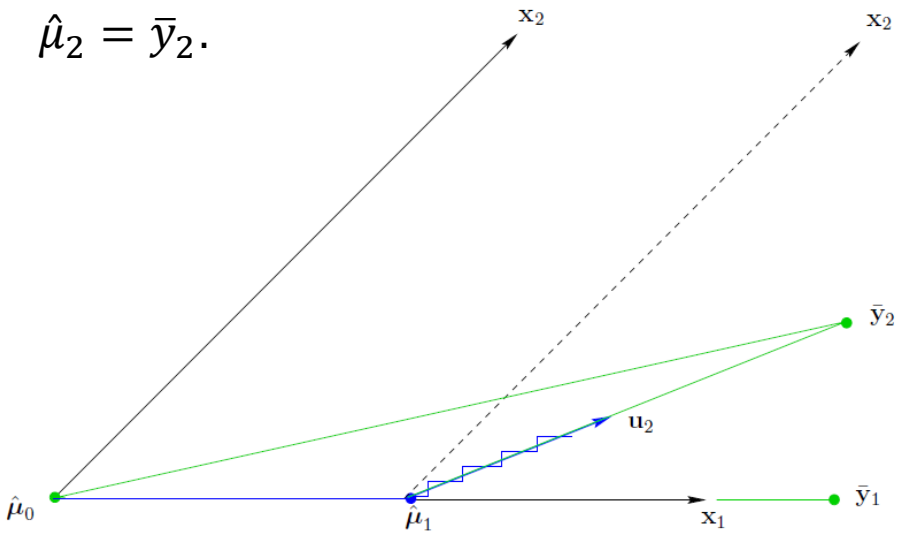


Forward Stepwise:



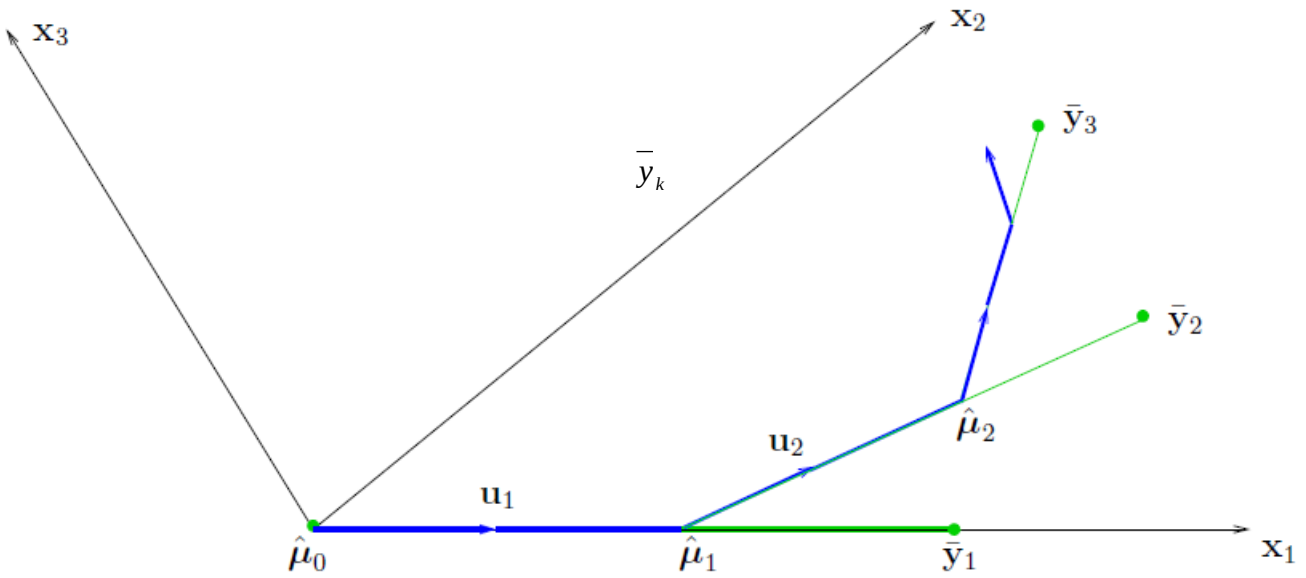
LAR: Двухмерный пример

- LARS $m = 2$ предиктора.
- $\bar{y}_2$ проекция y в $L(x_1, x_2)$ - решение.
- Начинаем $\hat{\mu}_0$= 0, вектор остатков $\bar{y}_2 - \hat{\mu}_0$ больше коррелирует с x_1 чем с x_2.
- LARS находит $\hat{\mu}_1 = \hat{\mu}_0 + \hat{\gamma}_1 x_1$ где $\hat{\gamma}_1$ выбран так что $\bar{y}_2 - \hat{\mu}_1$ биссектриса угла между x_1 и x_2.
- Далее $\hat{\mu}_2 = \hat{\mu}_1 + \hat{\gamma}_2 u_2$ где u_2 единичный вектор вдоль биссектрисы; $\hat{\mu}_2 = \bar{y}_2$.



LAR: Трехмерный пример

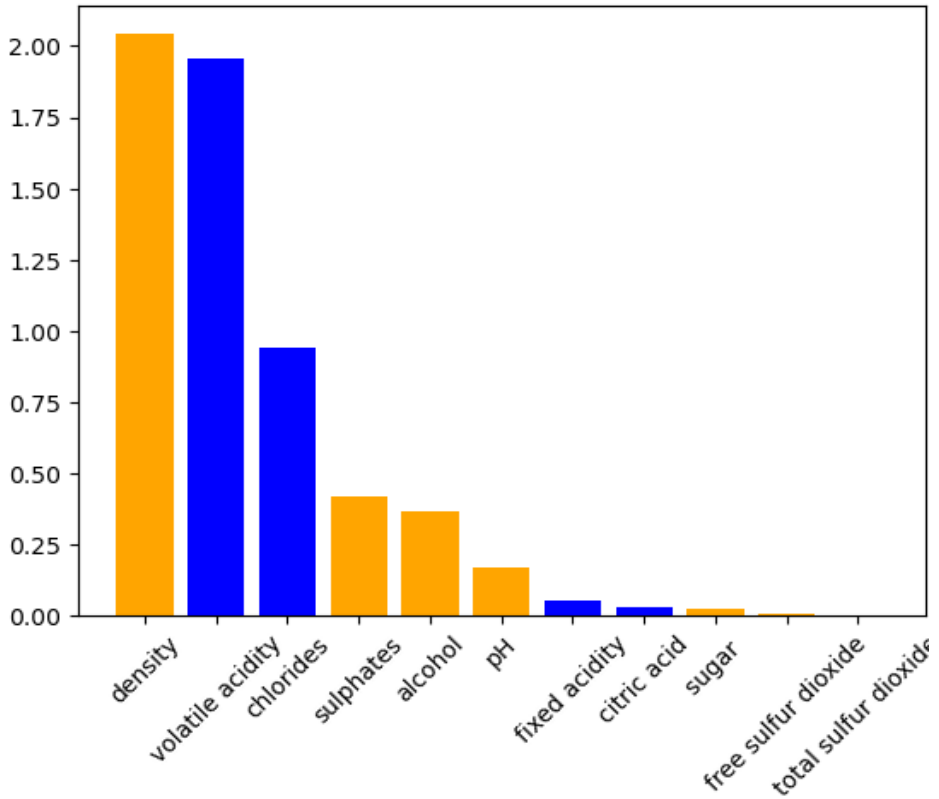
На каждом шаге LARS оценивает $\hat{\mu}_k$ сходясь в результате к МНК решению $\bar{y}_k$.



Пример LAR

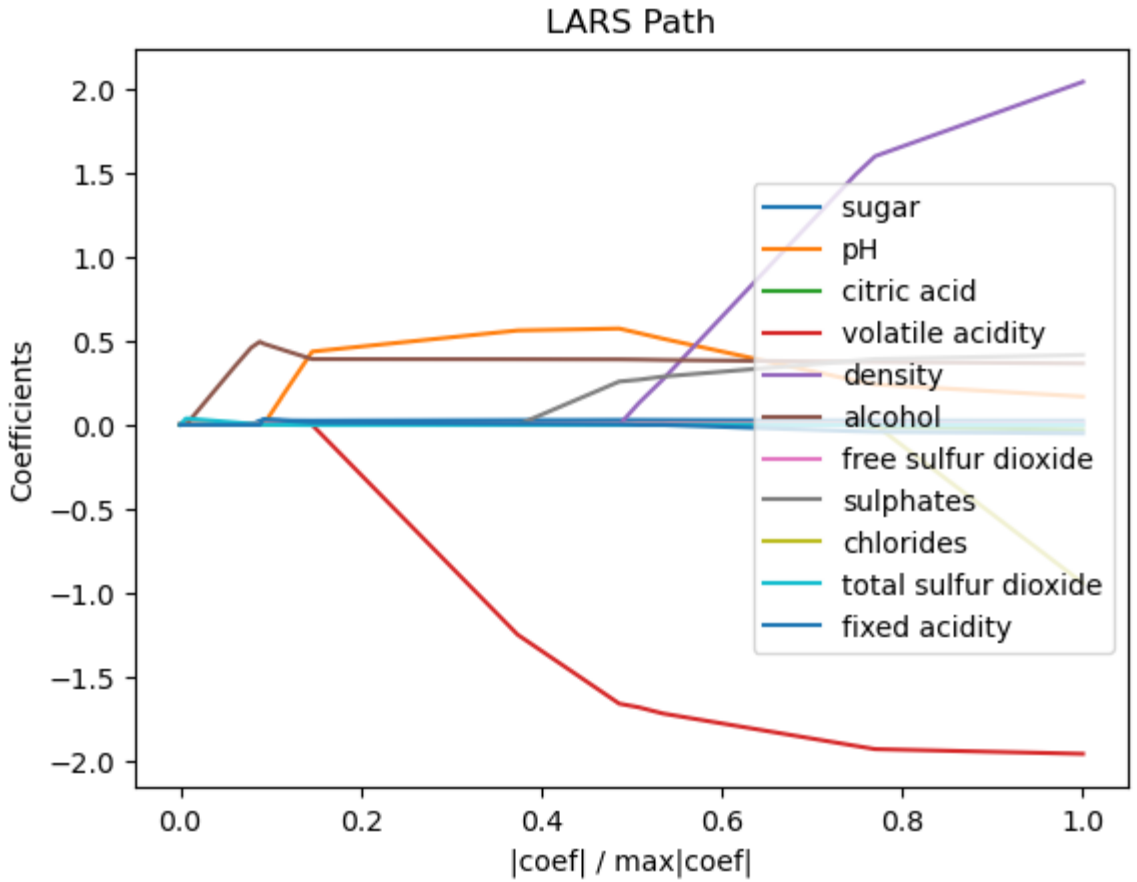
```
from sklearn.linear_model import LassoLarsIC

lars = LassoLarsIC(criterion="aic",
                   normalize=True,
                   fit_intercept=False)
lars.fit(X, y)
```

```
coef = lars.coef_
colors = {True:"orange", False:"blue"}
sign = pd.Series(coef > 0).map(colors)
coef = abs(coef)
ind = np.argsort(coef)[::-1]
plt.bar(lars.feature_names_in_[ind],
        coef[ind], color=sign[ind])
plt.xticks(rotation=45)
plt.show()
```



Пример lars path

```
from sklearn import linear_model
alphas, _, coefs = linear_model.lars_path(
    X.to_numpy(), y.to_numpy(), method="lasso")
xx = np.sum(np.abs(coefs.T), axis=1)
xx /= xx[-1]
```



Методы сокращения размерности: подробности

- Пусть $Z_1, Z_2, \ldots, Z_M$ – M линейных комбинаций ($M < p$) наших исходных p предикторов. т.е.

$$Z_m = \sum_{j=1}^{p} \phi_{mj} X_j \qquad (1)$$

для некоторых $\phi_{m1}, \ldots, \phi_{mp}$.

- Затем мы можем построить модель линейной регрессии

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \epsilon_i, \quad i = 1, \ldots, n, \qquad (2)$$

используя МНК.

- Отметим, что в модели (2), коэффициенты регрессии заданы значениями $\theta_0, \theta_1, \ldots, \theta_M$. Если $\phi_{m1}, \ldots, \phi_{mp}$.
- подобраны хорошо, то такие подходы к снижению размерности могут быть лучше, чем МНК регрессия.

- Заметим, что из определения (1) следует,

$$\sum_{m=1}^{M} \theta_m z_{im} = \sum_{m=1}^{M} \theta_m \sum_{j=1}^{p} \phi_{mj} x_{ij} = \sum_{j=1}^{p} \sum_{m=1}^{M} \theta_m \phi_{mj} x_{ij} = \sum_{j=1}^{p} \beta_j x_{ij},$$

где $$\beta_j = \sum_{m=1}^{M} \theta_m \phi_{mj}. \qquad (3)$$

- Следовательно, модель (2) можно рассматривать как частный случай исходной модели линейной регрессии.
- Снижение размерности необходимо для ограничения коэффициентов β_j , так как теперь они должны принимать форму (3).
- Это может дать выигрыш в компромиссе дисперсии смещения.

Регрессия главных компонент

- Мы применяем анализ главных компонентов (PCA), чтобы определить линейные комбинации предикторов для применения в регрессии.
- Первый главный компонент соответствует (нормализованной) линейной комбинации переменных с самой большой дисперсией.
- Второй главный компонент имеет самую большую дисперсию, при условии отсутствия корреляции с первым.
- И так далее.
- Поэтому если мы имеем много скоррелированных исходных переменных, мы заменим их небольшим набором главных компонентов, которые отражают их совместное изменение.

Применение регрессии главных компонент

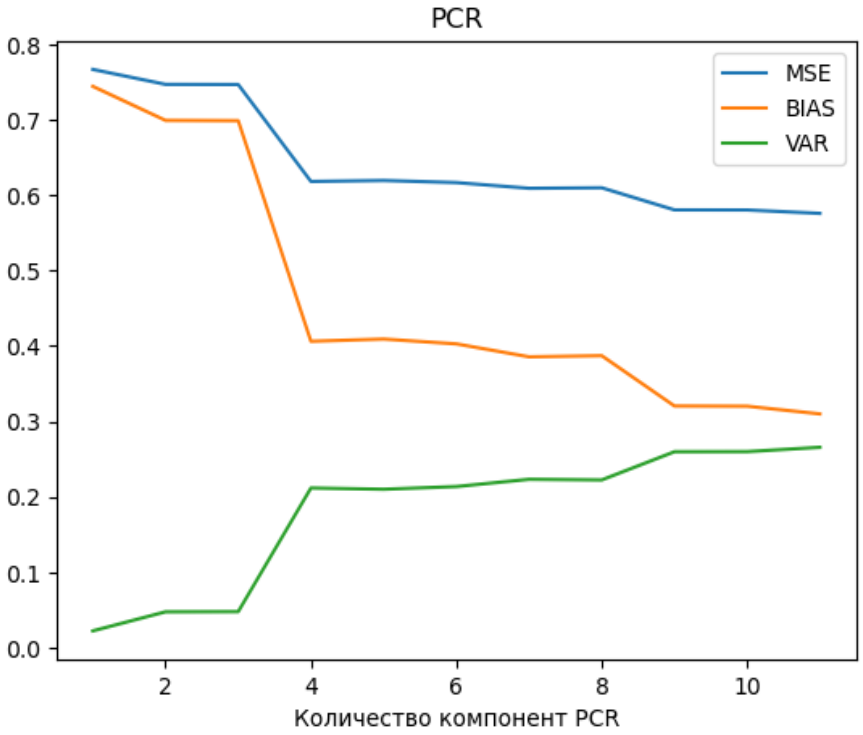
```
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_predict
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

PCR = make_pipeline(PCA(n_components=5), LinearRegression())
y_cv = cross_val_predict(PCR, X, y, cv=10)
mean_squared_error(y, y_cv)
```

```
0.6193829630825505
```

```
# Поиск количества компонент

def optimise_comp_cv(X, y, n_comp):
    PCR = make_pipeline(PCA(n_components=n_comp),
                        LinearRegression())
    # Расчет ошибки на кросс-валидации
    y_cv = cross_val_predict(PCR, X, y, cv=10)
    mse = mean_squared_error(y, y_cv)
    var = explained_variance_score(y, y_cv)
    bias = mse - var
    return mse, bias, var
```

Выбор количества компонент M

```
mse_, bias_, var_ = [], [], []
list_components = list(range(1, X.shape[1]+1))
for n_comp in list_components:
    mse, bias, var = optimise_comp_cv(X, y, n_comp)
    mse_.append(mse)
    bias_.append(bias)
    var_.append(var)

plt.plot(list_components, np.array(mse_))
plt.plot(list_components, np.array(bias_))
plt.plot(list_components, np.array(var_))
plt.xlabel('Количество компонент PCR')
plt.legend(["MSE","BIAS","VAR"])
plt.title('PCR')
plt.show()
```



Оранжевая, зеленая и синяя линии соответствуют смещению, дисперсии и тестовой среднеквадратической ошибке.

Метод частичных наименьших квадратов

- PCR определяет линейные комбинации, или *направления*, которые наилучшим образом представляют предикторы $X_1,\ldots,X_p$.
- Эти направления определяются *обучением без учителя*, так как отклик Y не используется при определении направлений главных компонент.
- То есть отклик не *контролирует* определение главных компонентов.
- Следовательно, PCR страдает от потенциально серьезного недостатка: нет никакой гарантии, что направления, которые наилучшим образом объясняют предикторы, также будут лучшими направлениями при использовании для прогнозирования отклика.

Метод частичных наименьших квадратов (PLS): продолжение

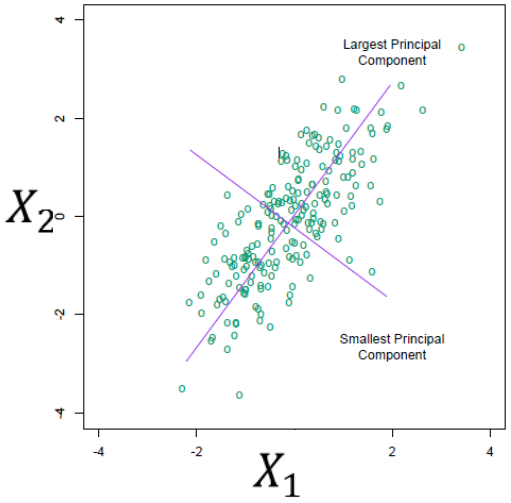
- Подобно PCR, PLS является метод снижения размерности, который сначала определяет новый набор признаков $Z_1,\ldots,Z_M$, которые являются линейными комбинациями исходных признаков, а затем строит линейную модель с помощью OLS с использованием этих *M* новых признаков.
- Но в отличие от PCR, PLS определяет эти признаки на основе контролируемого обучения - то есть, он использует отклик *Y* с целью выявления новых признаков, которые не только хорошо аппроксимируют исходные признаки, но и *связаны с откликом*.
- Грубо говоря, подход PLS пытается определить направления, которые позволяют объяснить как отклики, так и предикторы.

$$\max_{|\alpha|=1, v_l^T S\alpha=0, l=1,\ldots,m-1} Corr^2(y, X\alpha)Var(X\alpha)$$

НА ОСНОВЕ ЛАТЕНТНЫХ ПЕРЕМЕННЫХ

полезно, когда у нас много переменных, и они сильно коррелируют

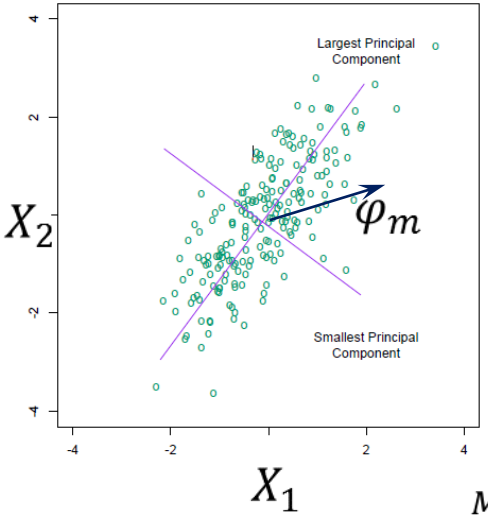
PRINCIPAL COMPONENTS REGRESSION

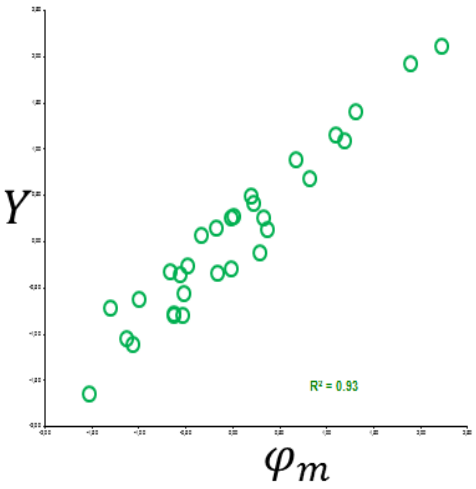


$$X_1, \dots, X_p \rightarrow Z_1, \dots, Z_M, M \ll p, Z_i = Xv_i$$

$$Y^{PCR} = \hat{Y} + \sum_{m=1}^{M} \theta_m Z_m$$

PARTIAL LEAST SQUARES





$$Z_m = \sum_{j=1}^{M} \varphi_m x_j$$

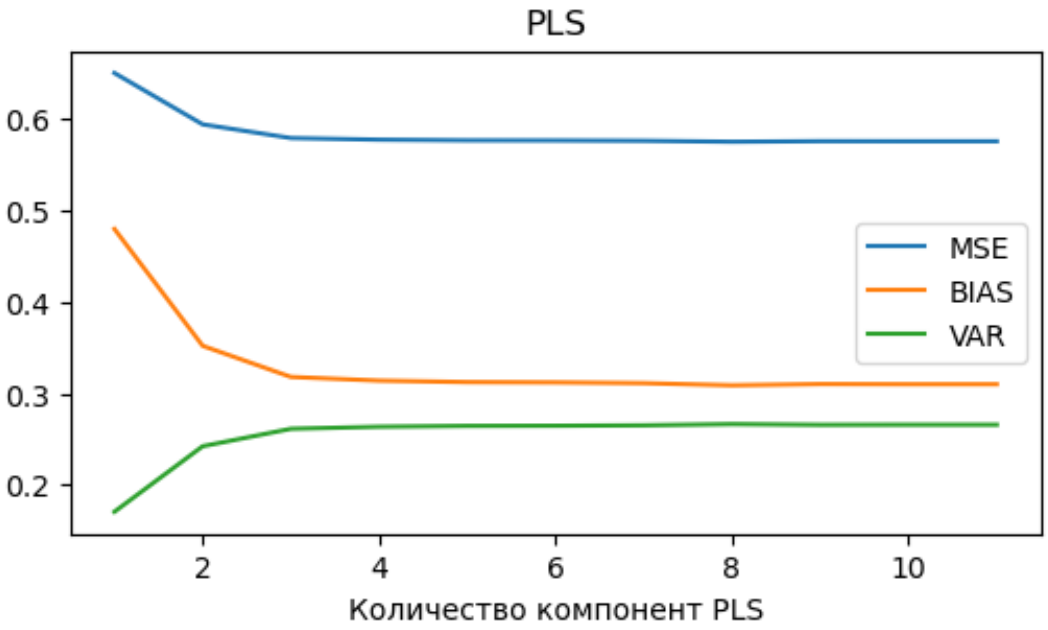
$$\varphi_m: \max_{\alpha} Corr^2(y, X\alpha) Var(X\alpha)$$

Подробности о методе частичных наименьших квадратов

- После стандартизации *p* предикторов, PLS вычисляет первое направление Z_1 на основе установки каждого ϕ_{1j} равным коэффициенту простой линейной регрессии *Y* для X_j.
- Можно показать, что этот коэффициент пропорционален корреляции между Y и X_j .
- Следовательно, при вычислении $Z_1 = \sum_{j=1}^{p} \phi_{1j} X_j,$ PLS устанавливает наибольший вес переменным, которые наиболее тесно связаны с откликом.
- Последующие направления определяются на основе расчета невязки, а затем повторения вышеописанного.

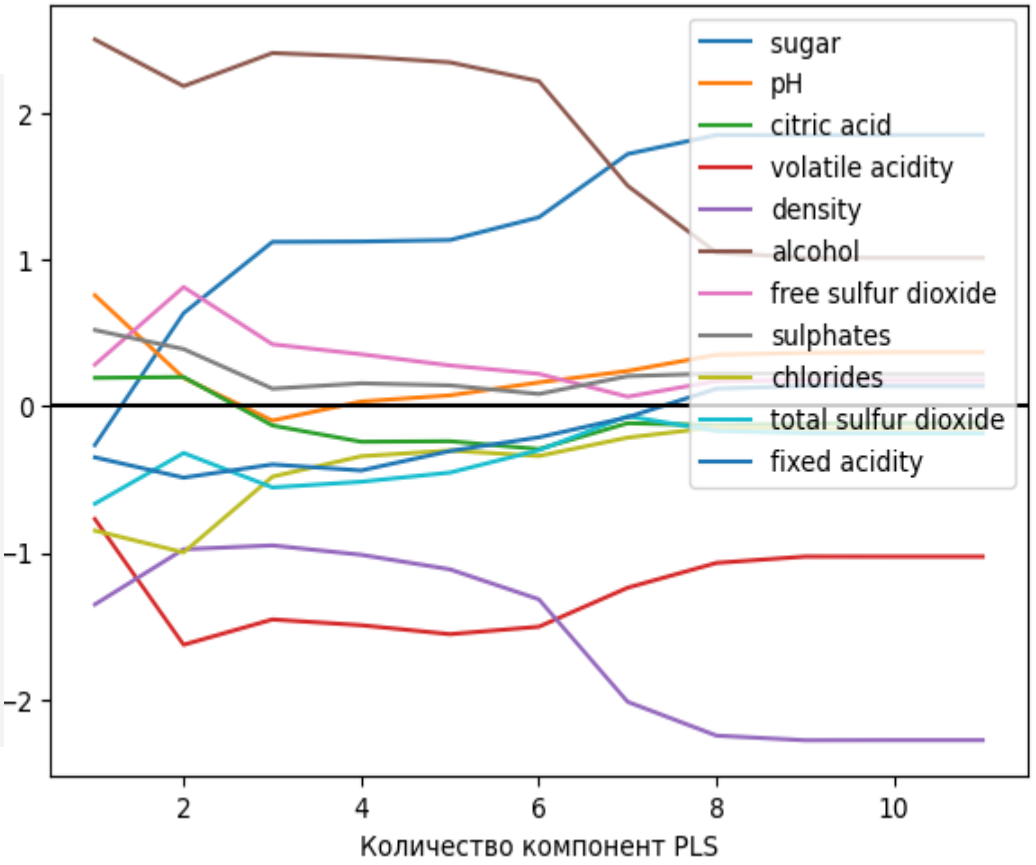
Применение PLS

```python
from sklearn.cross_decomposition import PLSRegression
# Поиск количества компонент

def optimise_comp_cv(X, y, n_comp):
    pls = PLSRegression(n_components=n_comp)
    # Расчет ошибки на кросс-валидации
    y_cv = cross_val_predict(pls, X, y, cv=10)
    mse = mean_squared_error(y, y_cv)
    var = explained_variance_score(y, y_cv)
    bias = mse - var
    return mse, bias, var
```



Применение PLS

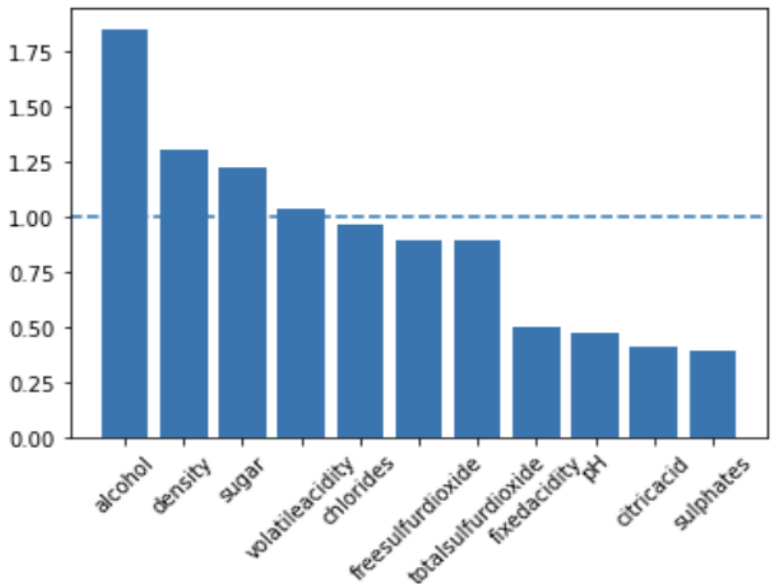
```
coefs = []
list_components = list(range(1, X.shape[1]+1))
for n_comp in list_components:
    pls = PLSRegression(n_components=n_comp)
    pls.fit(X, y)
    scaler = StandardScaler()
    coef = pls.coef_.reshape(-1, 1)
    coef = scaler.fit_transform(coef)
    coefs.append(coef.reshape(1, -1))

plt.figure(figsize=(7, 5))
plt.plot(list_components, np.vstack(coefs))
plt.legend(X.columns, loc='upper right')
plt.axhline(y=0, color="black")
plt.show()
```



Оценка важности переменных в PLS

- VIP статистика (Variable importance in projection) оценивает важность каждого предиктора x_j с учетом значений его коэффициентов w_{jk} в проекции на каждую из k главных компонент с учетом пропорции описанной ей вариации отклика SSY_k/SSY_{total} :

$$vip^2(x_j) = \sum_k w_{jk}^2 SSY_k/SSY_{total}$$




```
def vip(model):
  t = model.x_scores_
  w = model.x_weights_
  q = model.y_loadings_
  p, h = w.shape
  vips = np.zeros((p,))
  s = np.diag(t.T @ t @ q.T @ q).reshape(h, -1)
  total_s = np.sum(s)
  for i in range(p):
      weight = np.array([(w[i,j] / np.linalg.norm(w[:,j]))**2
                         for j in range(h)])
      vips[i] = np.sqrt(p*(s.T @ weight)/total_s)
  return vips

pls = PLSRegression(n_components=5)
pls.fit(X, y)

vips=vip(pls)
ind = np.argsort(vips)[::-1]
plt.bar(pls.feature_names_in_[ind], vips[ind])
plt.xticks(rotation=45)
plt.axhline(y=1, linestyle="--")
plt.show()
```

Пропущенные значения

- Не все значения атрибутов известны или достоверны
  - важная задача, так как многие к ней сводятся (удаление шума, не консистентностей и т.д.)
- Причины появления пропущенных значений
  - Ошибки «оборудования» и/или ПО при получении данных от датчиков и из экспериментов
  - Удаление несогласованных значений атрибутов
  - Просто не введены в систему из-за халатности или ошибки
  - Часть данных может быть опциональна с точки зрения бизнес процессов организации, но важна для анализа
  - Не хранится правильная история изменений – невозможно правильно определить значение на момент анализа
- Пропущенные данные:
  - Ведут к неточным результатам анализа
  - Допускаются не всеми алгоритмами анализа

Методы обработки пропущенных значений

- Игнорировать объект или запись:
  - Можем потерять важные объекты (например, опорные вектора)
  - Можем «испортить» выборочное распределение
  - В некоторых задачах процент пропущенных значений велик (>50%)
- Заполнение пропущенных значений «вручную»:
  - Нужен очень грамотный эксперт
  - Полностью «вручную» невозможно для больших объемов
  - Правила заполнения (импутации) трудно формулировать – проблема полноты, противоречивости, достоверности
- Использование глобальной спец. константы типа “unknown”
  - Не всеми алгоритмами анализа реализуемо
- Импутация «среднего» или «наиболее ожидаемого» значения
  - По всей выборке, по страту (срезу), по классу, по кластеру и т.д.
  - Наиболее популярный метод
  - но можем «испортить» выборочное распределение
- Методы импутации на основе DM
  - Будем рассматривать

Основные подходы к подстановке пропусков

- Импутация константным значением - все пропуски для переменной заменяются на:
  - Моду (для категориальных) или мат. ожидание,
    или пользовательскую константу
    или робастные оценки
- Импутация псевдослучайным значением:
  - В соответсвии с распределением
- Импутация прогнозом (оценкой)
  - Только деревья решений (но можно делать свои модели)

Для неслучайных пропусков – индикаторные переменные

  - Одна на все наблюдение
  - Своя для каждой переменной

Распределения

Оценки

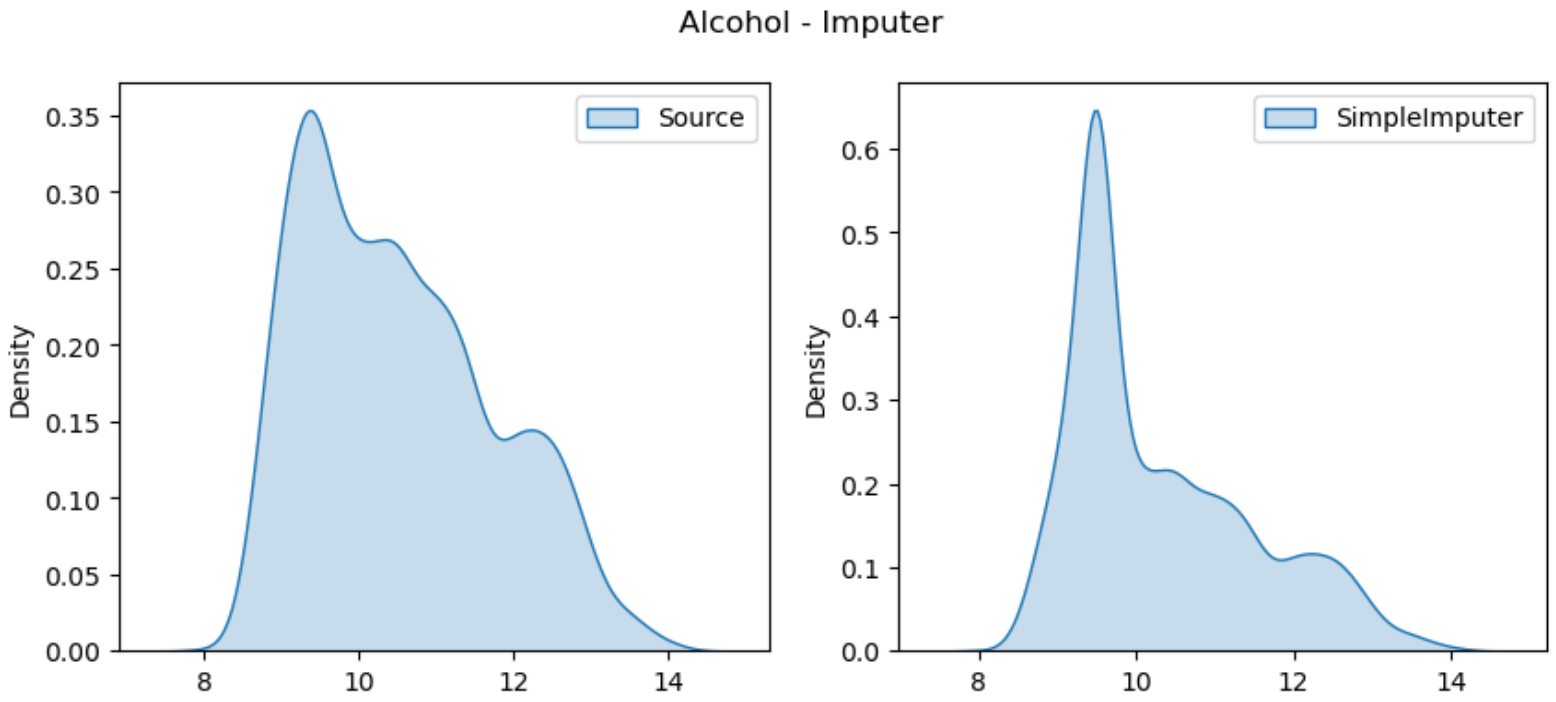
$x_i = f(x_1, \ldots, x_p)$

Пример Simple Imputer

```
X_ = X[["alcohol", "pH"]].values

X_nan = X_.copy()
# Генерация пропусков
ind = np.random.rand(X_nan.shape[0]) < 0.2
X_nan[ind, 0] = np.nan
np.isnan(X_nan).sum()
```

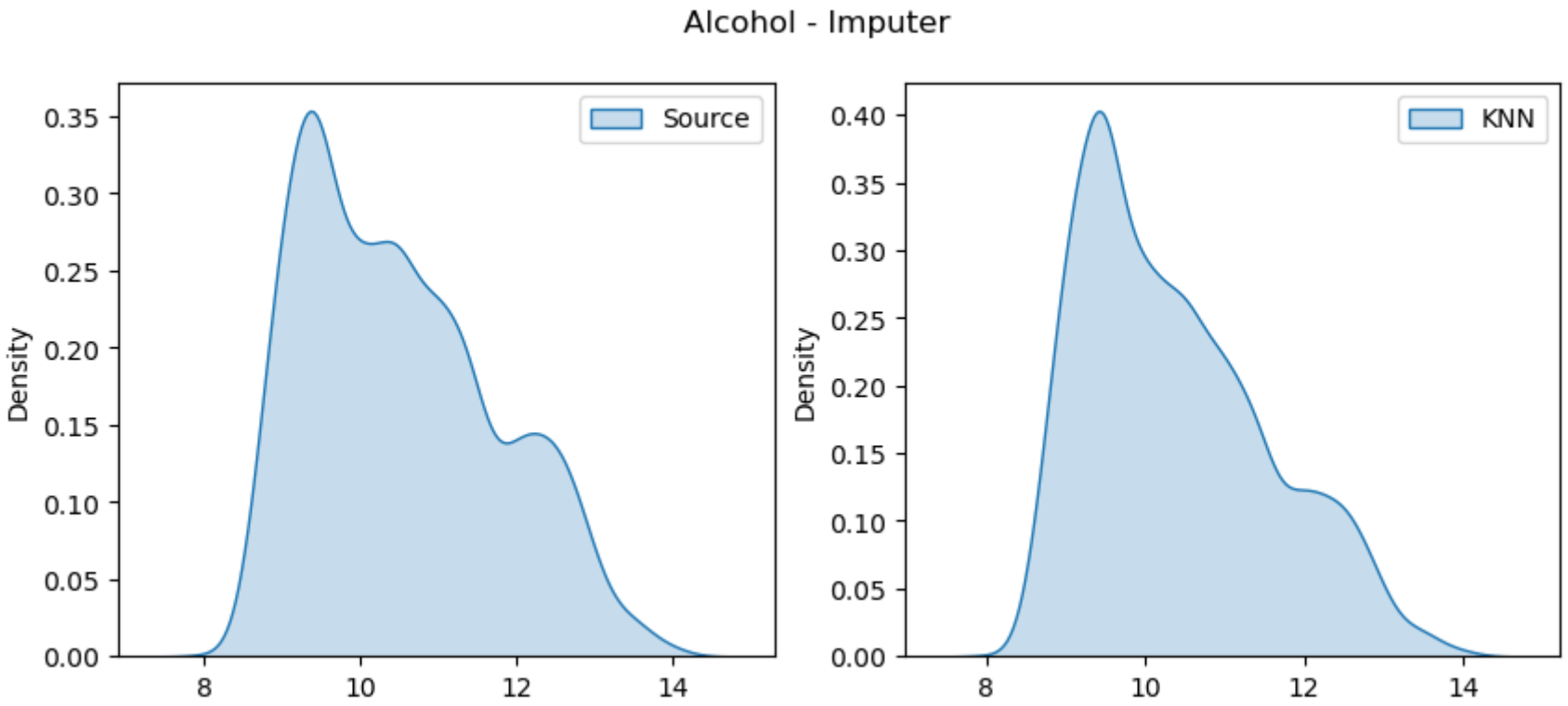
```
970
```

```
from sklearn.impute import SimpleImputer

X_only_nan = X_nan[:, 0].reshape(-1, 1)
a = pd.DataFrame(X_only_nan, columns=["Source"])
imp = SimpleImputer(strategy='most_frequent')
imp.fit_transform(X_only_nan)
```



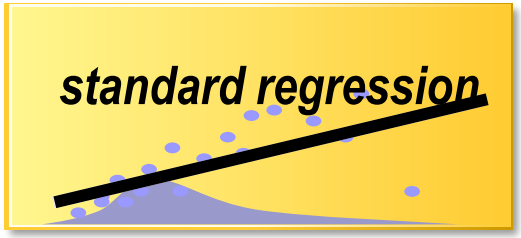
Пример KNN Imputer

```
from sklearn.impute import KNNImputer
a = pd.DataFrame(X_only_nan, columns=["Source"])
knn_imp = KNNImputer(n_neighbors=2)
knn_imp.fit_transform(X_nan)[:, 0]
```



Преобразование непрерывных переменных

- Простые преобразования:
  - Функции от исходной (log, exp, …)



  - Нормализация (z-score, центрирование, сведение на [0,1])

$$v' = \frac{v - mean_A}{stand\_dev_A} \qquad v' = \frac{v - min_A}{max_A - min_A}$$

  - Дискретизация (равные интервалы, равные группы, адаптивные интервалы с учетом отклика и т.д.)
- Адаптивные преобразования – перебор простых и выбор лучшего по некоторому критерию:
  - Нормальность распределения результата
  - Корреляция с откликом
  - Оптимальная дискретизация

Преобразование категориальных переменных

Level	D_A	D_B	D_C	D_D	D_E	D_F	D_G	D_H	D_I
A	1	0	0	0	0	0	0	0	0
B	0	1	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	0	0
D	0	0	0	1	0	0	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	0	0	1	0	0	0
G	0	0	0	0	0	0	1	0	0
H	0	0	0	0	0	0	0	1	0
I	0	0	0	0	0	0	0	0	1

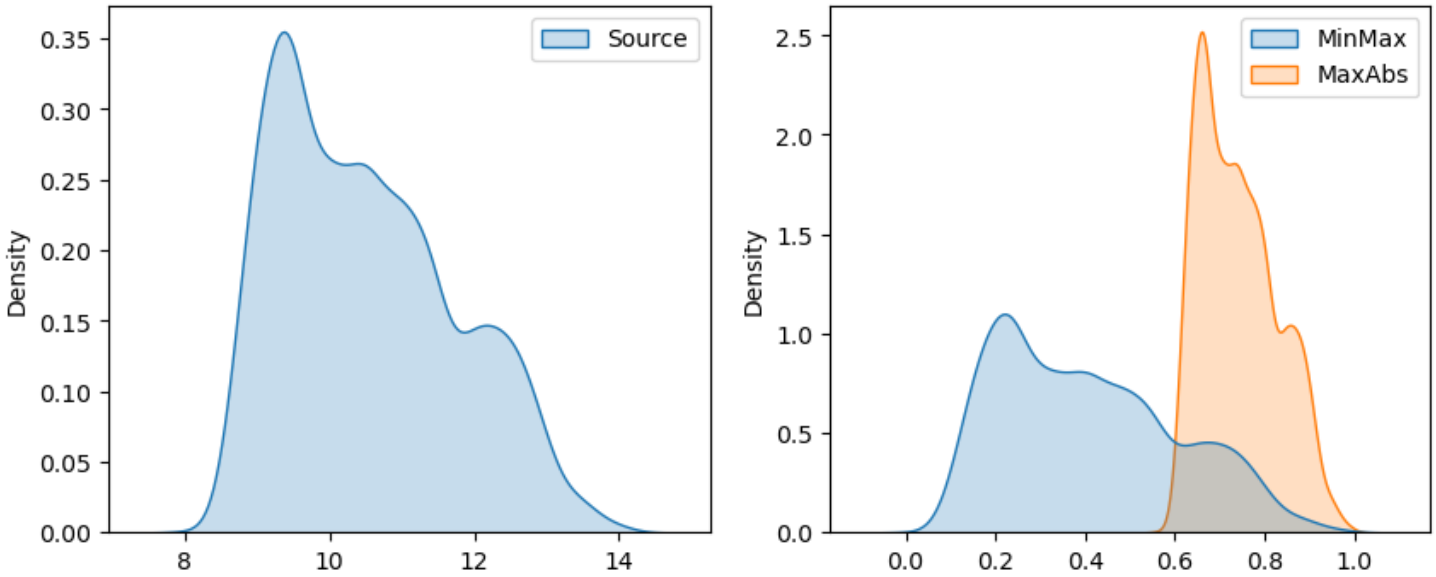
- Бинарное кодирование
- Выделение категории редких признаков
- Группировка признаков по поведению отклика (будет подробнее позже)

Level	N_i	ΣY_i	p_i
A	1562	430	0.28
B	970	432	0.45
C	223	45	0.20
D	111	36	0.32
E	85	23	0.27
F	50	20	0.40
G	23	8	0.35
H	17	5	0.29
I	12	6	0.50
J	5	5	1.00

Преобразование непрерывных переменных (Normalization)

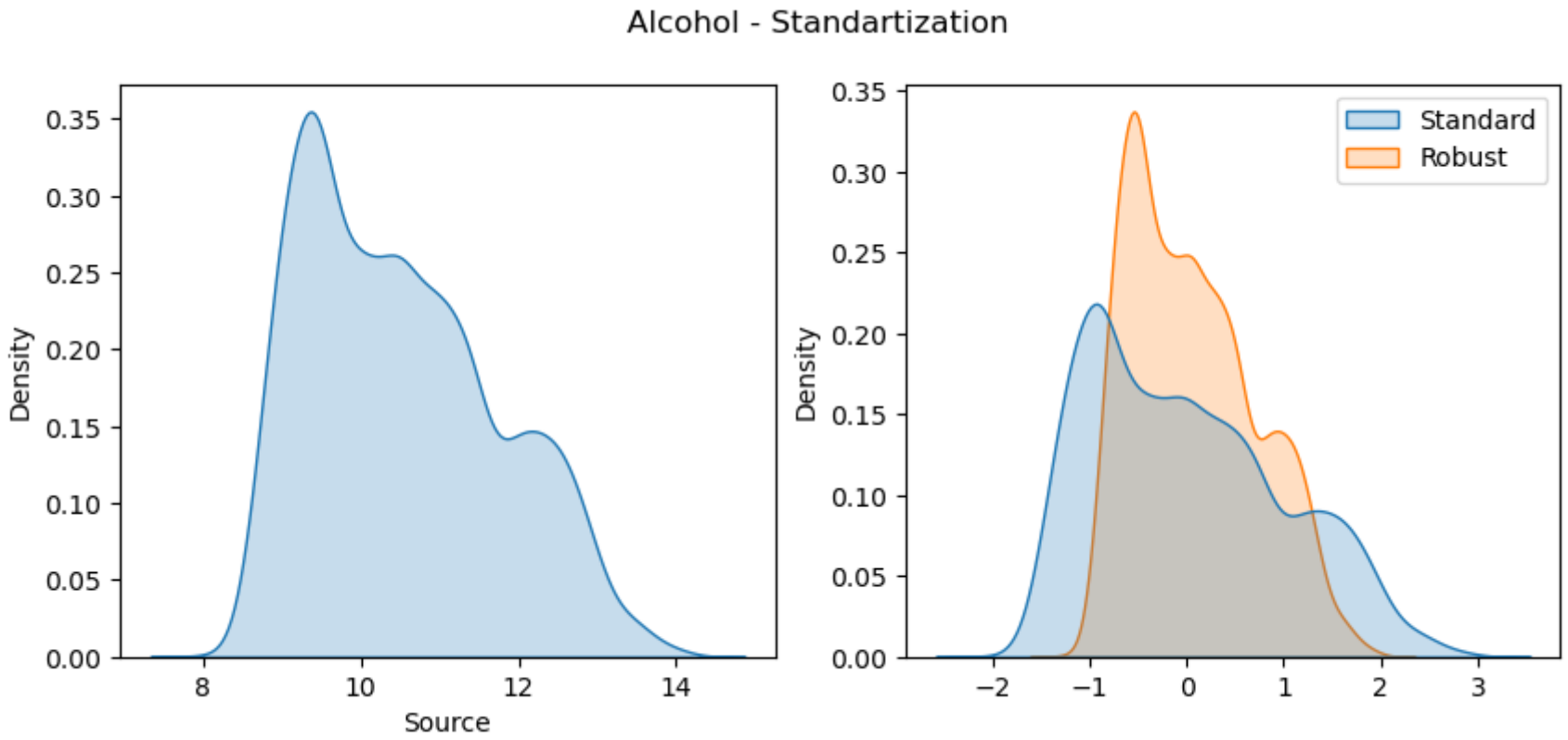
```
X_ = X["alcohol"].values.reshape(-1, 1)
```

```
from sklearn.preprocessing import MinMaxScaler, MaxAbsScaler
a = pd.DataFrame(X_, columns=["Source"])

fig, axes = plt.subplots(ncols=2, figsize=(10, 4))
fig.suptitle("Alcohol - Normalization")
sns.kdeplot(a[["Source"]], ax=axes[0], fill=True)

mm = MinMaxScaler().fit_transform(X_)
ma = MaxAbsScaler().fit_transform(X_)
```



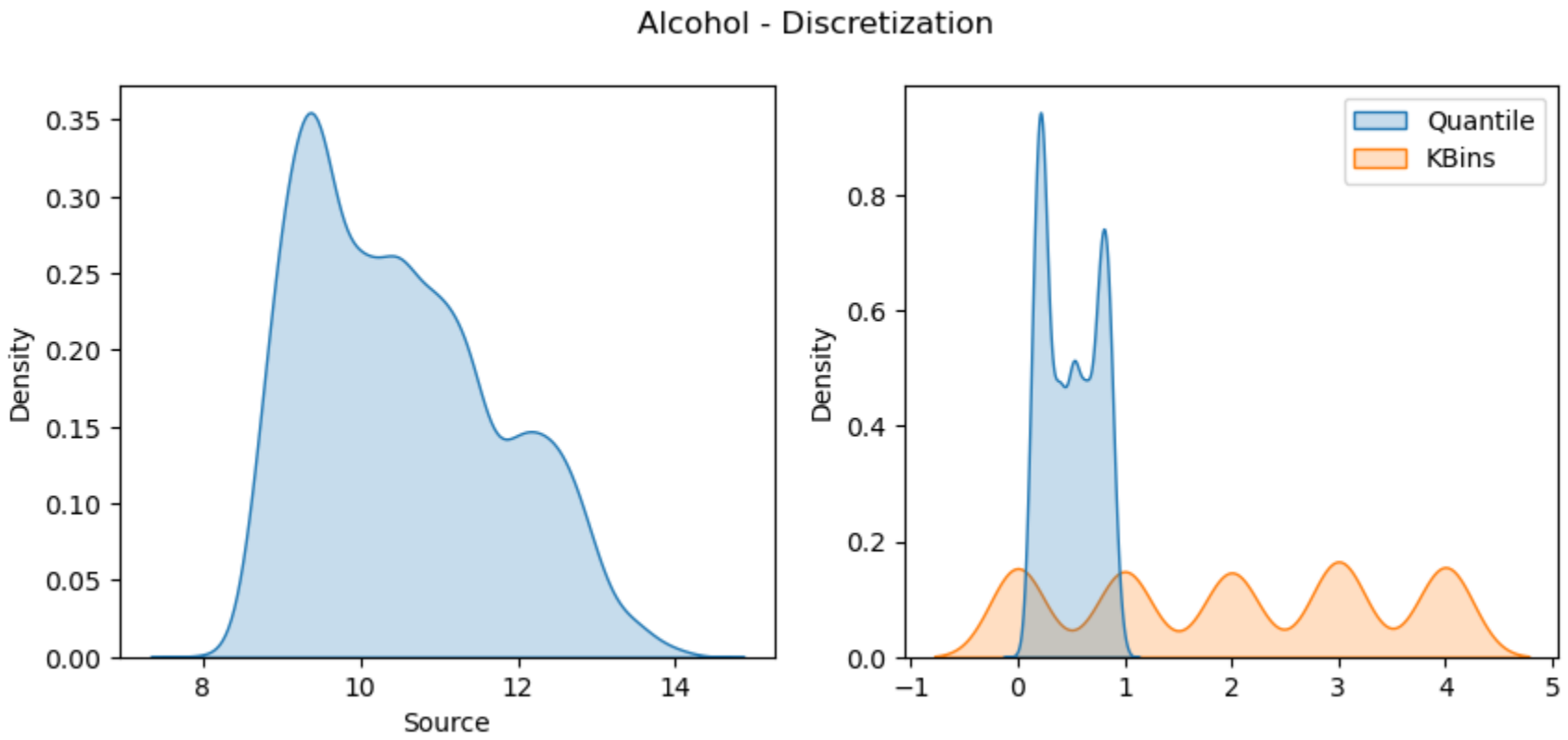
Преобразование непрерывных переменных (Standartization)

```
from sklearn.preprocessing import StandardScaler, RobustScaler

st_sc = StandardScaler().fit_transform(X_)
rb_sc = RobustScaler().fit_transform(X_)
```



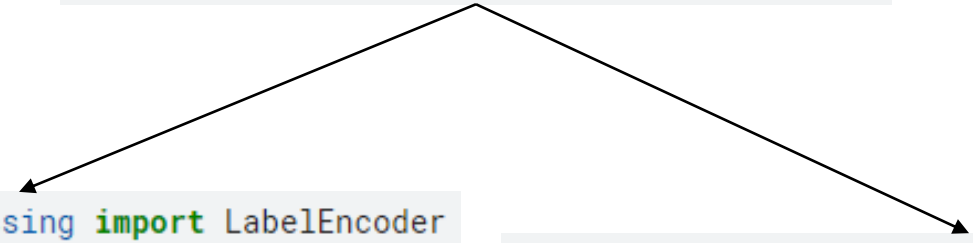
Преобразование непрерывных переменных (Discretization)

```
from sklearn.preprocessing import QuantileTransformer, KBinsDiscretizer

q_tr = QuantileTransformer(n_quantiles=5).fit_transform(X_)
kbn = KBinsDiscretizer(encode="ordinal").fit_transform(X_)
```



Преобразование категориальных переменных

```
quality = y.to_numpy().reshape(-1, 1)
```



```
from sklearn.preprocessing import LabelEncoder

print("quality", quality[-5:, 0])
le = LabelEncoder()
labels = le.fit_transform(quality)
print("Classes", le.classes_)
print("LabelEncoder")
print(labels[-5:])
```

```
quality [6 5 6 7 6]
Classes [3 4 5 6 7 8 9]
LabelEncoder
[3 2 3 4 3]
```

```
from sklearn.preprocessing import OneHotEncoder
print("quality", quality[-5:, 0])
ohe = OneHotEncoder()
code = ohe.fit_transform(quality).toarray()
print("OneHotEncoder")
print(code[-5:])
```

```
quality [6 5 6 7 6]
OneHotEncoder
[[0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0.]]
```