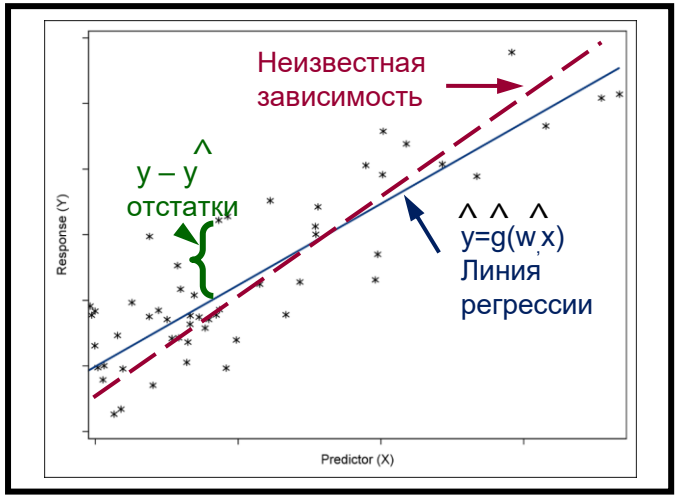
Основы линейных и обобщённых линейных моделей

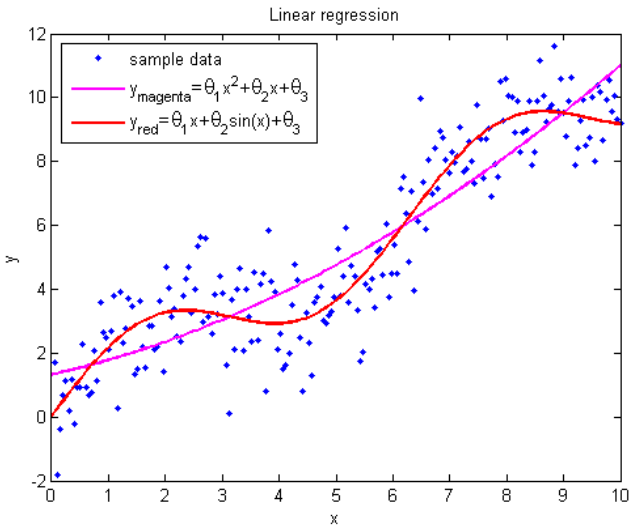
Параметрические модели

- Параметрическое семейство функций

  $A = \{g(x, w) \mid w \in \Theta\}, g: X \times \Theta \to Y$

  Θ – множество допустимых параметров
- Линейная регрессия:
  - Регрессия $g(x, w) = \sum_{i=1}^{q} w_i f_i(x), Y = \mathbb{R}$
    - w-вектор параметров
    - f_i-не обязательно линейная
    - если f_i-зависит от одного признака,
    - то модель аддитивная
- Пример:
  - Признаки $\{1, x, x^2\}$ vs $\{1, x, \sin(x)\}$





Линейная регрессия

- Линейная модель представляет собой важный пример параметрической модели для регрессии:

$$a(x_1, \ldots, x_p) = E(Y | X_1 = x_1, \ldots, X_p = x_p)$$

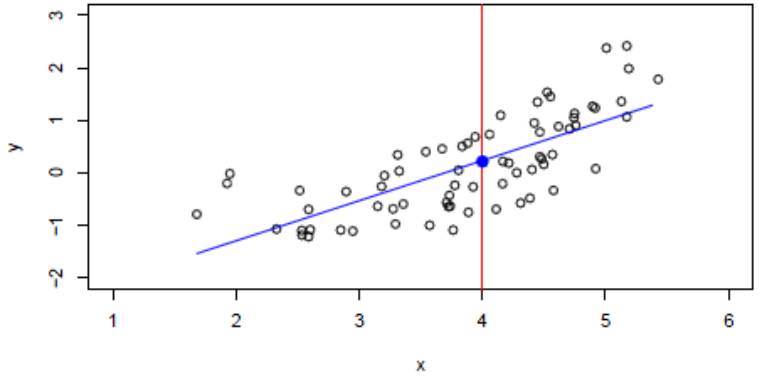
- Она определяется:
  - Линейным уравнением $a(x) = w_0 + \sum_{j=1}^{p} x_j w_j + \varepsilon$
  - *p + 1* параметром w_i, которые нужно найти по обучающей выборке
  - Минимизируя квадратичную функцию потерь

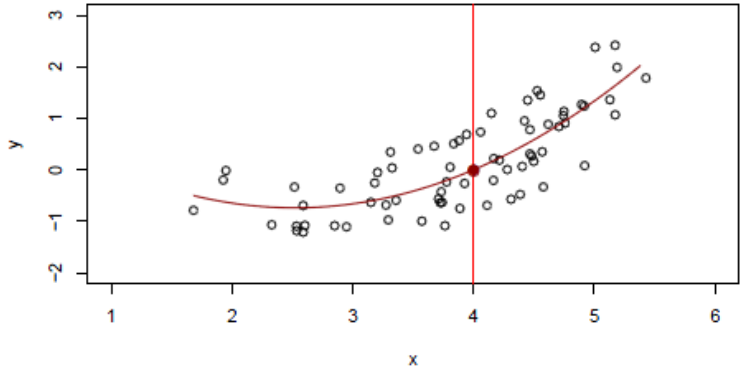
$$Q(w, Z) = \frac{1}{l} \sum_{i=1}^{l} (y_i - a(\bar{x}_i, w))^2 = \frac{1}{l} \sum_{i=1}^{l} \left(y_i - w_0 - \sum_{j=1}^{p} x_{ij} w_j \right)^2$$

- Должны быть выполнены предположения:
  - Шум распределен по закону $\varepsilon \sim N(0, \sigma^2)$
  - Наблюдения в выборке независимы и одинаково распределены (i.i.d)
  - Мы «угадали» с линейным уравнением и признаковым пространством

Регрессионный анализ

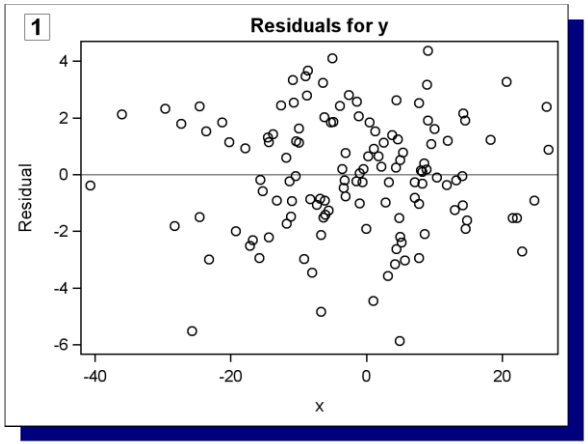
- Линейные модели *почти никогда* не показывают высокую точность, но служат хорошей и интерпретируемой аппроксимацией неизвестной истинной зависимости.



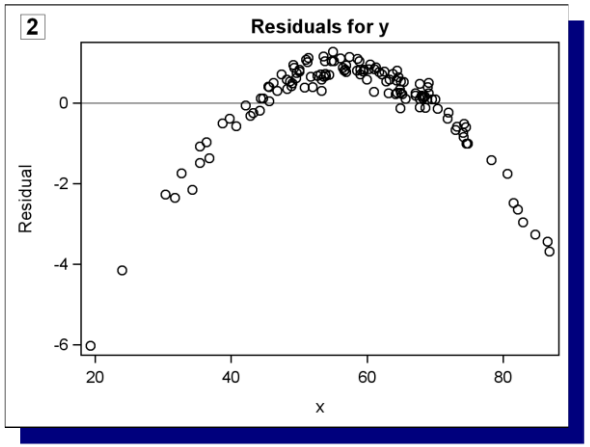


- Цель регрессионного анализа:
  - Определение наличия связи между переменными и характера этой связи (подбор уравнения)
  - Предсказание значения зависимой переменной с помощью независимых
  - Определение вклада отдельных независимых переменных в вариацию зависимой

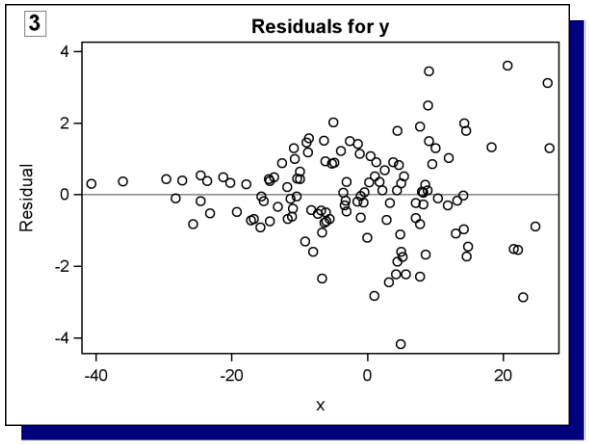
Графики остатков



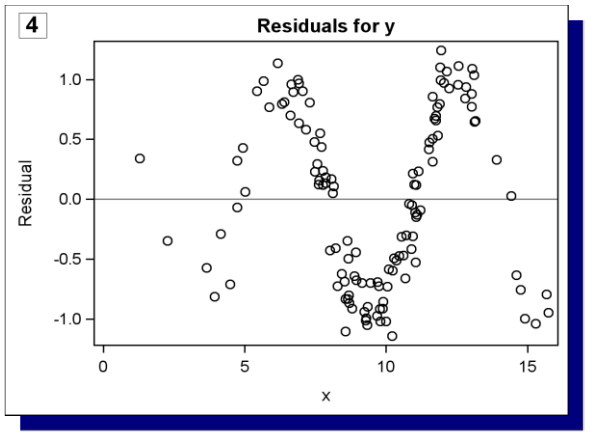
норма



Нелинейная зависимость



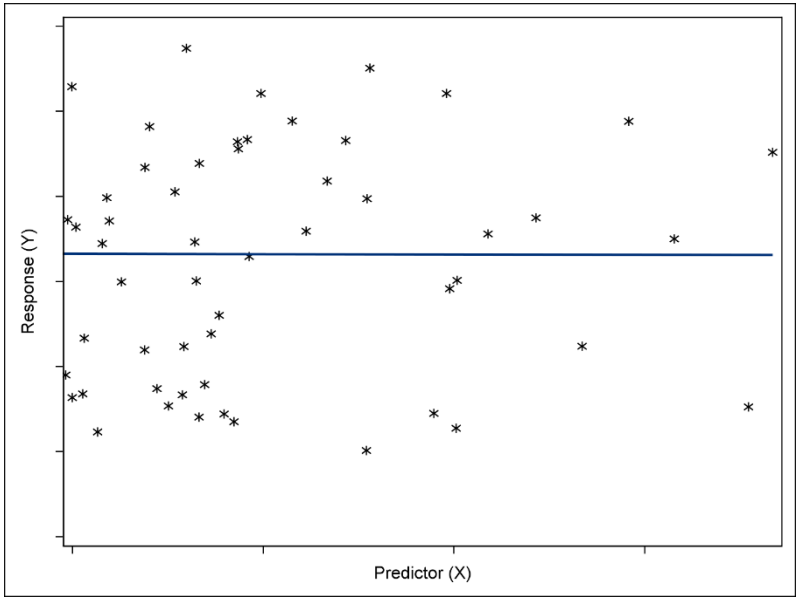
Гетероскедастичность

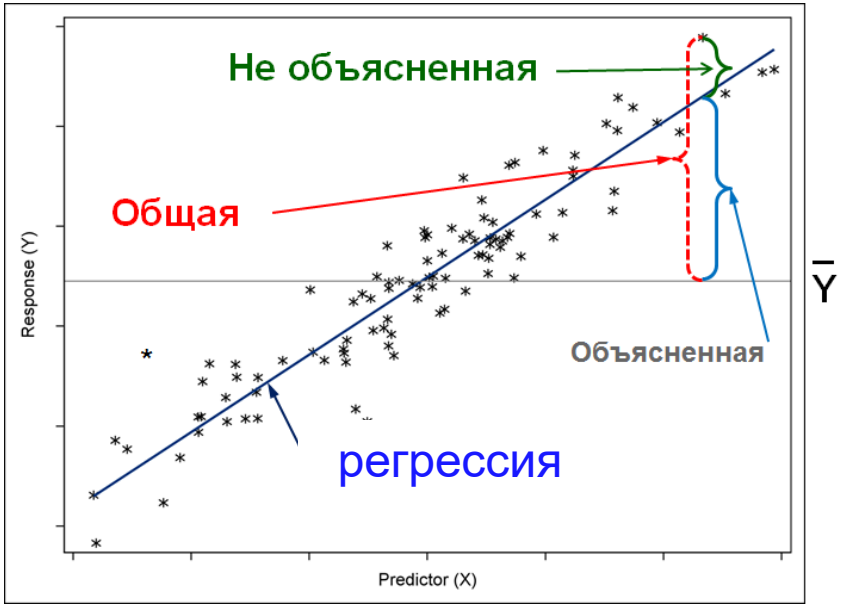


Зависимость наблюдений

Для проверки предположений регрессионной модели полезно построить графики зависимости остатков от прогноза

Объясненная и необъясненная дисперсия





- Необъясненная дисперсия отклика - сумма квадратов остатков (невязок): $RSS = \sum_l \left(y_i - a(x_i)\right)^2$
- Общая дисперсия отклика: $TSS = \sum_l (y_i - E(y))^2$
- Объясненная дисперсия отклика – разность общей и необъясненной: $MSS = TSS - RSS = \sum_l \left(E(y) - a(x_i)\right)^2$

Линейная регрессия с точки зрения статистики

- Стандартная ошибка невязок $RSE = \sqrt{\frac{1}{l-2} RSS}$
- Коэффициент детерминации *R-квадрат* или доля объясненной дисперсии:

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

- Критерий Фишера для проверки базовой гипотезы (что отклик не зависит от предикторов):

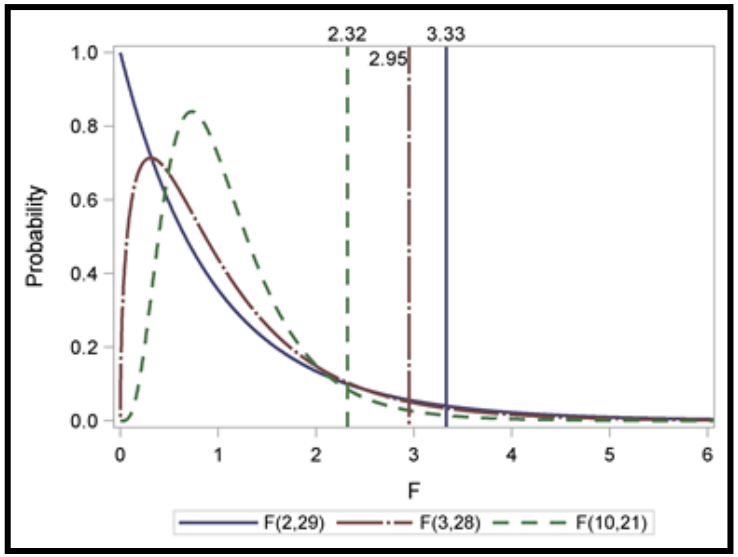
$$\mathrm{F} = \frac{(TSS-RSS)/p}{RSS/(l-p-1)} \sim F_{p,l-p-1}$$

Нулевая гипотеза:

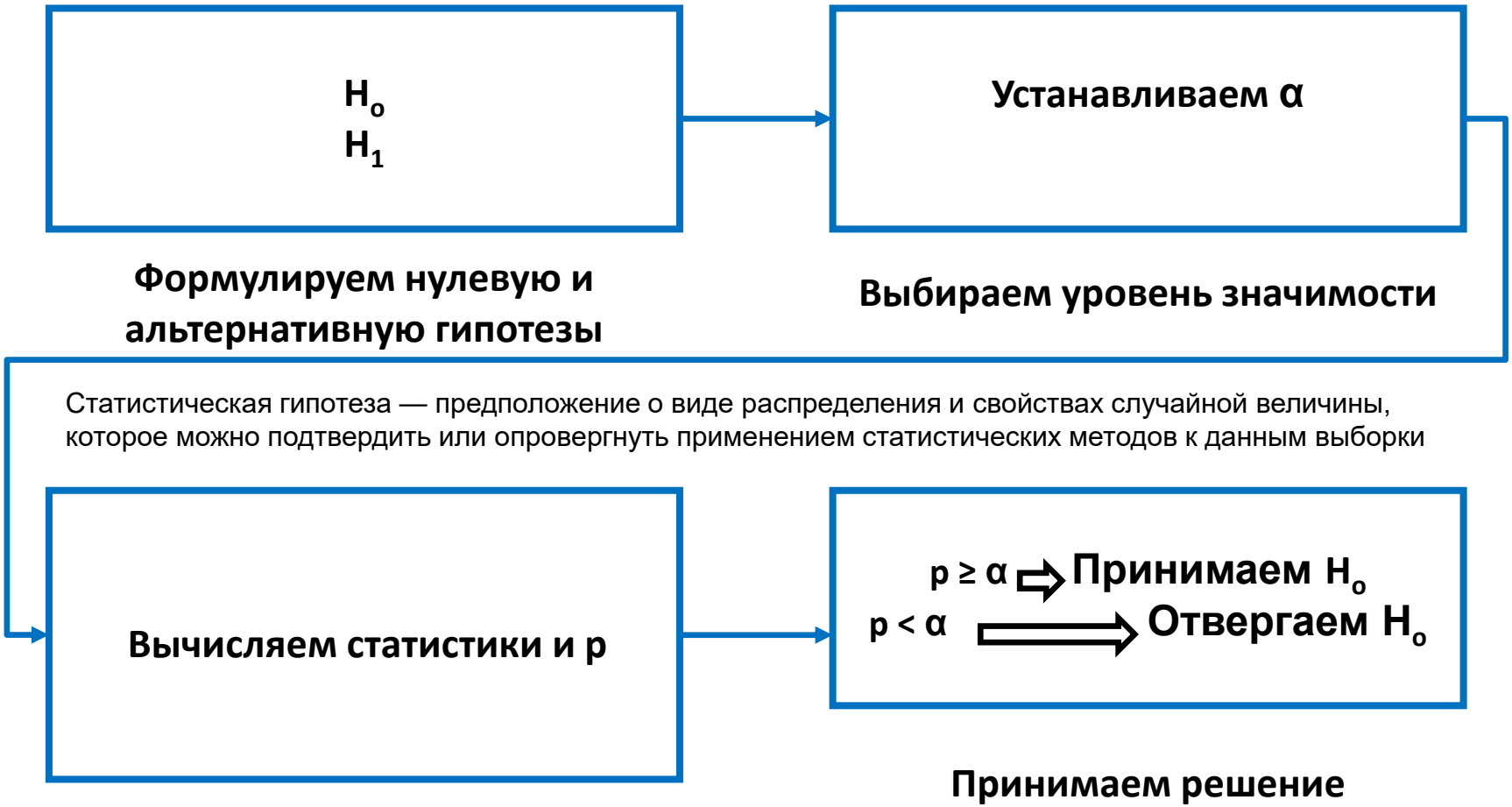
- все $w_i=0$

Альтернативная гипотеза:

- существует $w_i \neq 0$



Проверка статистических гипотез



P-значение равно вероятности того, что случайная величина с данным распределением тестовой статистики при нулевой гипотезе примет значение, более экстремальное, чем фактически полученное значение тестовой статистики (оно же вероятность ложно положительной ошибки, или ошибки I рода)

Уровень значимости и мощность

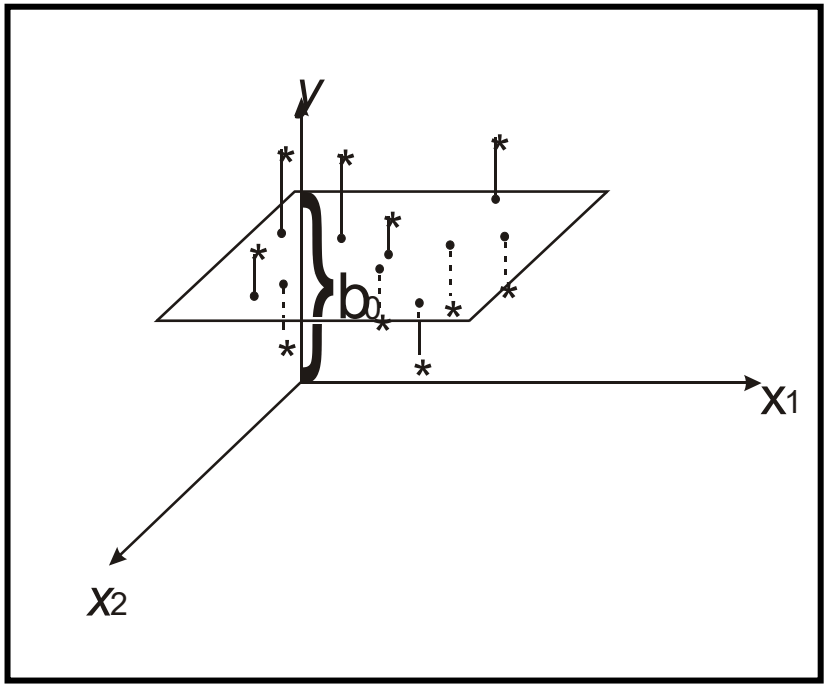
Реальность / Решение	H_0 Истина	H_0 Ложна
Принимаем H_0	Правильно	Ошибка II рода $p(\text{Type II}\|H_1)=\beta$
Отвергаем H_0	Ошибка I рода $p(\text{Type I}\|H_0)=\alpha$	Правильно $(1 - \beta)=$ *Мощность*

Мощность зависит (обратно) от α, размера выборки и зачастую от самой статистики.

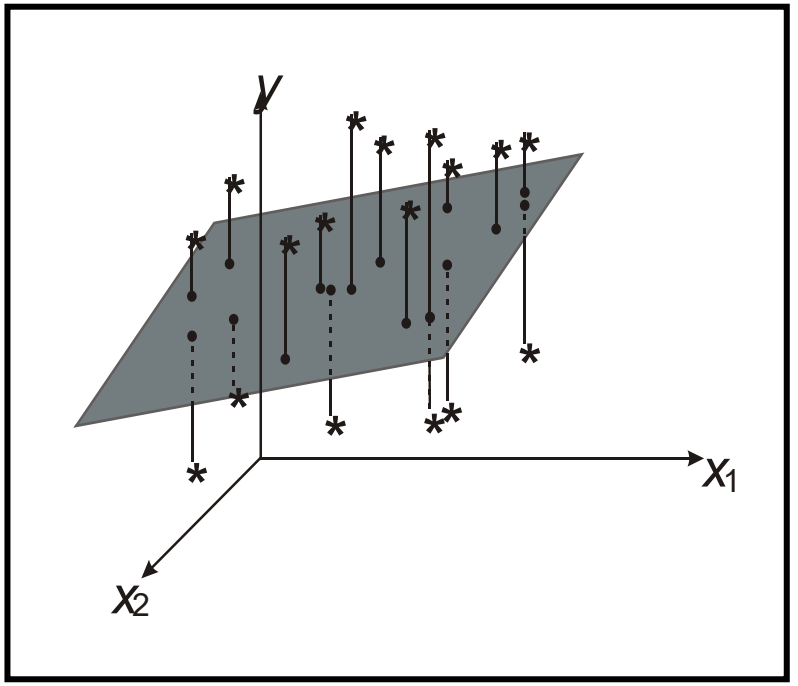
Для простых случаев можно напрямую найти необходимый размер выборки при заданных ограничениях на мощность, уровень значимости и в зависимости от проверяемой гипотезы

Односторонние и двусторонние тесты.

Множественная линейная регрессия

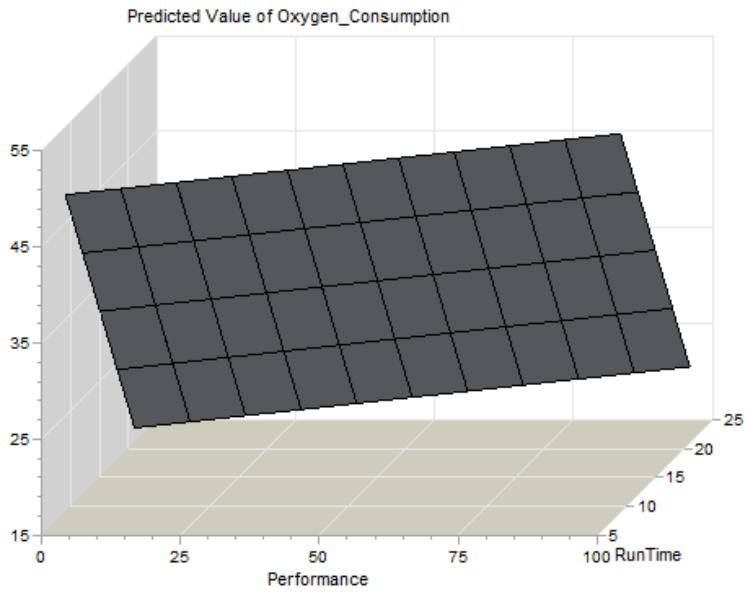


p-value для статистики Фишера больше заданного уровня значимости (например, 5%), тогда принимают базовую гипотезу, что **нет зависимости**



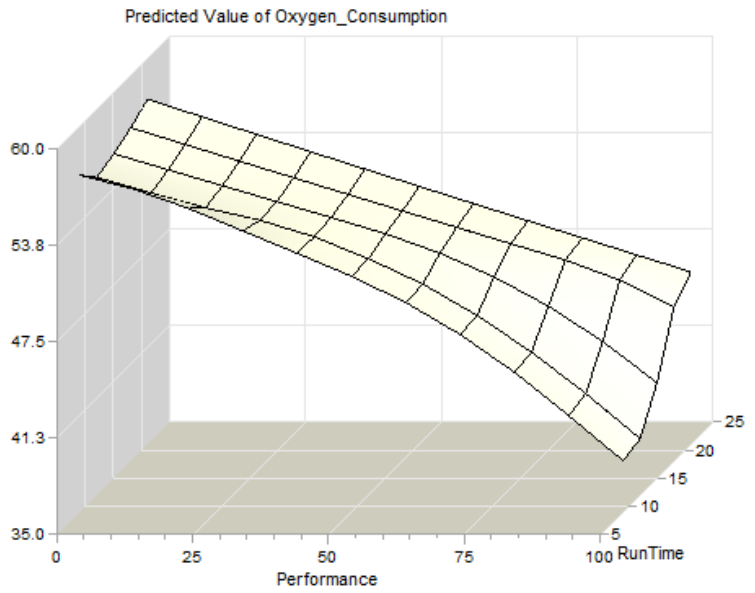
p-value для статистики Фишера меньше заданного уровня значимости (например, 5%), тогда **не** принимают базовую гипотезу, предполагая, что **зависимость есть**

Множественная линейная регрессия



Линейная модель с линейными эффектами

$$a(x) = w_0 + \sum_{j=1}^{p} x_j w_j + \varepsilon$$



Линейная модель с нелинейными эффектами, например

$$a(x) = w_0 + \sum_{j=1}^{p} x_j w_j + \sum_{j,i=1} x_i x_j w_{ij} + \varepsilon$$

или пример аддитивной

$$a(x) = w_0 + \sum_{j=1}^{p} f(x_j) w_j + \varepsilon$$

Метод наименьших квадратов для линейной регрессии

- Оценка ошибки - сумма регрессионных остатков (эмпирический риск с квадратичной функцией потерь):

$$RSS(w) = \sum_{i=1}^{l}(y_i - a(\bar{x}_i))^2 = \sum_{i=1}^{l}(y_i - w_0 - \sum_{j=1}^{p} x_{ij}w_j)^2$$

- В матричной форме: $RSS(w) = (y - Xw)^T(y - Xw)$ - квадратичная функция с p+1 параметрами, где w – вектор искомых параметров, X-матрица данных (строки – наблюдения, колонки - признаки), y – вектор известных откликов
- Найдем и приравняем нулю производную по вектору параметров, получим решение:

$$\nabla_w RSS(w) = -2X^T(y - Xw) = 0 \Rightarrow w = (X^T X)^{-1}X^T y$$

- Поскольку целевая функция выпуклая и матрица вторых производных имеет вид: $\nabla_w^2 RSS(w) = -2X^T X$ ⇒решение единственное и прогноз отклика можно найти по формуле:

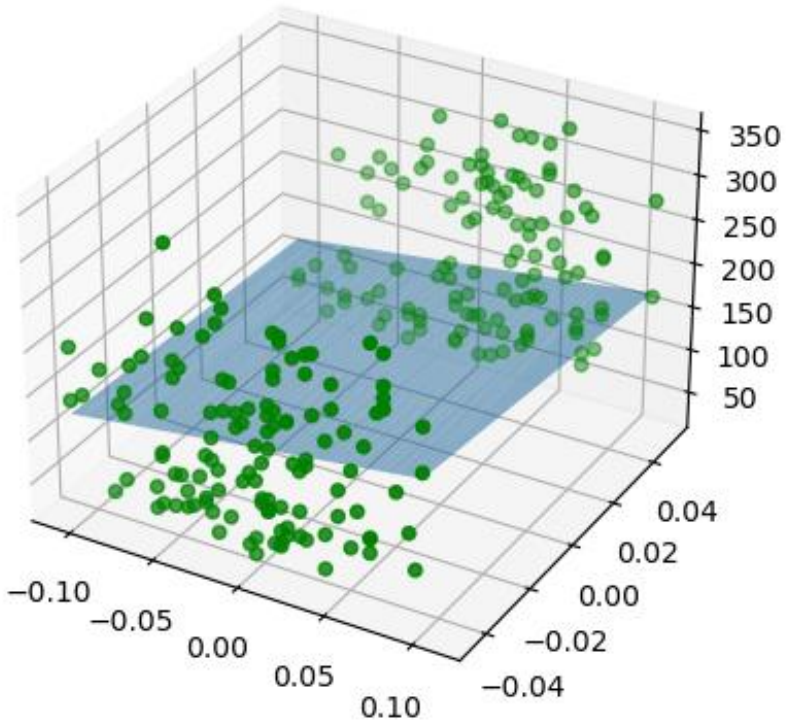
$$\hat{y} = Xw = X\left(X^T X\right)^{-1}X^T y = Hy$$

Пример (Python)

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

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

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




```python
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)
```

Метод наименьших квадратов для линейной регрессии

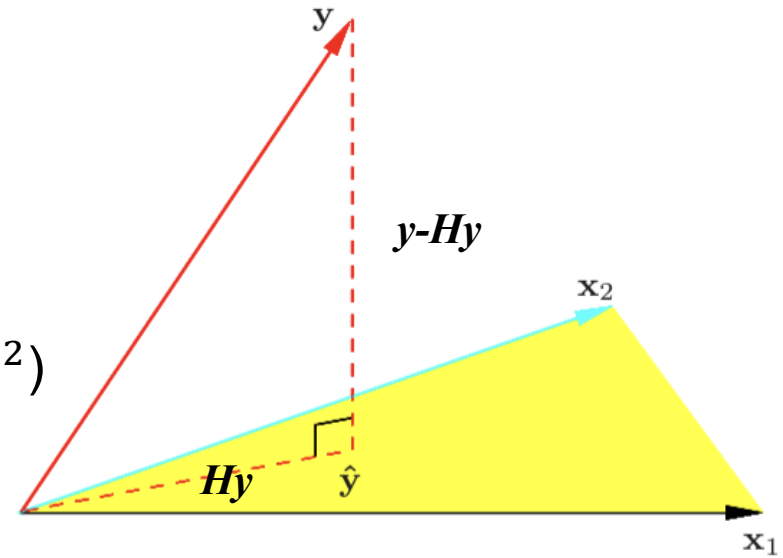
- $H = X\left(X^T X\right)^{-1} X^T$ - проекционная матрица (иногда называют «калькой» с английского – матрица «шляпы»), проецирует отклик на линейную оболочку столбцов матрицы данных (признаковое пространство)
- Можно оценить дисперсию прогноза и дисперсию оценок коэффициентов и соответствующих доверительных интервалов:

$$\hat{\sigma}^2 = \frac{RSS}{l-p-1},\ Var(w) = \left(X^T X\right)^{-1}\hat{\sigma}^2,$$

$$SE(w_i) = \hat{\sigma}\sqrt{diag\left((X^T X)^{-1}\right)_i}$$

- Предполагая $w \sim N(w^{true}, \left(X^T X\right)^{-1}\hat{\sigma}^2)$

получим 95% интервал $w_i \pm 2SE(w_i)$

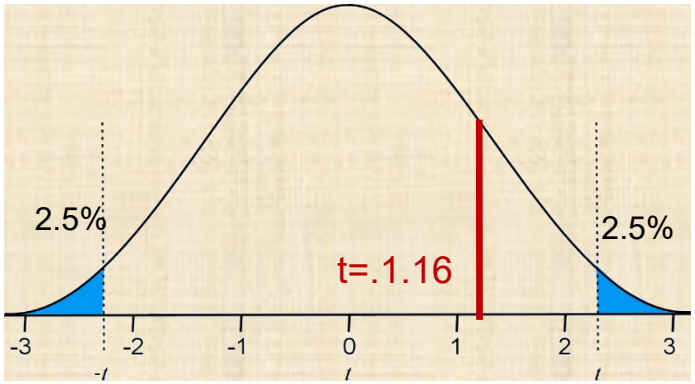


Проверка важности предикторов в МНК

- Для оценки важности предиктора можно проверить нулевую гипотезу (о равенстве нулю коэфициента), вычисляя *t-статистику* (критерий студента): $t_i = w_i / SE(w_i)$

Она будет иметь распределение Стьюдента с $l - 2$ степенями свободы

Можно вычислить вероятность наблюдения значения статистики, большего или равного |t|, это *p*-value.



$$\widehat{\texttt{sales}} = \beta_0 + \beta_1 \times \texttt{TV}$$

	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

Бинарные 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

Категориальные признаки с двумя и более 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

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

Категориальные признаки с двумя и более 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

Кодирование категориальных переменных по отклику

- Основная идея:
  - Отобразить категориальную переменную на числовую шкалу так, чтобы на новой шкале «рядом» были значения категориальных переменных, на которых отклик ведет себя одинаково, например:

$$X_i^{new} = E(y|\ X^{old} = Val_i),$$

где X^{old} - категориальная переменная,

Val_i - одно из значений X^{old},

$X^{new} \in \mathbb{R}$ - новая числовая переменная.

id	job	job_mean	target
1	Doctor	0,50	1
2	Doctor	0,50	0
3	Doctor	0,50	1
4	Doctor	0,50	0
5	Teacher	1	1
6	Teacher	1	1
7	Engineer	0,50	0
8	Engineer	0,50	1
9	Waiter	1	1
10	Driver	0	0

- Зачем?
  - Упрощает модель, сохраняя информацию
  - Уменьшает потенциальные корреляции с другими признаками (они часто возникают при one hot кодировании нескольких категориальных переменных)
  - Уменьшает возможность переобучения (т.к. модель проще)
  - Увеличивает стабильность модели (т.к. нет «редких уровней)

Пропущенные значения

- Не все значения признаков 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

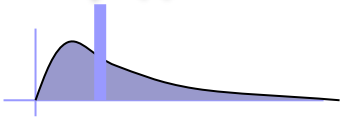
Методы обработки пропущенных значений

- 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 страте (срезу), по классу, по кластеру и т.д.
  - Наиболее популярный метод, но можем «испортить» выборочное распределение
  - Методы импутации на основе прогнозирования

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

Распределения



- Импутация псевдослучайным значением:
  - В соответсвии с распределением
- Импутация прогнозом (оценкой)
- Для неслучайных пропусков
  - индикаторные переменные
  - Одна на все наблюдение
  - Своя для каждой переменной

Оценки

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

Интерпретация коэффициентов регрессии

- Идеальный сценарий: предикторы не коррелированы:
  - Каждый коэффициент можно оценить и тестировать отдельно.
  - Интерпретации такие как «*единичное изменение предиктора связано с w_j –ым изменением в значении отклика, тогда как все остальные переменные остаются фиксированными*»
  - Для корректной оценки влияния предиктора на отклик нужно либо стандартизировать коэффициенты, либо нормировать признаковое пространство
- Корреляции между переменными вызывают проблемы:
  - Дисперсия всех коэффициентов имеет тенденцию к увеличению
  - Интерпретации становятся непредсказуемыми - когда предиктор меняется, зависимые с ним тоже меняется.

«По сути, все модели ошибочны, но некоторые из них полезны»

George Box

Библиотека statsmodels

- Установка:
  - `python -m pip install statsmodels`
- Зависимости:
  - Python >= 3.9, NumPy >= 1.22.3, SciPy >= 1.8, Pandas >= 1.4, Patsy >= 0.5.6
- Загрузка:
  - `In [1]: import statsmodels.api as sm`
  - `In [2]: import pandas`
  - `In [3]: from patsy import dmatrices`
- Работа с матрицами и формулами:
  - `In [10]: y, X = dmatrices('Lottery ~ Literacy + Wealth + Region', data=df, return_type='dataframe')`
- Основные особенности:
  - Линейные регрессии – класс OLS, Обобщенные линейные модели – GLM
  - Много разных типов моделей для стат. анализа
  - У классов есть метод **fit** – обучение (возвращает модель), **predict** – прогнозирование (возвращают прогноз)
  - Есть классы для статистических тестов `sm.stats` и визуализации `sm.graphics`

«R-подобные» формулы в statsmodels

символ	значение
~	Отделяет отклик(и) от предикторов y~x1+x2+x3
+	Отделяет предикторы друг от друга y~x1+x2+x3
:	Задает взаимодействие предикторов y~x1+x2+x3+x1:x2
*	Задает комбинации взаимодействующих предикторов y~x1*x2+x3 эквивалентно y~x1+x2+x1:x2+x3
^	Задает порядок взаимодействия предикторов y~(x1+x2+x3)^2 эквивалентно y~x1+x2+x3+x1:x2+x1:x3+x2:x3
.	Включает в модель все переменные источника y~. Эквивалентно y~x1+x2+x3, если в наборе есть x1,x2,x3
-	Исключает предиктор из формулы переменную или константу (если указано -1) y~(x1+x2+x3)^2-x2:x3 эквивалентно y~x1+x2+x3+x1:x2+x1:x3
I()	Внутри скобок арифметическое выражение y~I((x1+x2+x3)^2) означает x1+x2+x3 в квадрате
Q()	Внутри скобок «сложное» имя переменной, например, с пробелами, пунктуацией и кириллицей Q(“средняя зарплата за квартал в руб.”)
C()	Внутри скобок имя категориальной переменной
func	Использование функции func от предикторов или отклика

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

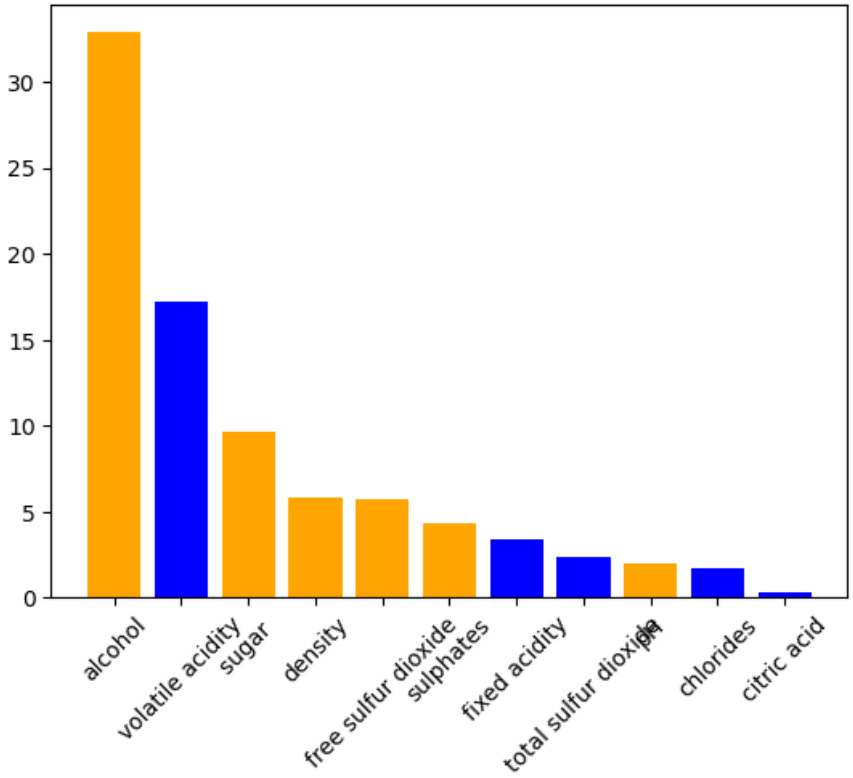
```python
import statsmodels.api as sm

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		

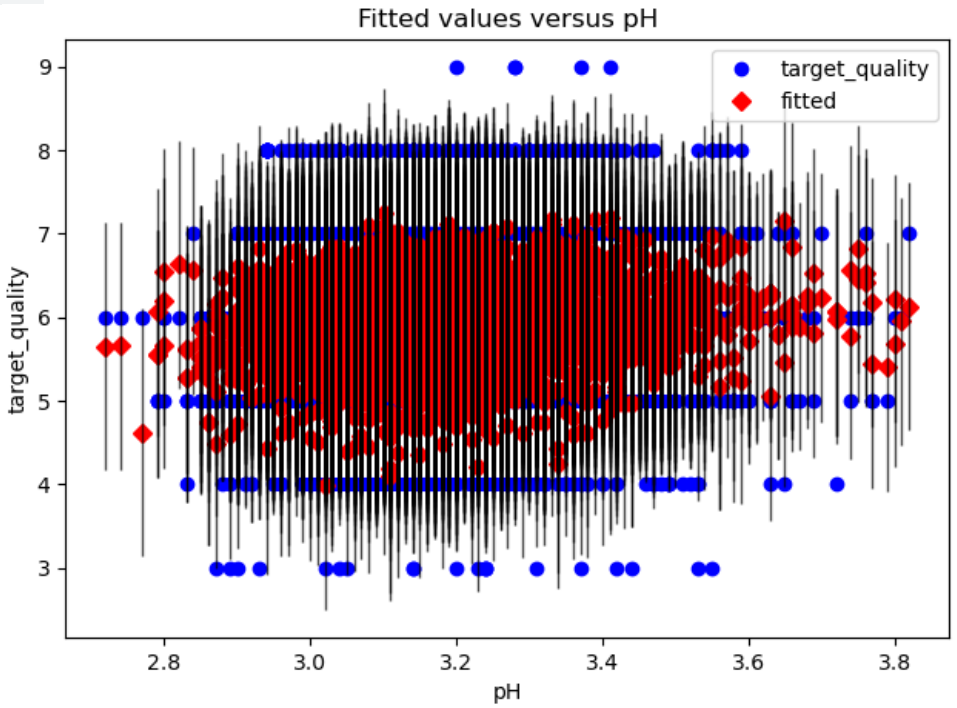
	coef	std err	t	P>|t|	[0.025	0.975]
sugar	0.0250	0.003	9.642	0.000	0.020	0.030
pH	0.1684	0.084	2.014	0.044	0.005	0.332
citric acid	-0.0293	0.096	-0.305	0.760	-0.218	0.159
volatile acidity	-1.9585	0.114	-17.196	0.000	-2.182	-1.735
density	2.0420	0.353	5.780	0.000	1.349	2.735
alcohol	0.3656	0.011	32.880	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()
```

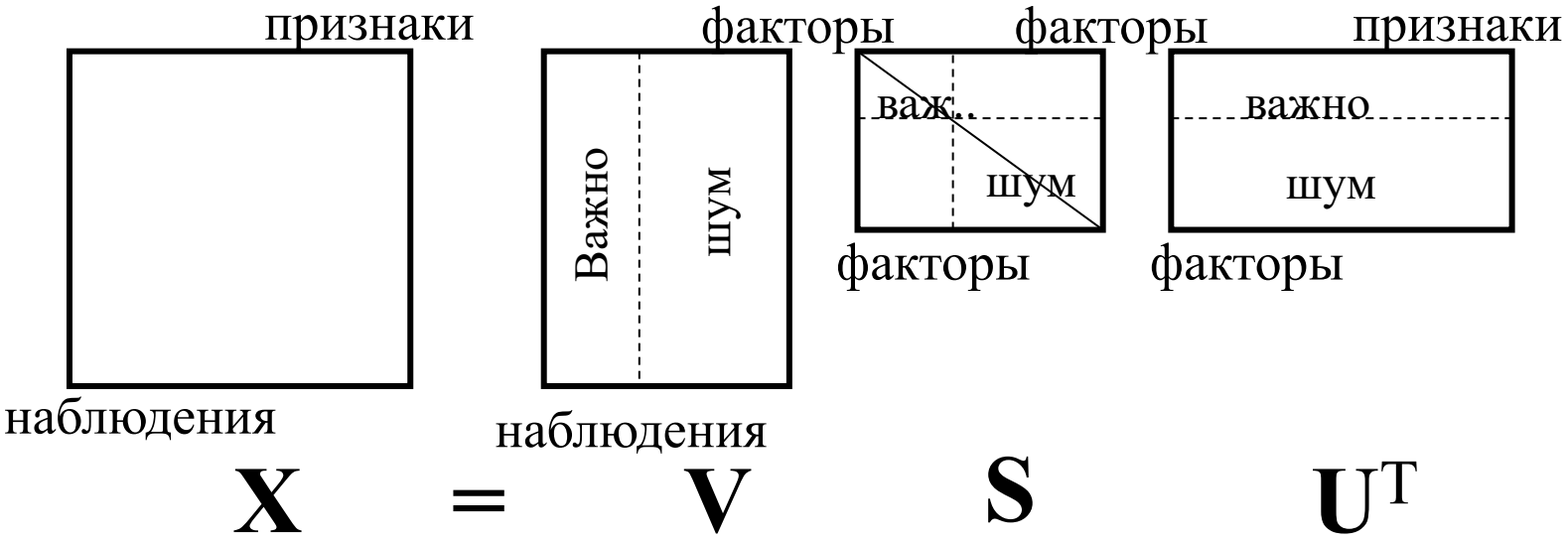
model2

	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		

Вычисление МНК

- Обычно на основе матричных разложений QR, SVD и др.
- Рассмотрим SVD:



$$\mathbf{X} = \mathbf{V}\ \mathbf{S}\ \mathbf{U}^{\mathrm{T}}$$

- По умолчанию число факторов равно числу признаков
- Орт. матрица $U^T U = I$ правых сингулярных векторов, с.в. $X^T X$
- Орт. матрица $VV^T = I$ левых сингулярных векторов, с.в. XX^T
- $S = diag(\sqrt{\lambda_1},\ldots,\sqrt{\lambda_p})$ – с.зн. XX^T и $X^T X$

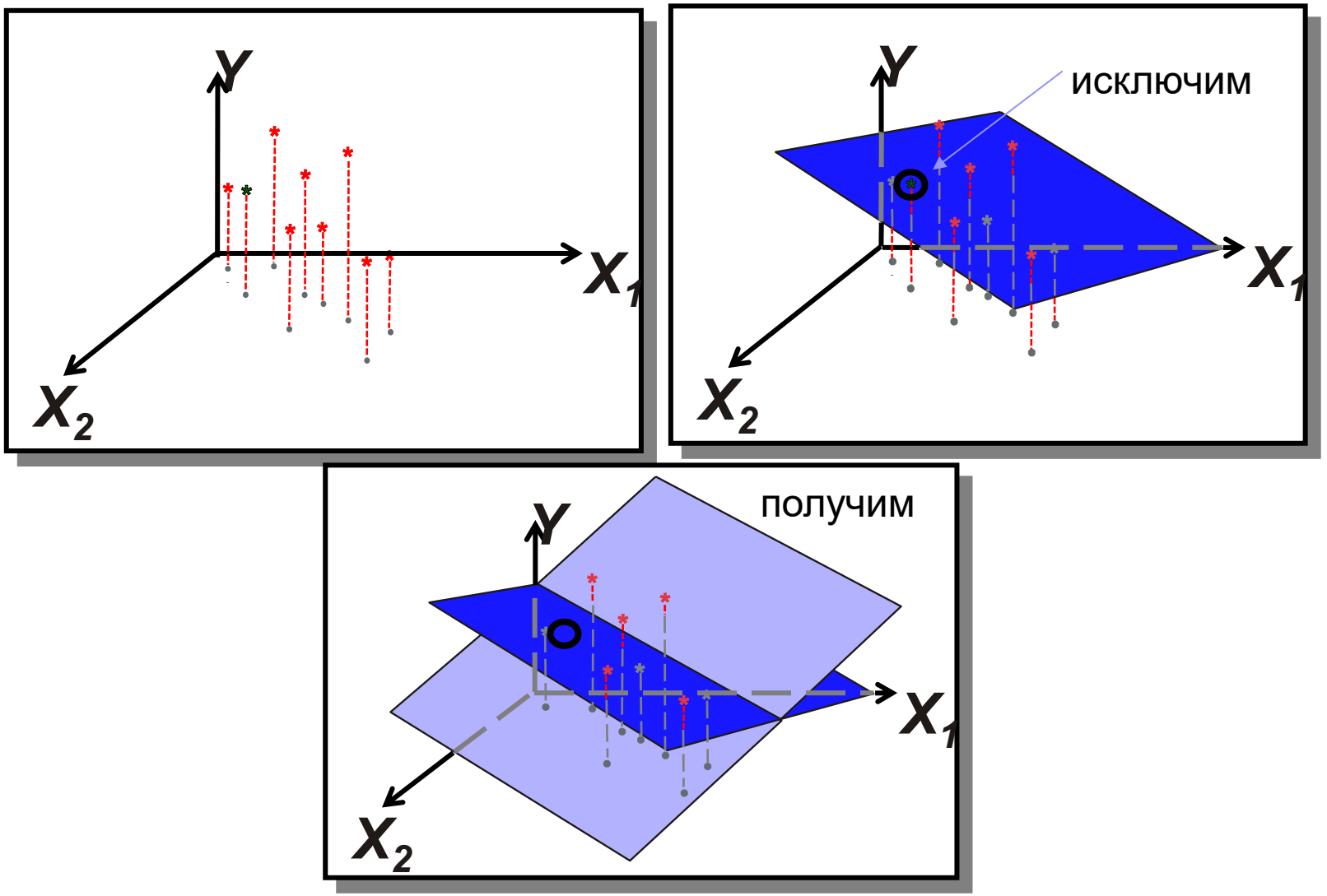
Вычисление МНК

- Псевдообратная матрия $X^+ = \left(X^T X\right)^{-1} X^T$:

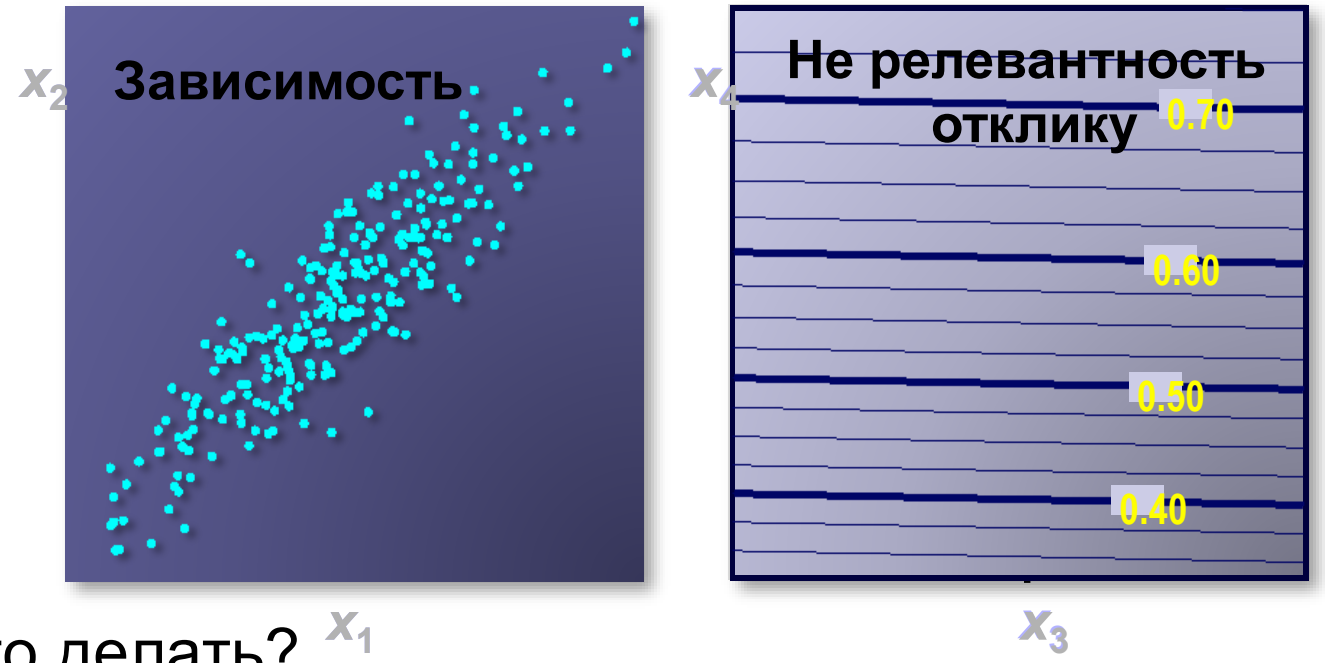
$$X^+ = \left(USV^T VSU^T\right)^{-1} USV^T = US^{-1}V^T = \sum_i \frac{1}{\sqrt{\lambda_i}} u_i v_i^T$$

- Решение: $w = X^+ y = \sum_i \frac{1}{\sqrt{\lambda_i}} u_i (v_i^T y)$
- Если есть корреляции между признаками в X, тогда:
  - $X^T X$ плохо обусловлена (велико $\lambda_{max}/\lambda_{min}$)
  - Увеличивается погрешность вычисления решения w и норма $\left|\left|w\right|\right|^2$
  - Решение неустойчиво и плохо интерпретируемо
  - Возникает переобучение
  - Портятся статистики с оценкой значимости переменных
  - Увеличивается вариативность оценки параметров и как следствие ошибка
  - Есть тенденция к неограниченному росту коэффициентов при завис. призн.

Иллюстрация неустойчивости при мультиколлинеарности



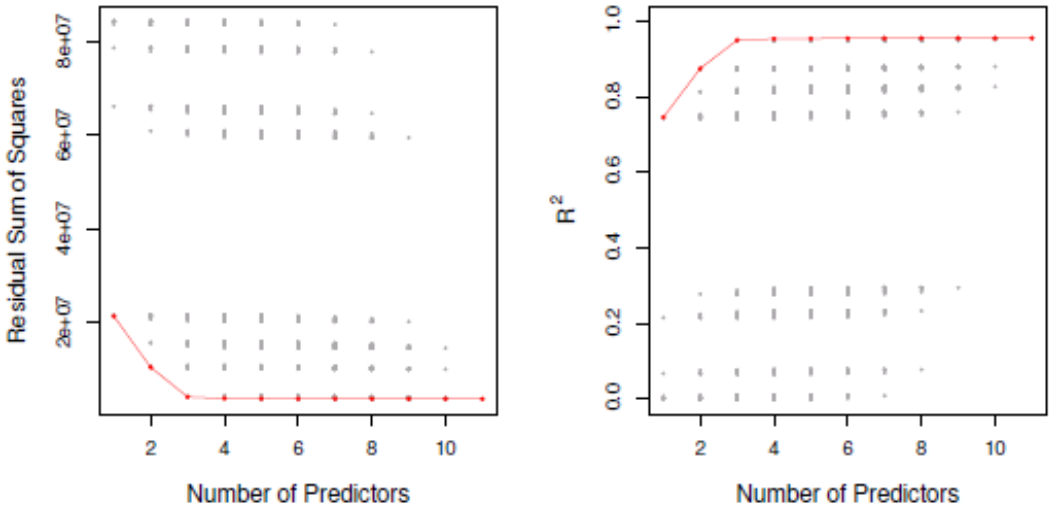
Проблемы входных переменных для МНК



- Что делать?
  - Отбирать признаки (пошаговые методы, прямой, обратный, комбинированные)
  - Использовать регуляризацию (штраф за сложность, гребневая регрессия, LASSO, Elastic Net)
  - Преобразовывать исходное пространство признаком (регрессия главных компонент и PLS)

Полный перебор переменных

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



- На практие – не всегда применимо для значительного числа переменных, экспоненциально растет число проверяемых моделей, поэтому жадный пошаговый перебор

Особенности пошаговых методов отбора

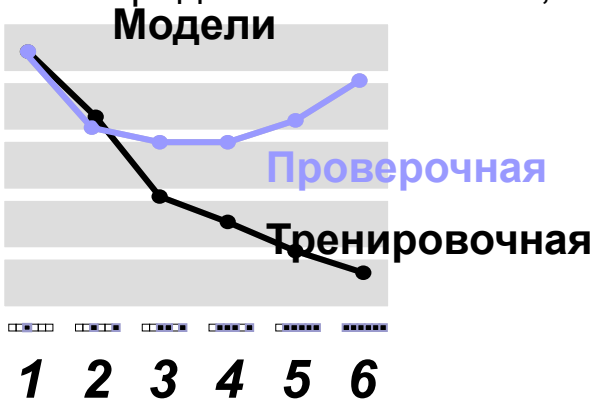
- Жадные алгоритмы:
  - не находят глобально лучшую модель даже с точки зрения эмпирического риска на тренировочном набор
- Необходимо определить 3 правила:
  - Правило для выбора следующего шага (выбор переменной для добавления и/или удаления)
  - Правило для останова (когда дальнейшее изменение модели не целесообразно)
  - Правило выбора лучшей модели из семейства (если не совпадает с правилом останова)

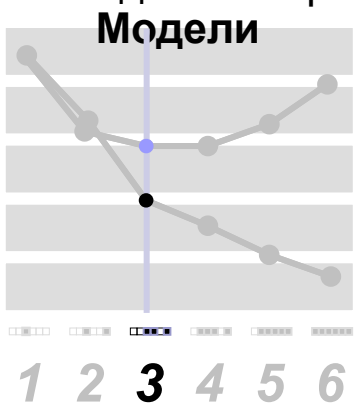
Правила на основе «лучшего» эмпирического риска на тренировочном наборе

- Суть правила:
  - наибольшее улучшение эмпирического риска для шага добавления переменных в модель и наименьшее ухудшение эмпирического риска для удаления переменных из модели
  - правило для останова можно задать, определив пороги: минимальный допустимый «прирост» при добавлении переменных или максимально допустимое «ухудшение» модели при удалении переменной
  - для линейной регрессии можно использовать выбор по RSS, коэффициенту детерминации, корреляции с остатками и т.д.
- Достоинство
  - простота и скорость расчета
- Недостатки:
  - не учитывается сложность модели, сложные всегда лучше на тренировочном наборе, значит не подходит для выбора лучшей модели
  - тяжело определить порог останова – нет общей шкалы, у каждой задачи будет своя, в зависимости от дисперсии отклика
  - не оценивается обобщающая способность получаемой модели

На основе лучшей обобщающей способности

- Суть правила:
  - Использование валидационной выборки, кросс-валидации или бутстреппинга для сравнения моделей
- Достоинство
  - Самый «правильный» метод с точки зрения теории, т.к. оценивается обобщающая способность получаемой модели
- Недостатки:
  - Большая вычислительная сложность, поэтому редко используется для выбора шага и определения останова, но часто для выбора лучше модели





Правила на основе статистического подхода – тест Стьюдента

- Суть правила:
  - Используем статистическую оценку важности переменной на основе критерия Стьюдента
  - При выборе предиктора-кандидата на удаление выбираем переменную с максимальным p-value для t-статистики
  - При выборе предиктора-кандидата на добавление выбираем с минимальным p-value для t-статистики
- Достоинство
  - Пороги для останова задаются для p-value на шкале [0,1] не зависимо от разброса отклика
- Недостатки:
  - Не учитывается сложность модели
  - Не оценивается обобщающая способность получаемой модели

Правила на основе статистического подхода – тест Фишера, тест Уальда

- Суть правила:
  - Оцениваем не отдельные переменные, а модели «до» и «после» шага добавления/удаления, например, по Уальду:

$$W = \frac{\left(RSS_{short} - RSS_{long}\right)}{RSS_{long}/l}, \sim \chi^2_{(p_{long} - p_{short})}$$

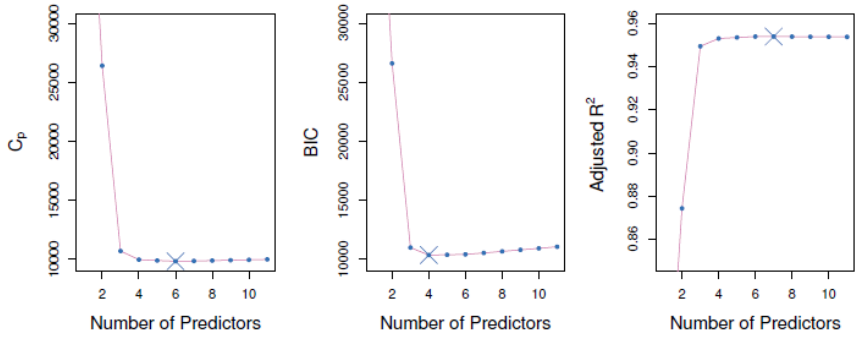
  - или сравниваем с самой плохой моделью (без предикторов) с помощью критерия Фишера:

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

- Достоинства:
  - Пороги для останова задаются для p-value на шкале [0,1]
  - Учитывается сложность модели
- Недостатки:
  - Напрямую не оценивается обобщающая способность модели

Правила на основе информационных критериев C_p, AIC, BIC и скорректированного R^2

- Суть правила:
  - Корректируют ошибку обучения с учетом сложности модели, и могут быть использованы для выбора лучшего шага, останова и лучшей среди множества моделей с различным числом предикторов



- Достоинства:
  - Учитывается сложность модели
  - Считается, что приблизительно оценивается обобщающая способность модели
- Недостатки:
  - Если используем в качестве правила останова, то прекращаем отбор, если критерий перестает улучшаться

C_p и AIC

- Mallow C_p :

$$C_p = \frac{1}{l}(RSS + 2p\sigma^2)$$

  - где p – число степеней свободы модели, обобщенное значение числа используемых параметров
  - σ^2 - оценка дисперсии ошибки ε, связанной с каждым измерением отклика.
- Критерий *AIC*, определяемый для более широкого класса моделей, рассчитывается методом максимального правдоподобия:

$$AIC = -2loglik + 2p$$

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

BIC и скорректированный R^2

- Байесовский информационный критерий:

$$BIC = \frac{1}{l}(RSS + \mathbf{log}(\boldsymbol{l})\, p\sigma^2)$$

- Аналогично *AIC* и C_p, но *BIC*, увеличивает штраф за сложность в $\log(l)$ раз, и при $l > 7$ сильнее штрафует модели, и приводит к выбору модели меньшего размера.
- Для модели МНК с *p* переменными скорректированная R^2 :

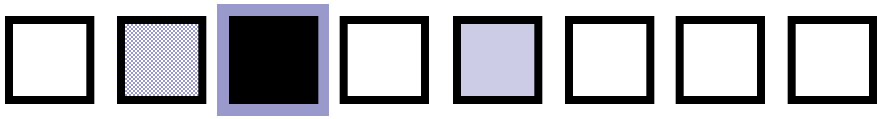
$$Adj\ R^2 = 1 - \frac{RSS(\boldsymbol{l-1})}{(\boldsymbol{l-p-1})TSS}$$

- В отличие от C_p, AIC и BIC, *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.

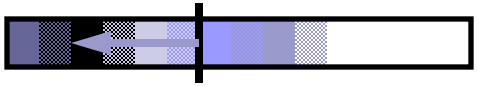
Прямой отбор

- Начинаем с *нулевой модели* содержащей только константу.
- Цикл:
  - Рассматриваем все варианты добавления одной из еще не добавленных переменных
  - Выбираем тот вариант, что соответствует выбранному правилу отбора (например, наименьшее значение RSS, или наименьшее p-value для критерия Фишера, Уальда или Стьюдента, наибольше уменьшение информационного критерия и т.д.)
  - Проверяем правило останова (по порогам для статистических критериев и правил на основе эмпирического риска или по ухудшению информационного критерия)
  - Если правило останова сработало, то выходим из цикла, иначе добавляем переменную в модель, переобучаем ее и переходим на следующую итерацию

Input *p*-value

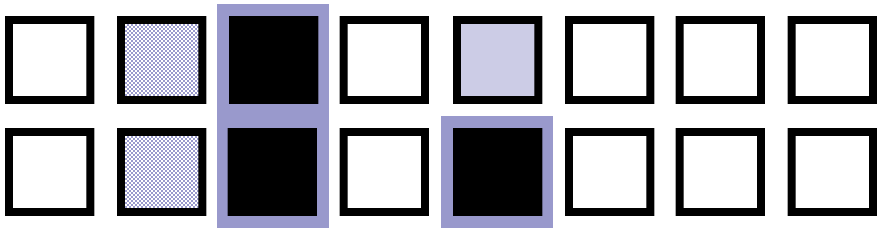


Entry Cutoff

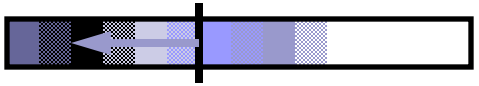


...

Input *p*-value

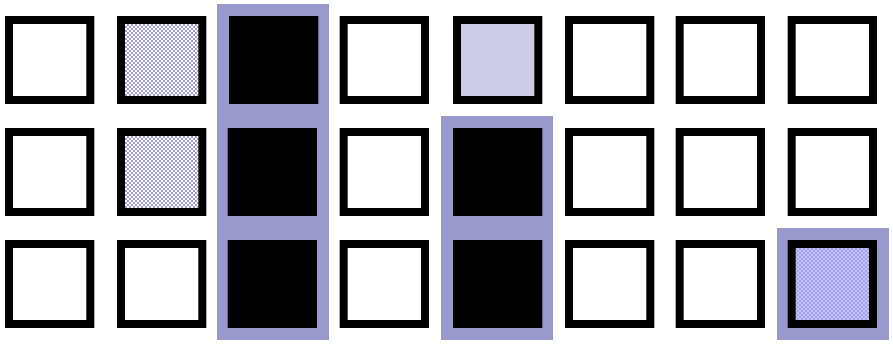


Entry Cutoff

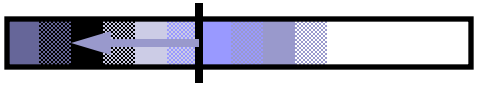


...

Input *p*-value

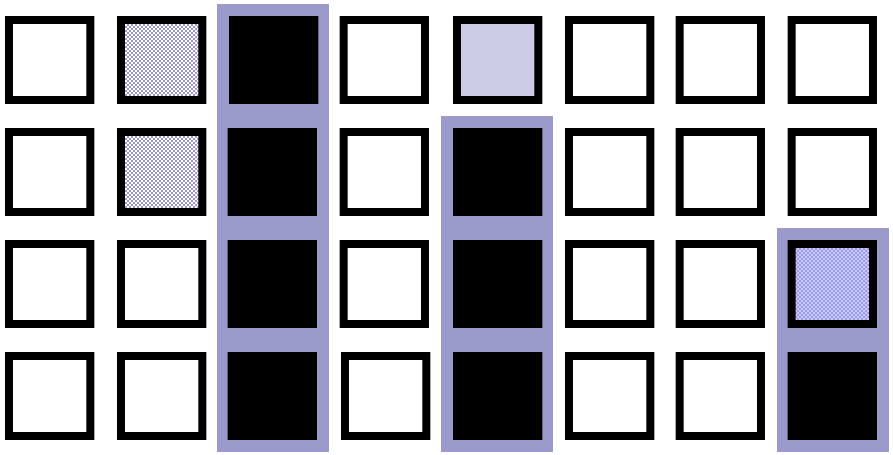


Entry Cutoff

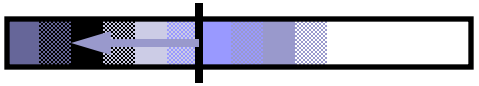


...

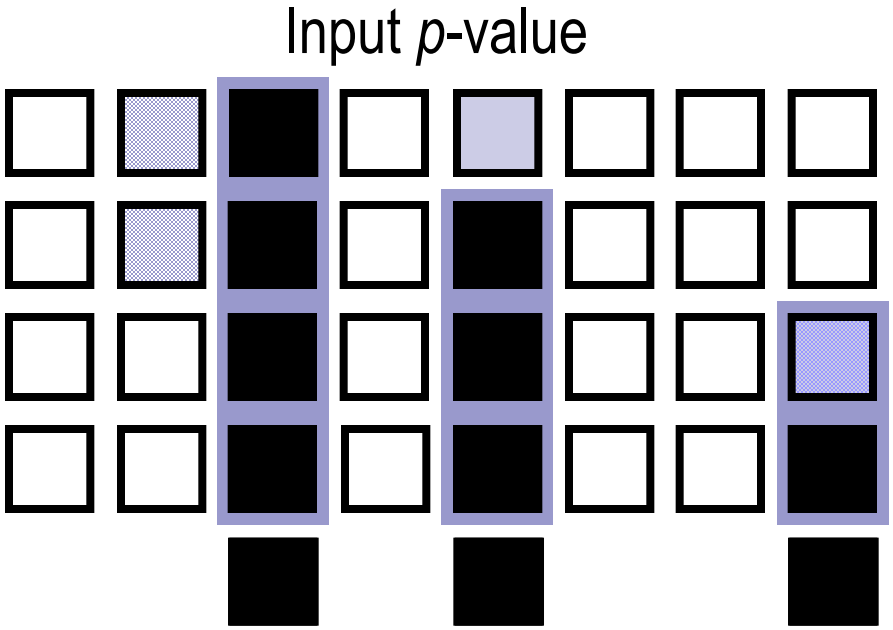
Input *p*-value

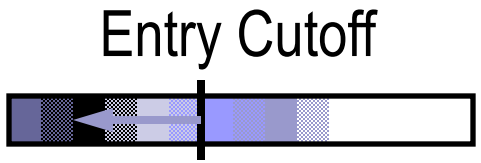


Entry Cutoff



...

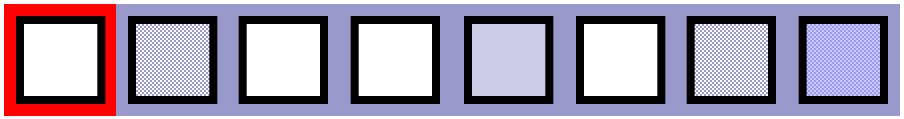

Input *p*-value


Entry Cutoff

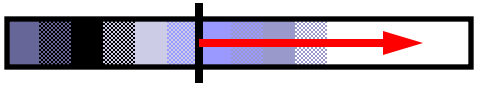
Обратный отбор

- Начинаем с *полной модели,* содержащей все переменный.
- Цикл:
  - Рассматриваем все варианты удаление одной из оставшихся переменных
  - Выбираем тот вариант, что соответствует выбранному правилу отбора (например, наибольшее значение RSS, или наибольшее p-value для критерия Фишера, Уальда или Стьюдента, наибольше уменьшение информационного критерия и т.д.)
  - Проверяем правило останова (по порогам для статистических критериев и правил на основе эмпирического риска или по ухудшению информационного критерия)
  - Если правило останова сработало, то выходим из цикла, иначе удаляем выбранную переменную из модели, перестраиваем ее и переходим на следующую итерацию

Input *p*-value

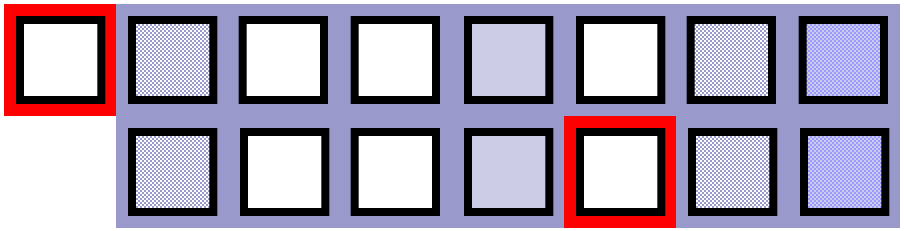


Stay Cutoff

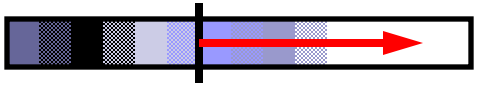


...

Input *p*-value

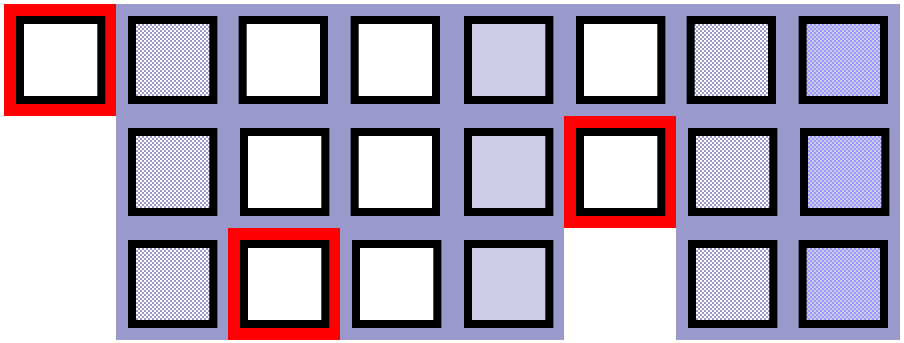


Stay Cutoff

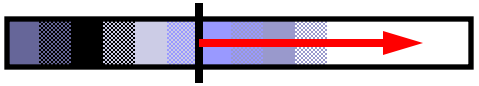


...

Input *p*-value

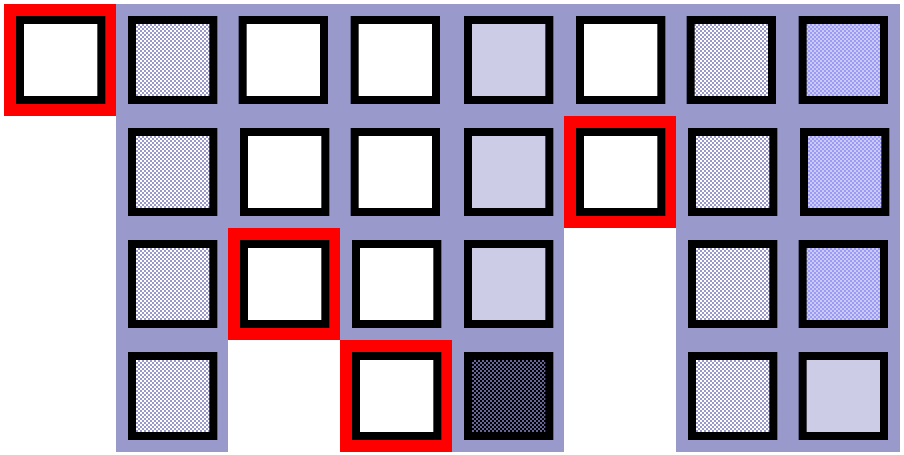


Stay Cutoff

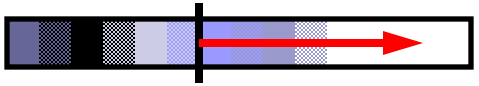


...

Input *p*-value

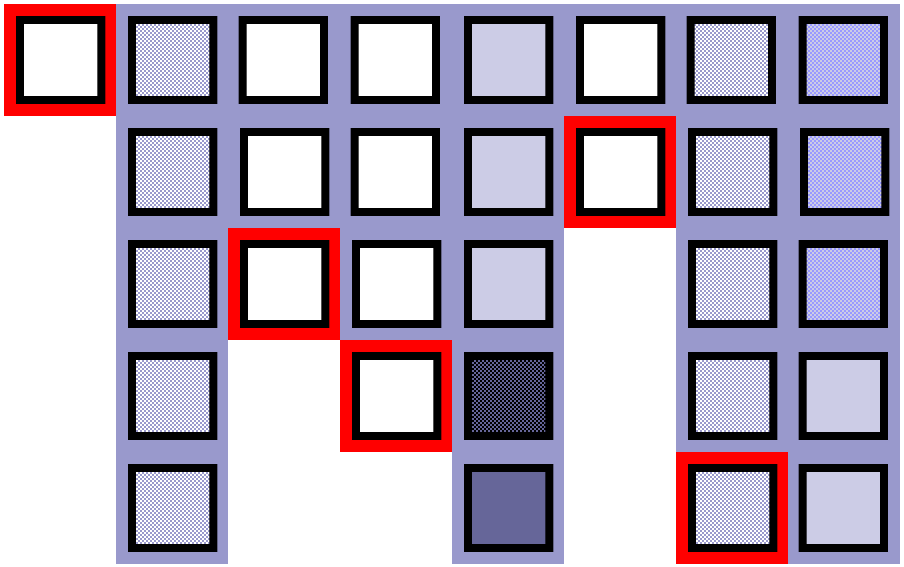


Stay Cutoff

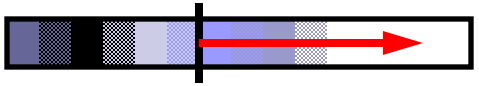


...

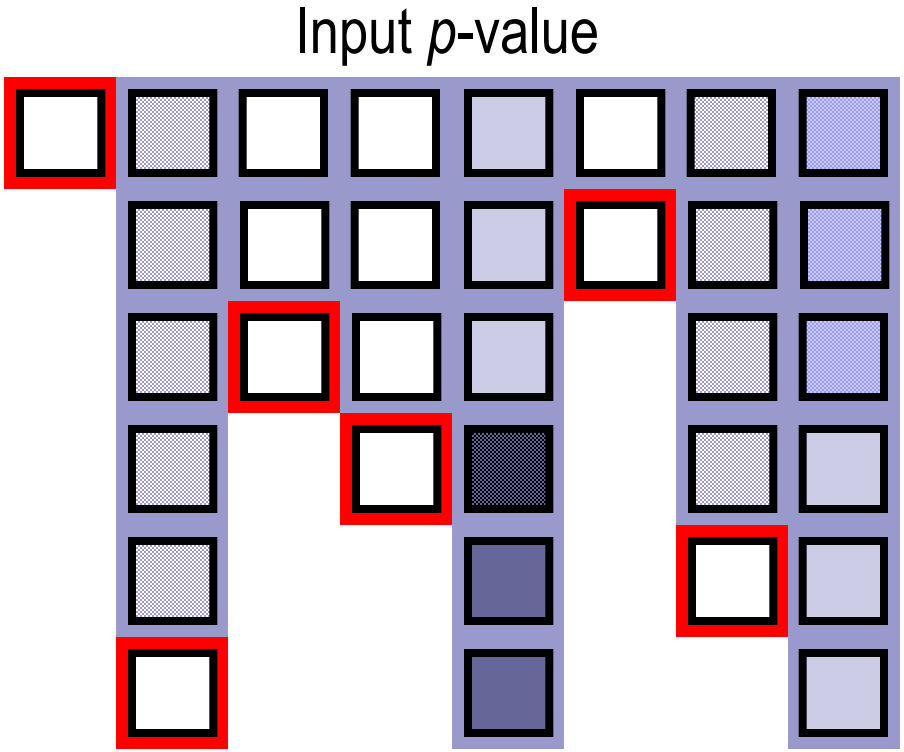
Input *p*-value

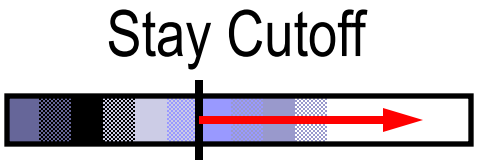


Stay Cutoff

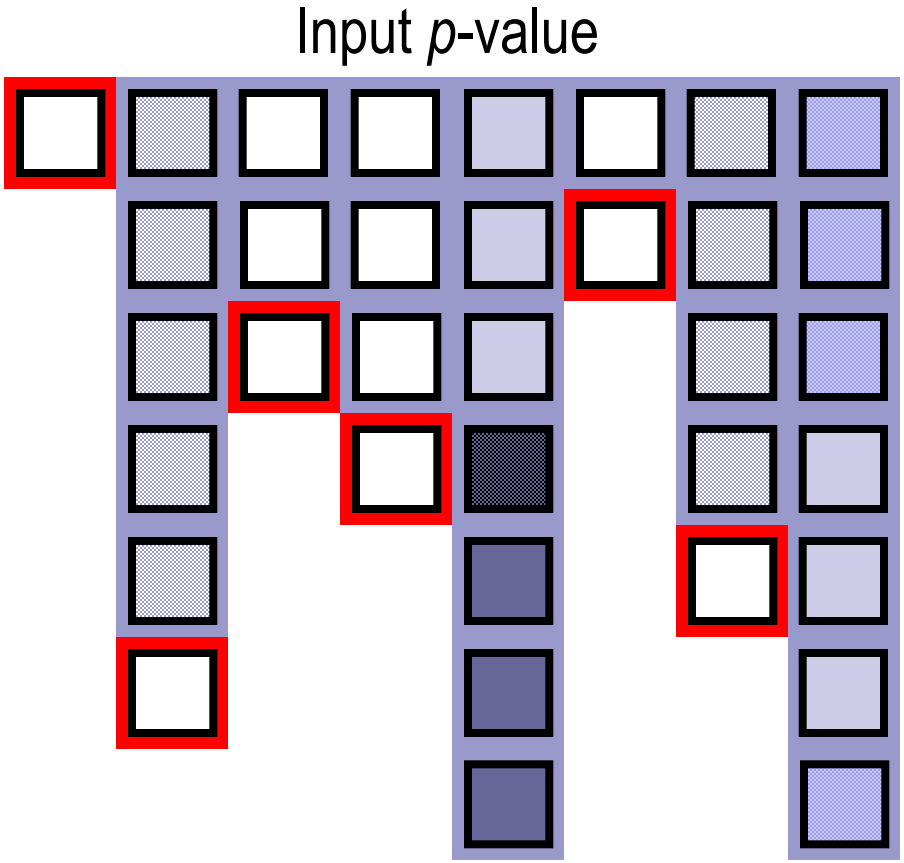


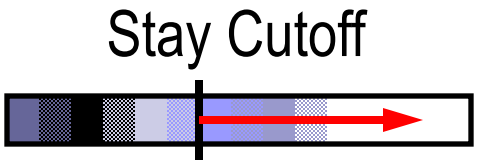
...


Input *p*-value

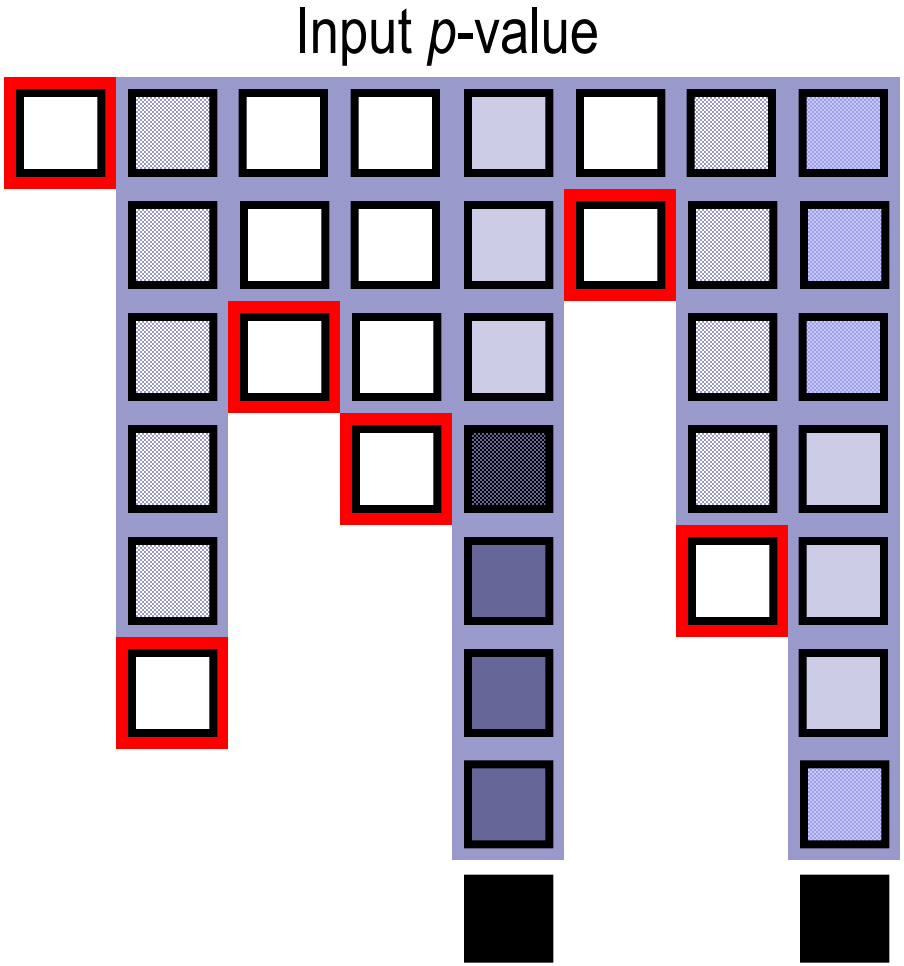

Stay Cutoff

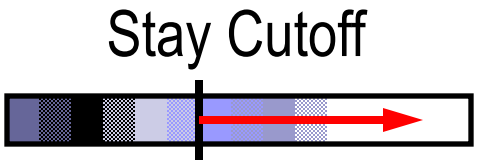
...


Input *p*-value


Stay Cutoff

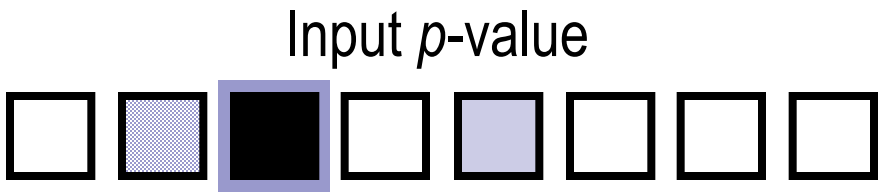
...

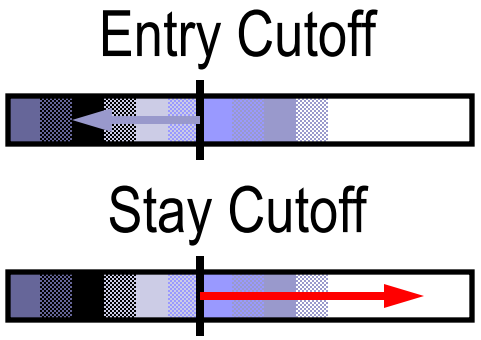

Input *p*-value


Stay Cutoff

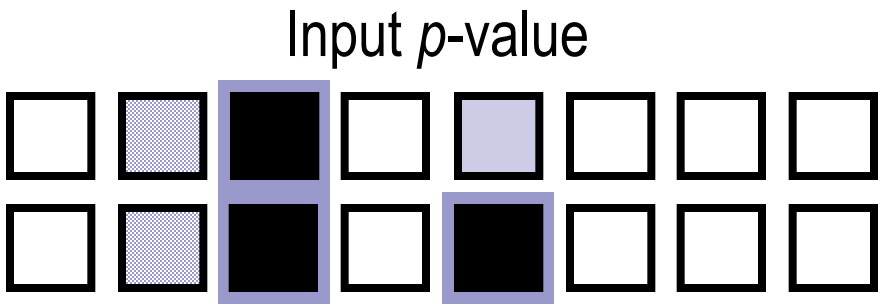
Комбинированный отбор

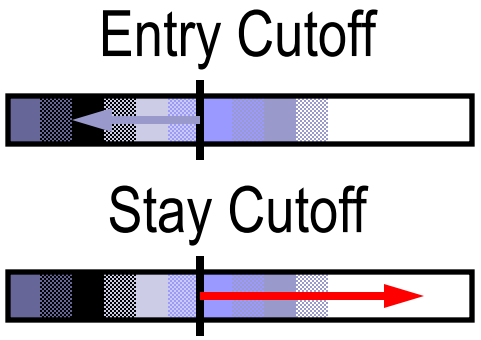
- Прямой и обратный методы – жадные (как и комбинированный) с не сильно гибким перебором, пытаемся улучшить
- Комбинированный метод (более гибкий перебор):
  - Пытаемся сделать шаг вперед (сначала из нулевой модели)
  - Затем пытаемся сделать шаг назад
  - Продолжаем 1 и 2 до тех пор, пока не сработает правило останова и для добавления, и для удаления переменных или пока не попали в «цикл» (последоватльное добавление и удаление одной и той же переменной)
- Есть варианты дополнительного «свопа»:
  - не просто пытаемся добавить, а потом удалить, а сначала ищем лучшую пару на замену (одну не добавленную с одной добавленной), если не находим, то обычный комбинированный шаг.


Input *p*-value

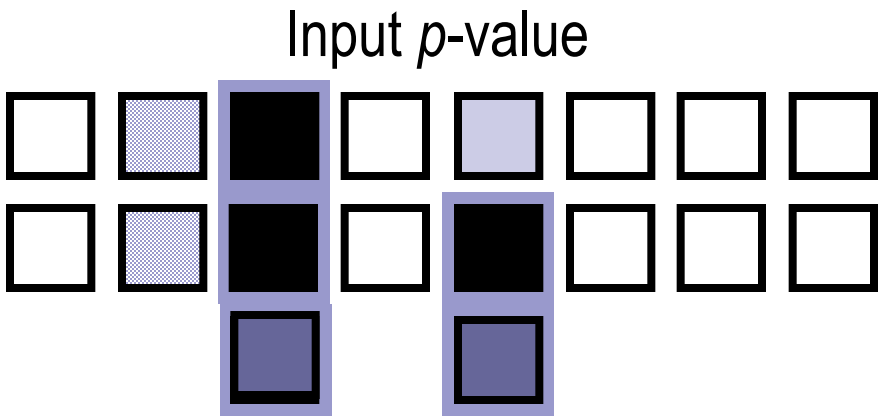

Entry Cutoff
Stay Cutoff

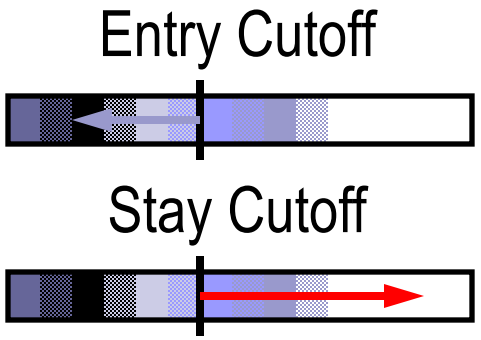
...


Input *p*-value

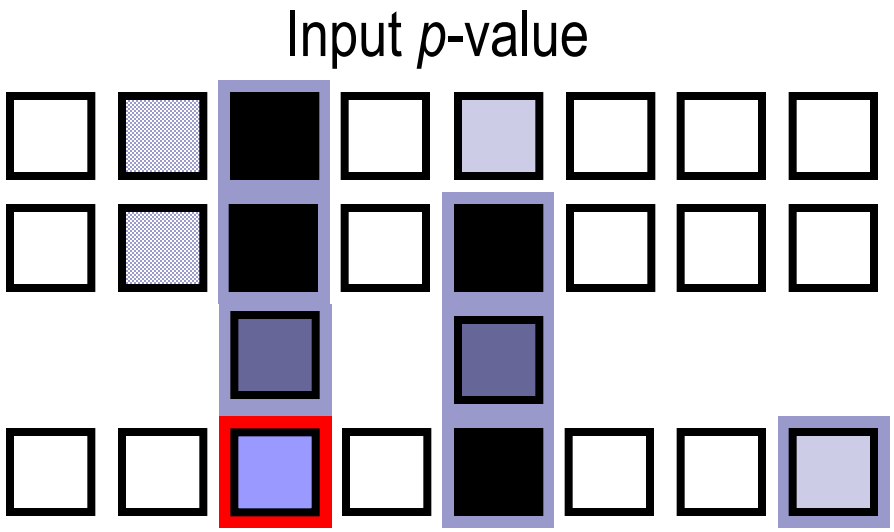

Entry Cutoff
Stay Cutoff

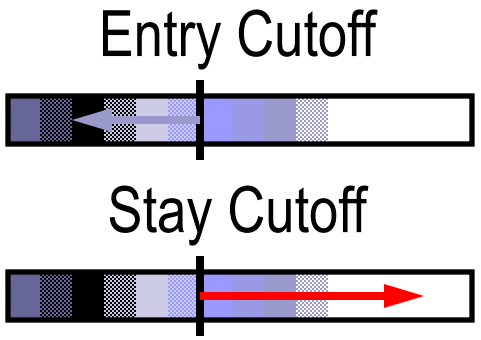
...


Input *p*-value

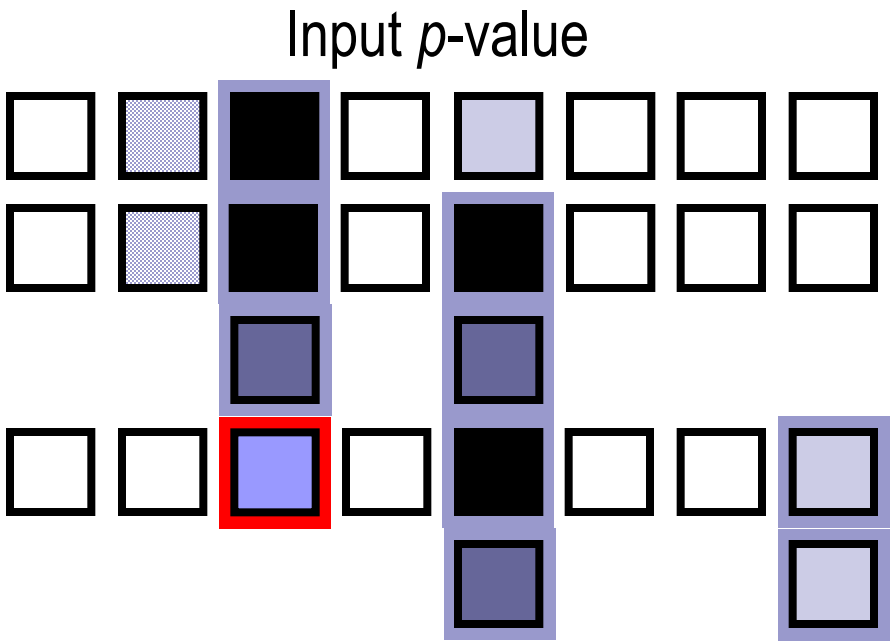

Entry Cutoff
Stay Cutoff

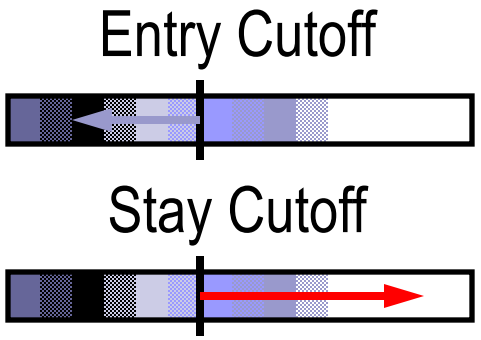
...


Input *p*-value

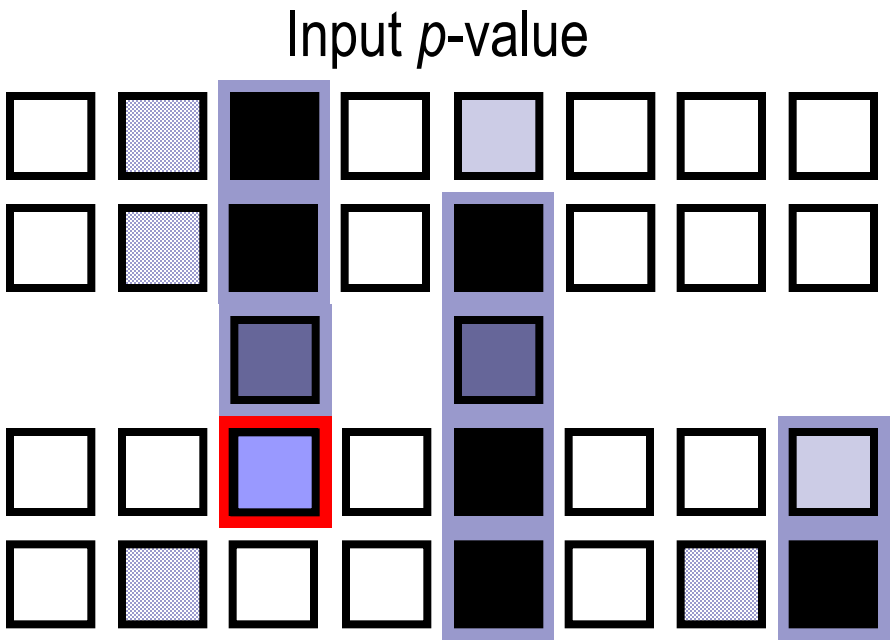

Entry Cutoff
Stay Cutoff

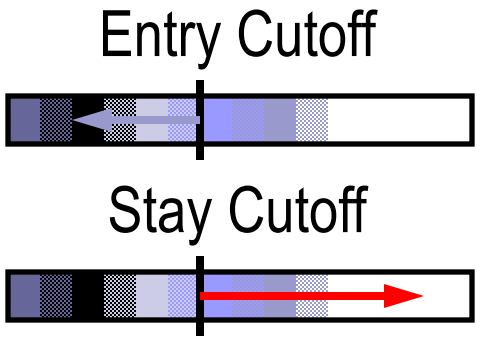
...


Input *p*-value

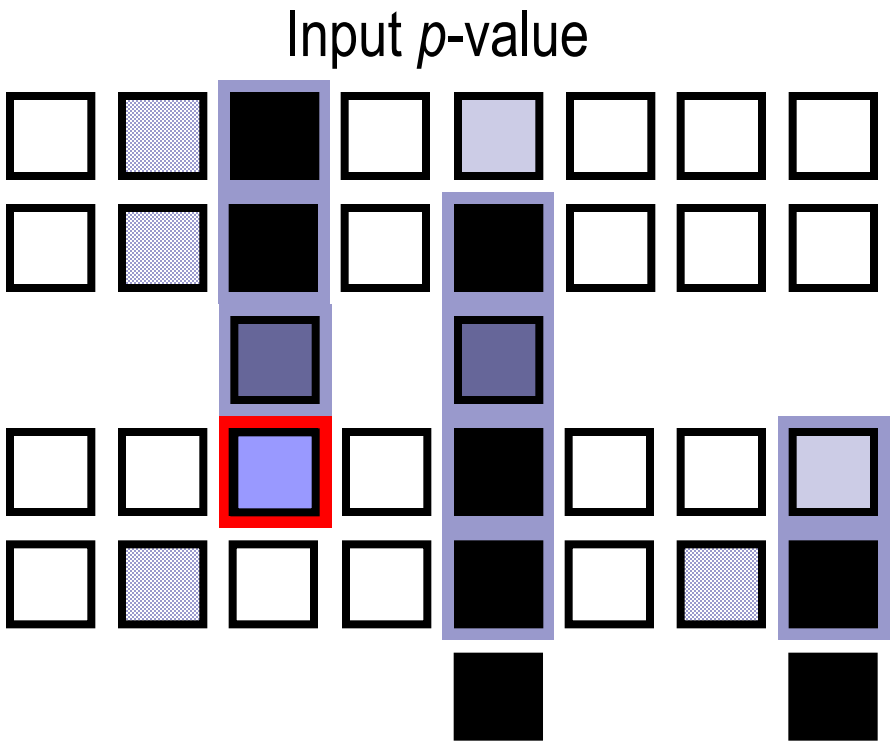

Entry Cutoff
Stay Cutoff

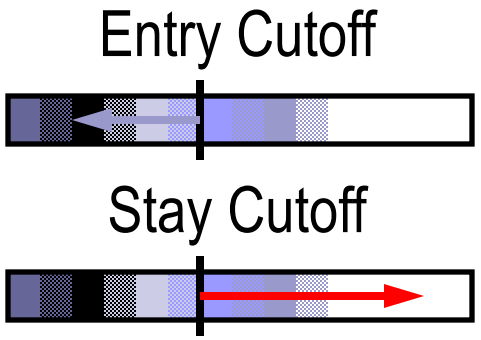
...


Input *p*-value


Entry Cutoff
Stay Cutoff

...


Input *p*-value


Entry Cutoff
Stay Cutoff

Пример прямой R^2

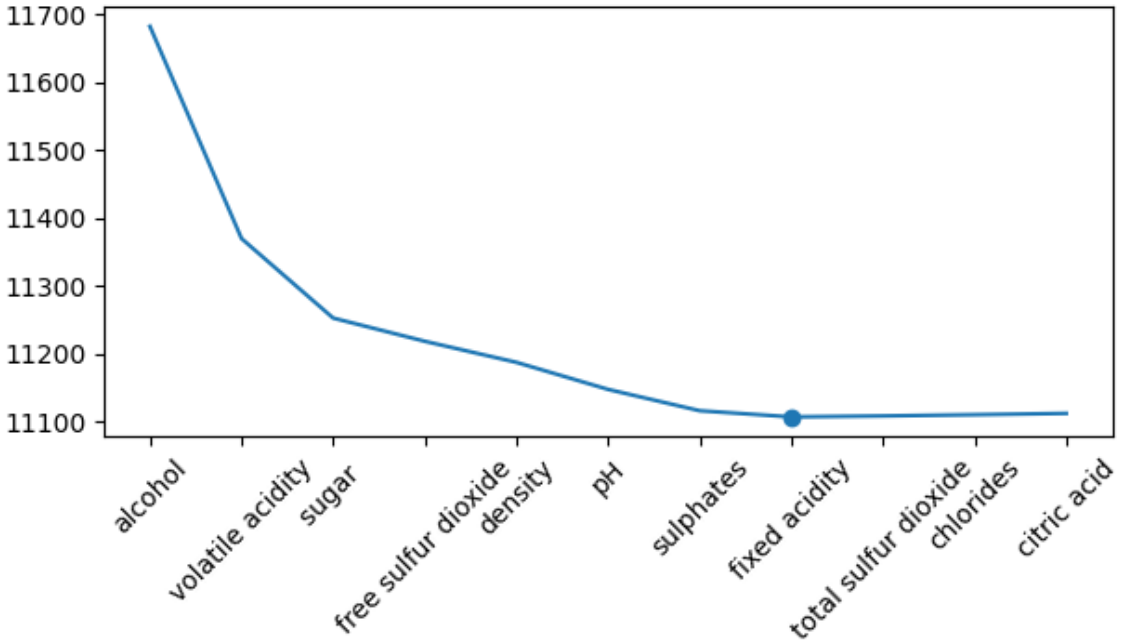
```python
def next_possible_feature(X, y, current_features,
                          by="rsquared", asc=False):
    feat_dict = {'feat': [], by:[]}
    for col in X.columns:
        if col not in (current_features):
            sub_X = X[current_features + [col]]
            sub_X = sm.add_constant(sub_X)
            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 прямой 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)
```



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)
```

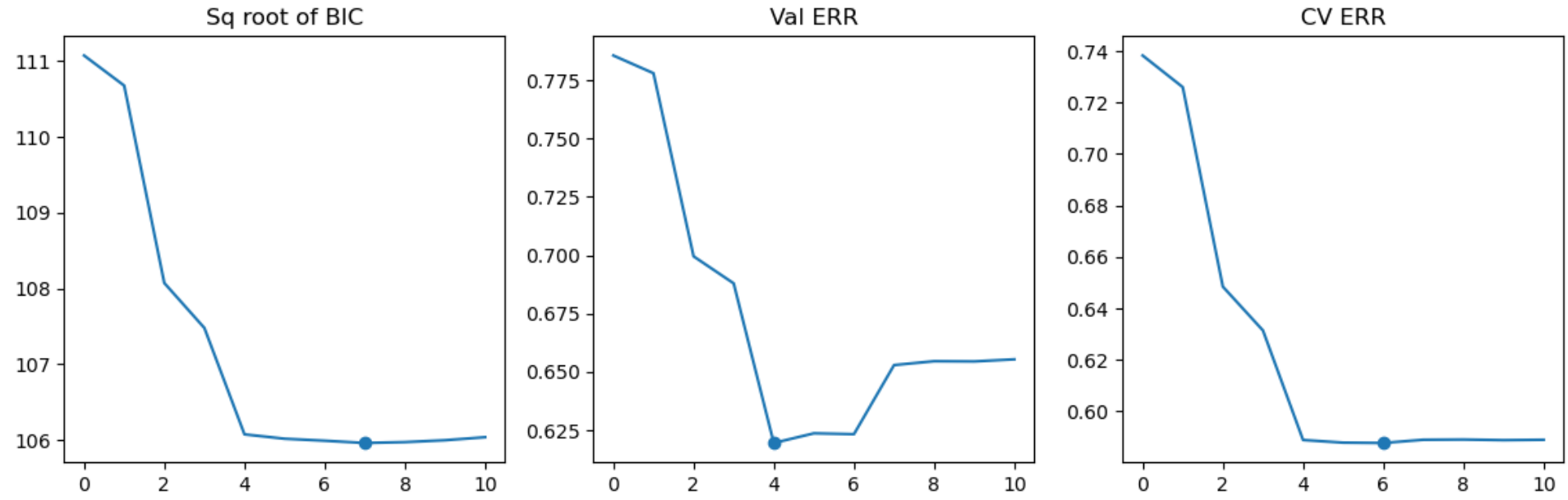
Пример

```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()
```



Валидация, кросс-валидация, бутстреппинг

- Имеет преимущество при выборе лучшей модели по сравнению с AIC, BIC, C_p и скорректированным R^2, т.к. обеспечивает непосредственную оценку тестовой ошибки, и *не требует оценки дисперсии ошибки*
- Также могут быть использованы в более широком диапазоне задач выбора модели, даже в случаях, когда трудно точно определить число степеней свободы модели (например, для непараметрических методов) или трудно оценить дисперсию ошибок

Методы регуляризации (штраф за сложность)

- Методы выбора подмножества используют МНК для линейной модели, которая содержит подмножество предикторов.
- В качестве альтернативы, можно построить модель потенциально содержащую все предикторы с использованием методики, которая *ограничивает или регуляризирует* оценки коэффициентов:

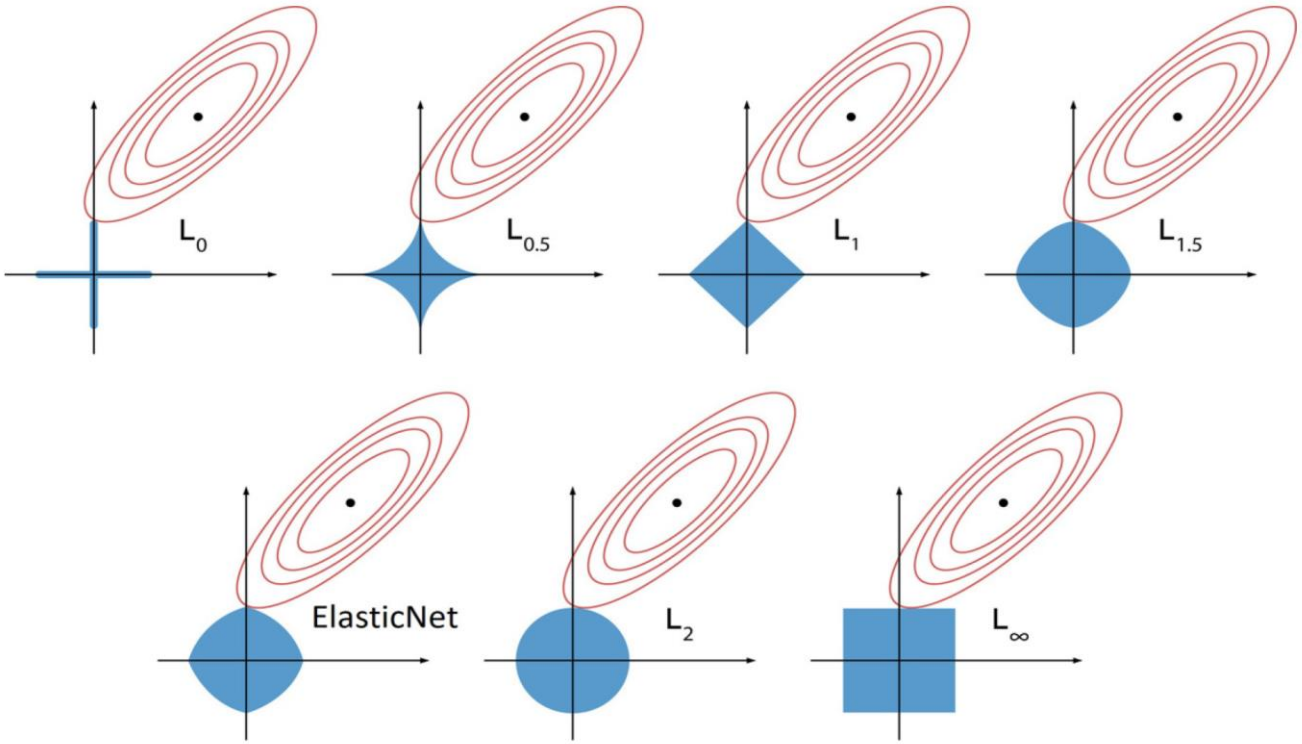
$$\min_{w}\left[\sum_{i=1}^{l}\left(y_i - w_0 - \sum_{j=1}^{p} x_{ij}w_j\right)^2 + \boldsymbol{\gamma R(w)}\right]$$

Штраф за сложность

- Может быть не сразу понятно, почему ограничение на абсолютные значения коэффициентов может улучшить модель, но оказывается:
  - оно может значительно уменьшить дисперсию модели
  - оно уменьшает (и даже устраняет) влияние мультиколинеарности, т.к. не дает неограниченно расти коэффициентам при зависимых переменных

Регуляризация L_p

$$\min_w \left[RSS(w) + \gamma L_p(w)\right] \Leftrightarrow \begin{cases} \min_w [RSS(w)] \\ L_p(w) \le C \end{cases}$$



Масштабирование предикторов

- Оценки коэффициентов стандартным МНК являются масштабируемой - умножение *предиктора* на константу просто приводит к масштабированию оценок коэффициентов МНК.
- Оценки коэффициентов с регуляризацией наоборот - *не масштабируемые*, т.е. могут *существенно* измениться при умножении заданного предиктора на константу.
- Поэтому лучше всего применять регуляризацию:
  - либо после нормирования признакового пространства, например:

$$x_i' = \frac{x_i - E(x_i)}{SE(x_i)}$$

  - либо в штрафе (регуляризаторе) нормировать сами коэффициенты, например, делить их на соответствующие коэффициенты модели МНК w_{OLS} без регуляризации:

$$R(w) = L_p(|w|/|w_{OLS}|)$$

Гребневая регрессия (L_2)

- Целевая функция:

$$\min_w \left[\sum_{i=1}^{l} \left(y_i - w_0 - \sum_{j=1}^{p} x_{ij} w_j \right)^2 + \boldsymbol{\gamma} \|\boldsymbol{w}\|^2 \right]$$

  - Гребневая регрессия (как и МНК) стремится найти коэффициенты регрессионного уравнения, которые минимизируют RSS, но, второе слагаемое (штраф), мал при коэффициентах, близких к нулю
  - Метапараметр регуляризации $\gamma \geq 0$ управляет относительным влиянием этих двух слагаемых на оценки коэффициентов.

- Можно показать, что:

$$\nabla_w \left[RSS(w) + \gamma w^T w \right] = -2X^T(y - Xw) - 2\gamma w = 0 \Rightarrow$$

$$w_\gamma = (\boldsymbol{X^T X + \gamma I})^{-1} X^T y$$

Никогда не вырождена

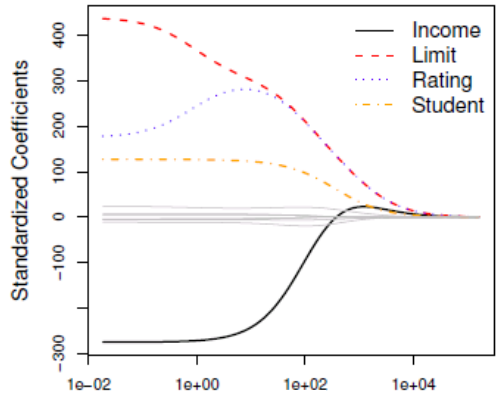
- Аналогично через SVD разложение получаем:

$$w_\gamma = \sum_i \frac{\sqrt{\lambda_i}}{(\lambda_i + \gamma)} u_i (v_i^T y)$$

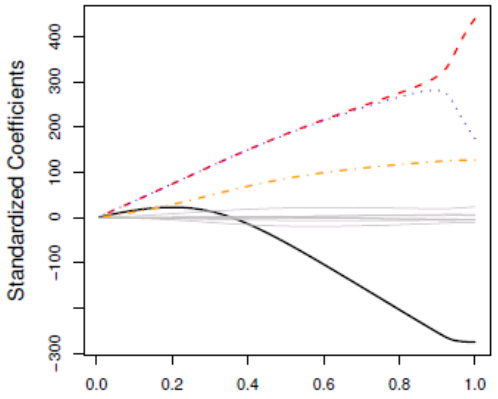
Влияние параметра регуляризации в L_2

- По SVD разложению также можно показать:

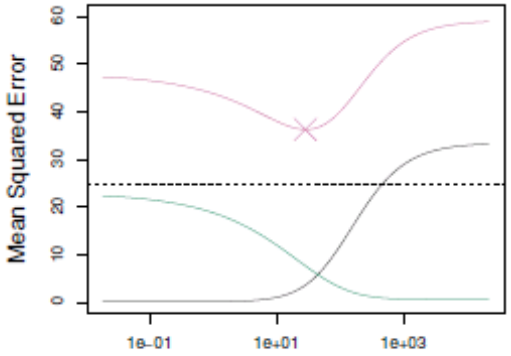
$$\|w_\gamma\|^2 = \sum_i \frac{\lambda_i}{(\lambda_i + \gamma)^2} (v_i^T y)^2 \leq \|w_{OLS}\|^2 = \sum_i \frac{1}{\lambda_i} (v_i^T y)^2$$



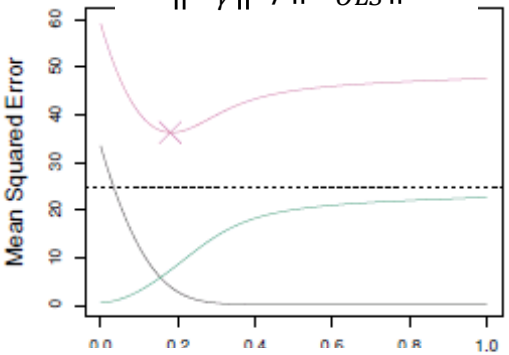
γ



$\|w_\gamma\|^2 / \|w_{OLS}\|^2$



γ



$\|w_\gamma\|^2 / \|w_{OLS}\|^2$

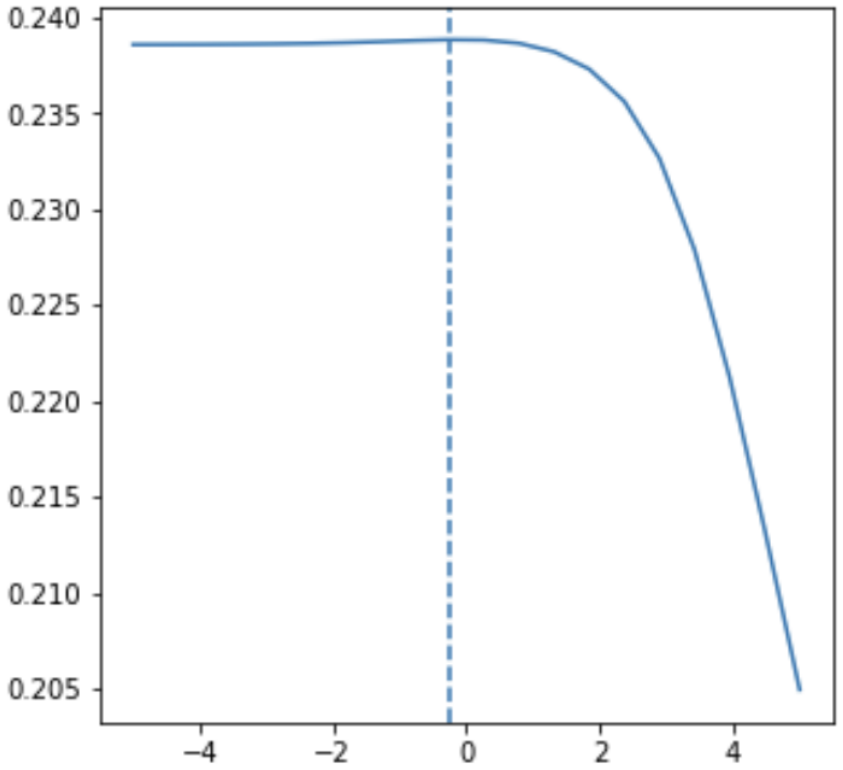
Сокращение «эффективной размерности»:

$$tr[(X^T X + \gamma I)^{-1} X^T] \leq tr[(X^T X)^{-1} X^T]$$

- Трассы коэффициентов в зависимости от параметра регуляризации
- MSE декомпозиция в зависимости от параметра регуляризации:
  - MSE на тестовой выборке
  - Дисперсия модели
  - Смещение модели

Пример Ridge

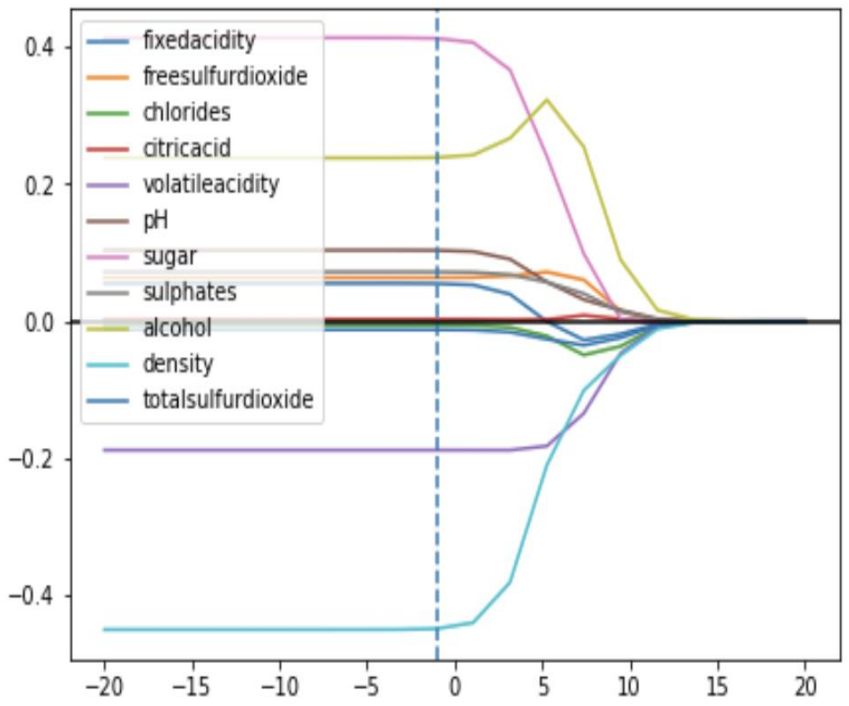
```
from sklearn.linear_model import Ridge

degree = np.linspace(-5, 5, 20)
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=(5, 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 параметра регуляризации (шкала логарифмическая, варьируется степень экспоненты).

Пример Ridge

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

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

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



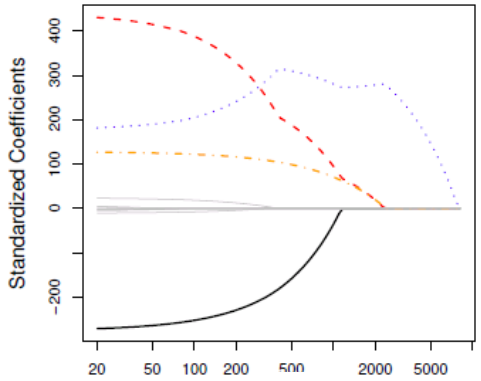
Оценки коэффициентов в зависимости от параметра регуляризации. Вертикальная пунктирная линия обозначает лучшее значение параметра регуляризации, полученное в результате перекрестной проверки.

Метод LASSO (L_1)

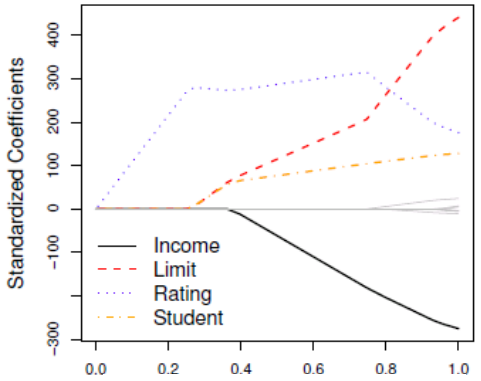
- Гребневая регрессия имеет важный недостаток - включает все предикторы, не осуществляет отбор
- *LASSO* (Least Absolute Shrinkage and Selection Operator) - преодолевает этот недостаток за счет регуляризации L_1
- Целевая функция:

$$\min_{w} \left[\sum_{i=1}^{l} \left(y_i - w_0 - \sum_{j=1}^{p} x_{ij} w_j \right)^2 + \boldsymbol{\gamma} \sum_{\boldsymbol{j=1}}^{\boldsymbol{p}} |\boldsymbol{w_j}| \right]$$

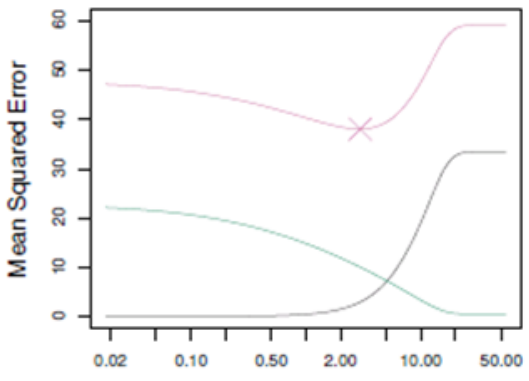
- Обнуление «не важных» коэффициентов: MSE разложение:



γ



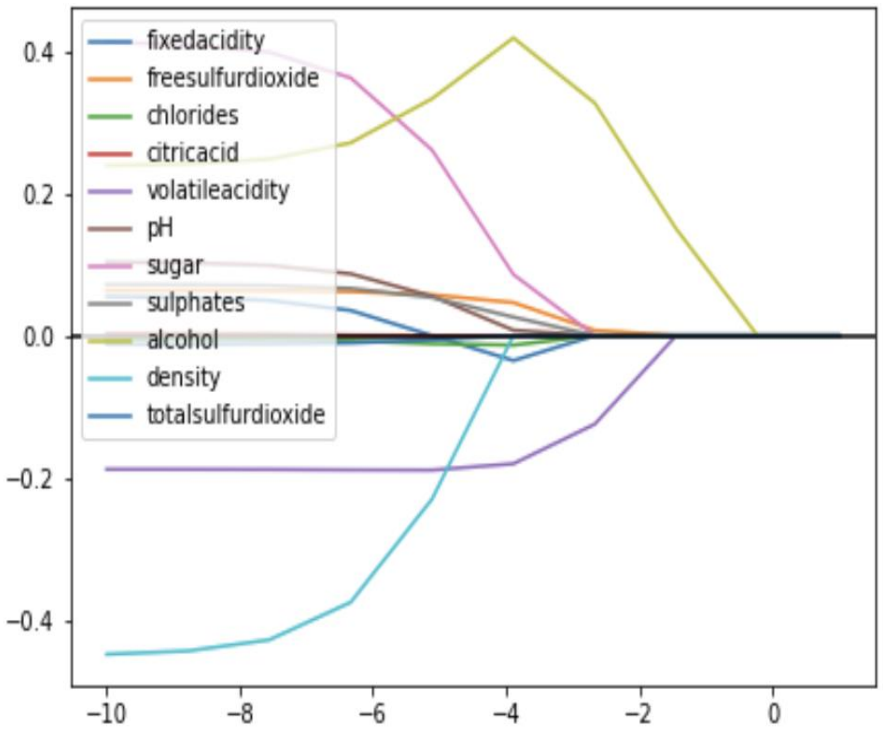
$\|w_\lambda\|_1 / \|w_{OLS}\|_1$



γ

Пример LASSO

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

scaler = StandardScaler()
s_X = scaler.fit_transform(X)

coefs = []
degree = np.linspace(-10, 1, 10)
alphas = np.exp(degree)
for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(s_X, y)
    coef = lasso.coef_.reshape(-1, 1)
    coefs.append(coef.reshape(1, -1))

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



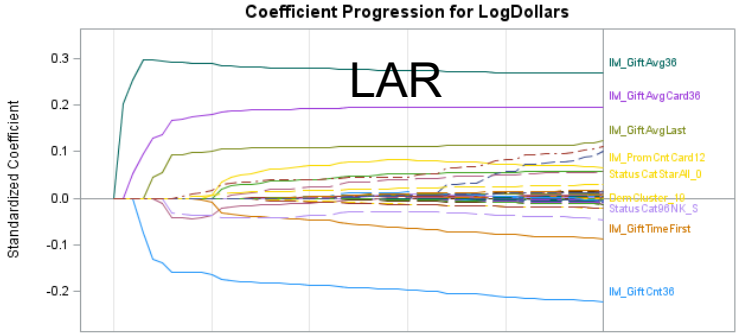
Оценки коэффициентов в зависимости от параметра регуляризации.

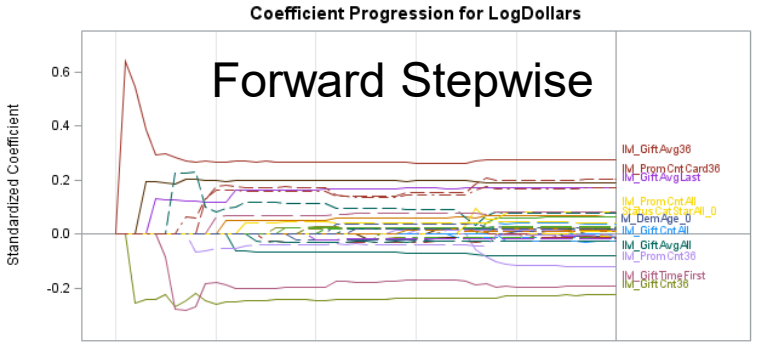
Поиск решения и выбор параметра регуляризации

- Параметр регуляризации для регрессий с L_1 и L_2 можно выбирать **по сетке** *перекрестной проверкой* или на основе *информационных критериев* AIC или BIC
- Для линейной регрессии с L_2 есть точное решение в *матричном виде* через разложения
- Для линейной регрессии с L_1 :
  - точного решение в матричном виде нет, задача сводится к задаче условной оптимизации (квадратичного программирования), но при наличии корреляций признаков решение не единственное
  - *ElasticNet* технически сводится к той же постановке задачи, что и L_1
  - *используются численные методы* (часто - покоординатный спуск, он сходится быстрее чем стандартный градиентный и требует меньше вычислений чем стандартный ньютоновский)
  - Относительно недавно удалось найти новый эффективный метод одновременно для поиска решения L_1 и перебора параметра регуляризации на основе регрессии наименьшего угла – **без сетки**

Метод наименьшего угла (least angle regression)

- *Прямой пошаговый* метод с *заглядыванием* на шаг вперед - на каждом шаге выбирается предиктор-кандидат на следующий шаг
- *Нежадный* пересчет коэффициентов - на каждом шаге, не полностью минимизируется целевая функция, остается часть вариации неописанной, так, чтобы ее можно было описать при добавлении следующего предиктора,
- Коэффициенты уже добавленных переменных не пересчитываются заново на каждом шаге, а меняются *пропорционально* (биссектрисе угла между ними), что не дает неограниченно расти коэффициентам при коррелированных предикторах
- После добавления всех предикторов модель эквивалента МНК
- Модели на каждом шаге позволяют найти ограничение на параметр регуляризации LASSO и на следующих шагах это ограничение не уменьшается





Метод наименьшего угла (алгоритм)

- Шаг 1. Стандартизация данных (включая отклик) :

$$\sum_i y_i = 0\,, \sum_i x_{ij} = 0\,, \sum_i x_{ij}^2 = 1\,, w_j = 0, i = \overline{1,l}, j = \overline{1,p}$$

- Шаг 2. Находим x_{j_1} наиболее коррелирующий с y:

$$j_1 = \underset{j}{\operatorname{argmax}} |\langle x_j, y \rangle|$$

- Шаг 3. Делаем максимальный шаг (находим коэффициент) в направлении x_{j_1}, пока не найдется другой x_{j_2} - кандидат, имеющий такую же корреляцию с еще не описанным остатком
- Пока не добавим все p предикторов повторяем Шаг (k): Добавляем кандидата x_{j_k} и ищем следующее направление пересчета коэффициентов − биссектриса угла между всеми добавленными $\{x_{j_1}, x_{j_2}, \dots, x_{j_k}\}$, пока не найдется $x_{j_{k+1}}$,также коррелирующий с остатками модели от $\{x_{j_1}, x_{j_2}, \dots, x_{j_k}\}$

Метод наименьшего угла (основной шаг)

- k- номер итерации,

  r_k - регрессионные остатки ($r_0 = y$)

  $\mathbf{E}_k$ - матрица единичных векторов координат уже добавленных переменных

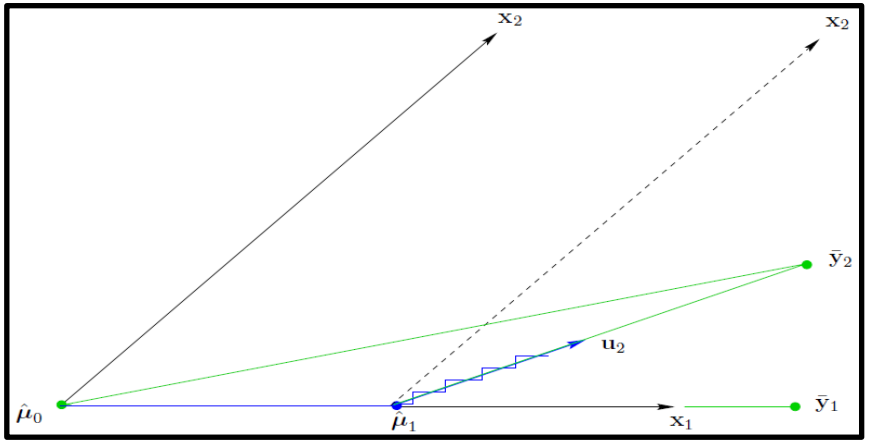
  $\mathbf{Z}_k = \mathbf{X}\mathbf{E}_k$ - ограниченная матрица данных (по добавленным переменным)

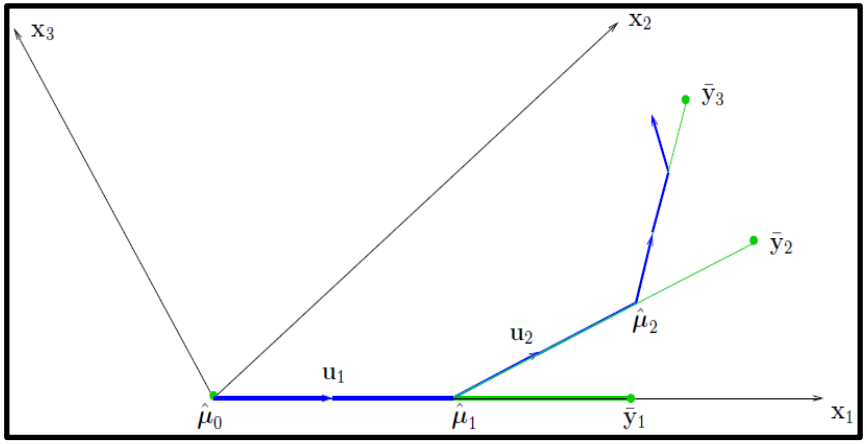
  $\mathbf{H}_k = \mathbf{Z}_k(\mathbf{Z}_k^T\mathbf{Z}_k)^{-1}\mathbf{Z}_k^T$ аналогично ограниченная матрица проекции
- направление пересчета $u_k = \mathbf{H}_k r_{k-1}$,
- пересчет коэффициентов $w^{(k)} = w^{(k-1)} + \mu u_k$
- для еще не выбранного x_j находим:

$$\mu_{k,j}^{+} = \frac{\langle r_{k-1}, x_{j_k}\rangle - \langle r_{k-1}, x_j\rangle}{\langle r_{k-1}, x_{j_k}\rangle - \langle \mathbf{X}u_k, \mathbf{H}_k x_j\rangle}, \mu_{k,j}^{-} = \frac{\langle r_{k-1}, x_{jk}\rangle + \langle r_{k-1}, x_j\rangle}{\langle r_{k-1}, x_{j_k}\rangle + \langle \mathbf{X}u_k, \mathbf{H}_k x_j\rangle}$$

- выбираем такой x_j и $\mu_k = \min_j\{\mu \in [0,1]: (\mu = \mu_{k,j}^{+}) \vee (\mu = \mu_{k,j}^{-})\}$
- пересчет остатков: $\langle r_k(\mu_k), x_j\rangle = \langle r_{k-1}, x_j\rangle - \mu_k\langle r_{k-1}, \mathbf{H}_k x_j\rangle$

2D и 3D примеры



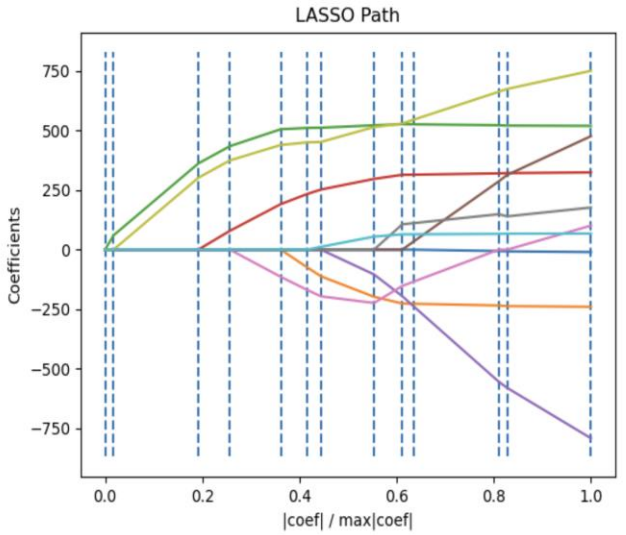


- j – номер шага
- x_j - переменные
- u_j - направление пересчета коэффициентов
- μ_j - длина шага
- $\bar{y}_j$ - проекция отклика (с учетом текущей проекционной матрицы)

Поиск решения для LASSO на основе LAR

- при переборе кандидатов на добавление и расчете длины шага легко учесть условие $\sum\left|w^{(k)}\right| \leq C$
- процесс добавления переменных на каждом шаге k можно рассматривать при условии $\sum\left|w^{(k)}\right| \leq C_k$, и получить невозрастающую последовательность параметров регуляризации $\{C_0 = \infty, C_1, C_2, \ldots, C_k, \ldots, C_p = 0 | C_k \geq C_{k+1}\}$

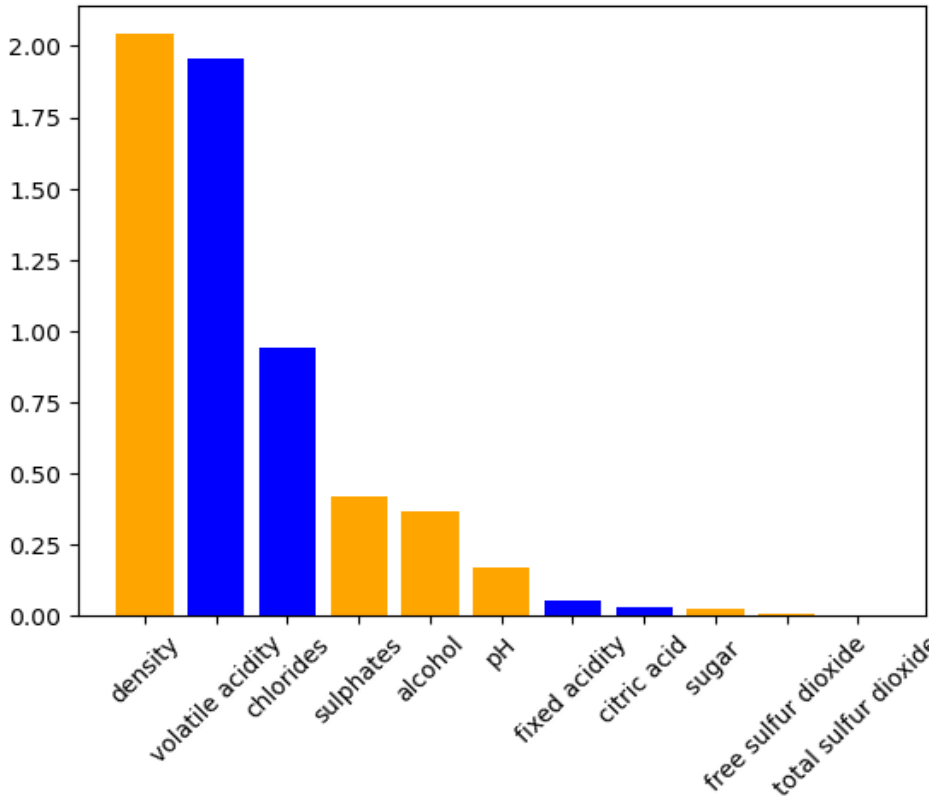
Таким образом мы сразу получаем все варианты для процедуры перекрёстной проверки или расчета информационных критериев, не нужно делать свою сетку



Пример LAR

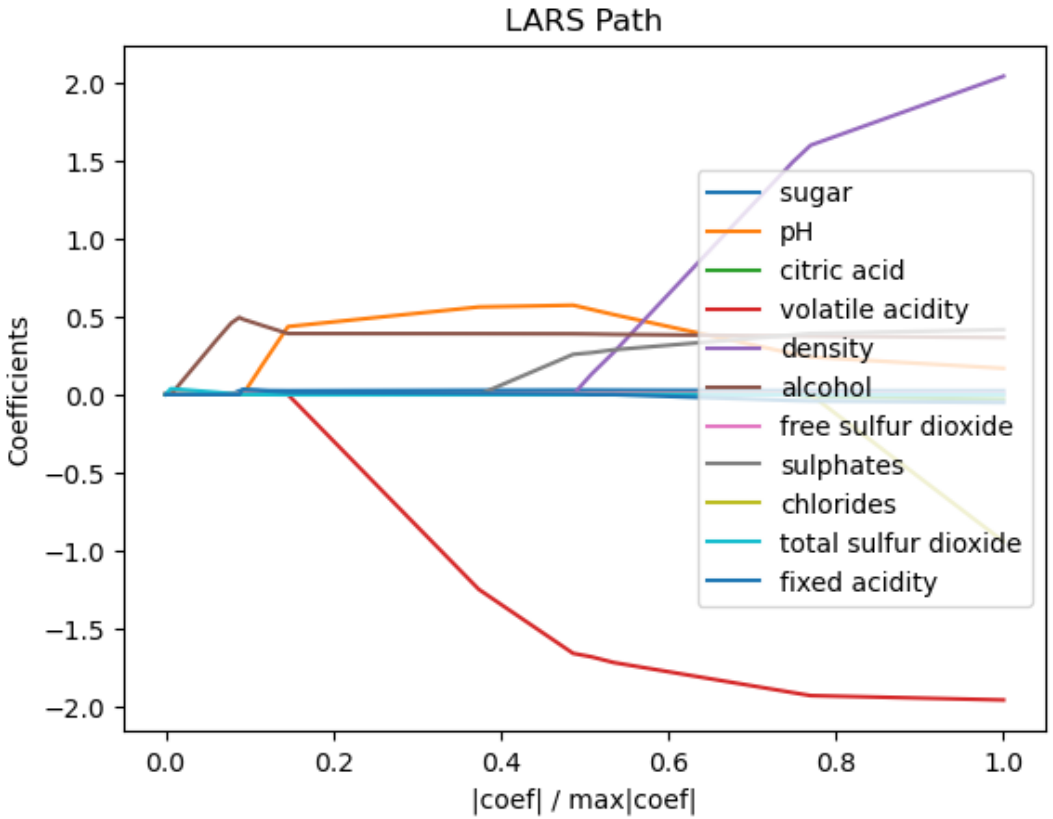
```
from sklearn.linear_model import LassoLarsIC

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 трассы коэффициентов регрессии наименьшего угла

```
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]
```



Регрессии на основе преобразования пространства признаков

- Рассмотренные методы (отбор, регуляризация и их комбинация как в LASSO/LARS) были связаны с построением модели линейной регрессии в *исходном пространстве* признаков
- Основные проблемы: **шум, зависимости, большая размерность** и как результат: переобучение, нестабильность,…
- Можно попробовать перейти из исходного пространства признаков X в *новое пространство* G размерности меньше исходного $m \ll p$, чтобы избавится от основных проблем

$$X_{l\times p} = \begin{pmatrix} x_{11}, x_{12}, \dots x_{1p} \\ \dots \\ x_{l1}, x_{l2}, \dots x_{lp} \end{pmatrix} \Rightarrow G_{l\times m} = \begin{pmatrix} g_1(x_1) & g_2(x_1) & \dots g_m(x_1) \\ & \dots & \\ g_1(x_l) & g_2(x_l) & \dots g_m(x_l) \end{pmatrix}$$

и максимально сохранить информацию об исходном пространстве:

$$\left\| X - g^{-1}(G) \right\| \to \min_{g}$$

Линейное преобразования пространства признаков

- Линейная функция $g_i: X \to G$, $g_i(x) = \sum_j u_{ij} x_j$
- Матрица линейного преобразования:

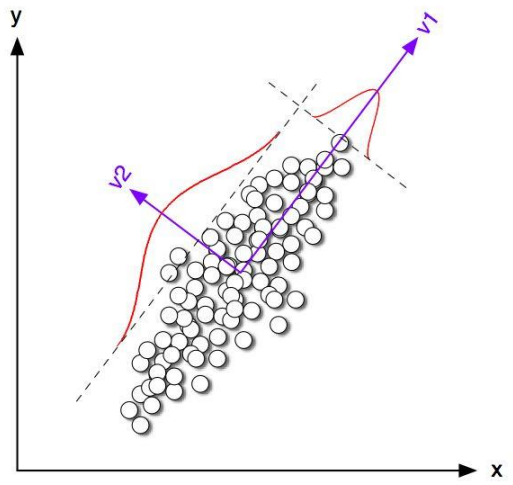
$$U_{p \times m} = \begin{pmatrix} u_{11}, u_{12}, \dots, u_{1m} \\ \dots \\ u_{p1}, u_{p2}, \dots, u_{pm} \end{pmatrix}$$

- Хотим, чтобы:
  - $X \approx GU^T \Rightarrow \left\| X - GU^T \right\| \to \min\limits_U$
  - G – ортогональна, $GG^T = \Delta$ – диагональная матрица
- Такое преобразование существует, оно находится с помощью метода главных компонент PCA (principal component analysis)

Общая идея PCA

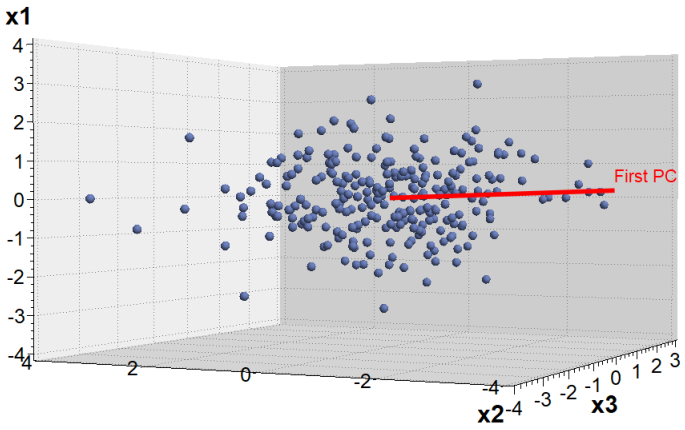
- Строится новый базис (линейное преобразование исходного пространства) такой, что:
  - Центр координат совпадает с мат. ожиданием наблюдений
  - Первый вектор направлен таким образом, что дисперсия вдоль него была максимальной
  - Каждый последующий вектор ортогонален предыдущим и направлен по направлению максимальной дисперсии
  - Последние компоненты – не очень важны
- Формально:

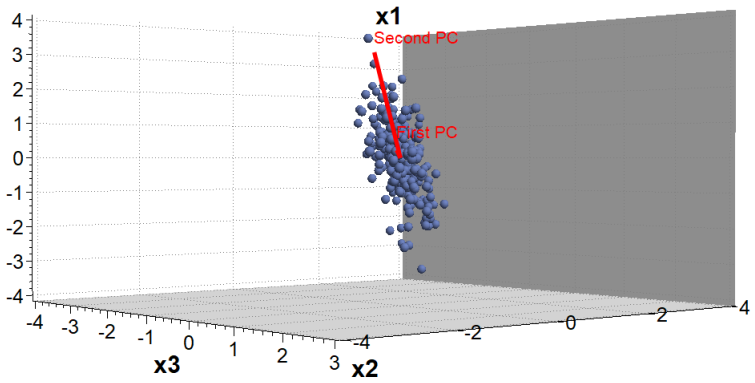
$$v = \underset{||v||=1}{arg\max} E\left\{\left[v^T\left(x - \sum_{i=1}^{k-1} v_i v_i^T x\right)\right]^2\right\}$$

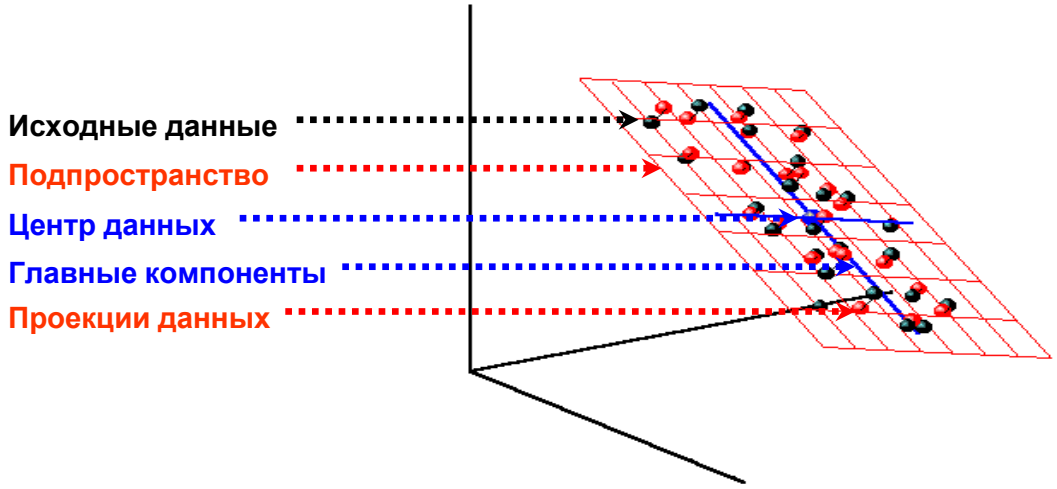


  - Решение - собственные значения ковариационной матрицы
  - Можем найти с помощью SVD разложения

Геометрическая интерпретация







Собственные значения и вектора ковариационной (корреляционной) матрицы

- Рассчитаем ковариационную (или корреляционную) матрицу:
  - $cov(x_i, x_j) = \mathbf{E}[(x_i - \mathbf{E}(x_i))(x_j - \mathbf{E}(x_j))]$
  - Ковариация = 0 – независимы
  - Ковариация > 0 – вместе растут и убывают
  - Ковариация < 0 – «противофаза»

$$\mathrm{C_p} = \begin{pmatrix} cov(x_1, x_1) & \cdots & cov(x_1, x_p) \\ \vdots & \ddots & \vdots \\ cov(x_p, x_1) & \cdots & cov(x_m, x_p) \end{pmatrix}$$

- Проблема с.зн.:

$$\mathrm{C}v = \lambda v$$

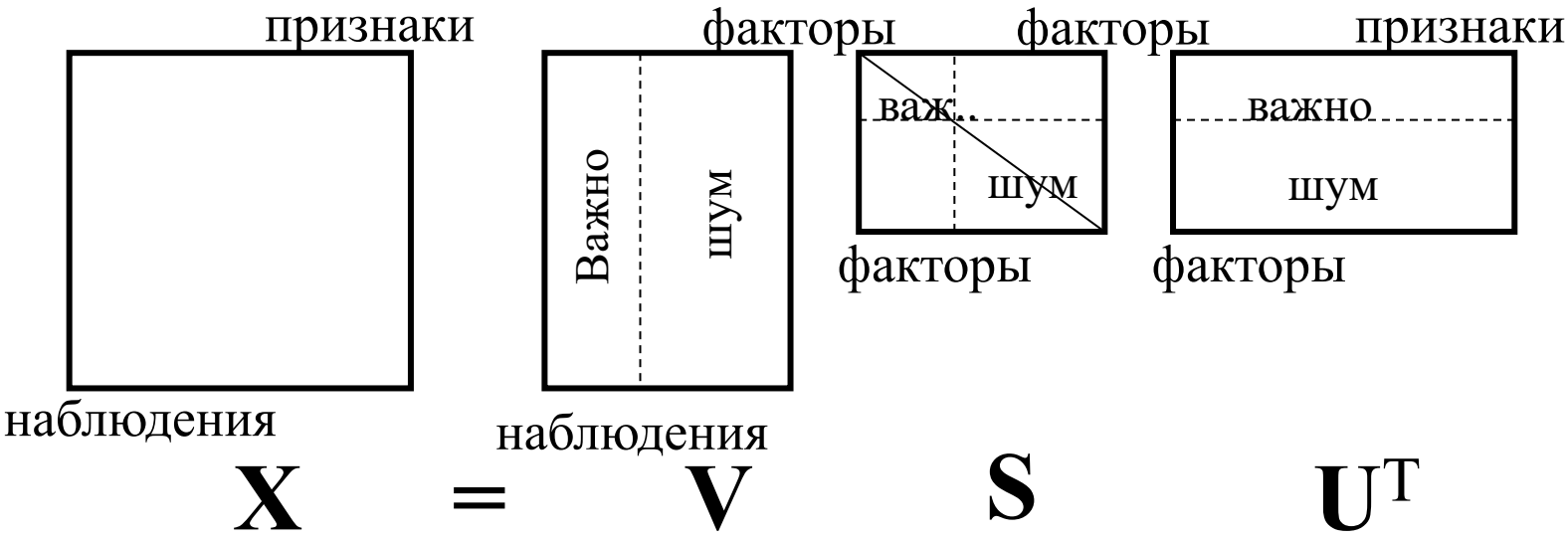
решение: поиск корней $\det(C - \lambda I) = 0$, матрица положительно определенная – есть вещественные корни

- Результат:
  - λ – дисперсии, с.в. – главные компоненты

$$g_1 = u_{11}x_1 + \cdots + u_{1p}x_p, \ldots, g_m = u_{m1}x_1 + \cdots + u_{mp}x_p, \forall i \sum_{j=1}^{p} u_{ij}^2 = 1$$

Сингулярное разложение в PCA

- Рассмотрим SVD, но отбросим наименее значимые гл. компоненты:

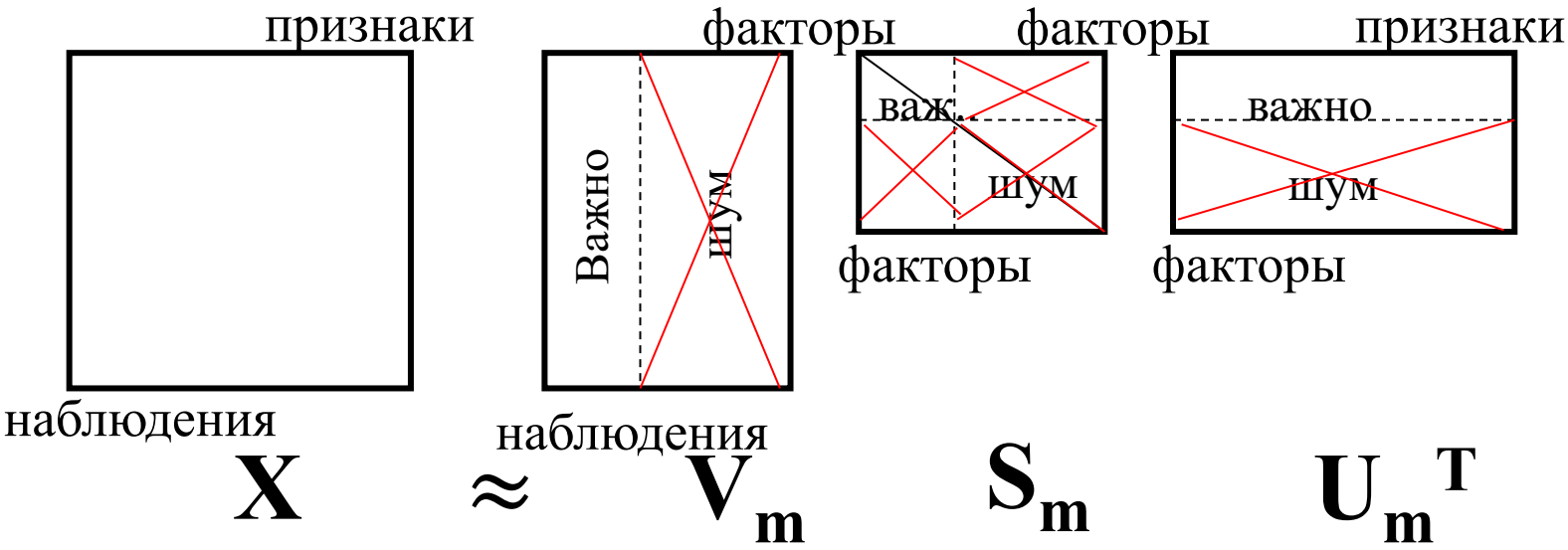


$$\mathbf{X} = \mathbf{V}\ \mathbf{S}\ \mathbf{U}^{\mathrm{T}}$$

- Число факторов m = число признаков p
- Орт. матрица $U^T U = I$ правых сингулярных векторов, с.в. $X^T X$
- Орт. матрица $VV^T = I$ левых сингулярных векторов, с.в. XX^T
- $S = diag(\sqrt{\lambda_1},\ldots,\sqrt{\lambda_p})$ – с.зн. XX^T и $X^T X$

Сингулярное разложение в PCA

- Рассмотрим SVD, но отбросим наименее значимые гл. компоненты:



$$\mathbf{X} \approx \mathbf{V_m}\ \mathbf{S_m}\ \mathbf{U_m^T}$$

- **Число факторов m << числа признаков p**
- Орт. матрица $U_m^T U_m = I$ для первых m с.в. $X^T X$
- Орт. матрица $V_m V_m^T = I$ для первых m с.в. XX^T
- $S_m = diag(\sqrt{\lambda_1}, \ldots \sqrt{\lambda_m}, \ldots, \sqrt{\lambda_p})$ – первых m с.зн. XX^T и $X^T X$

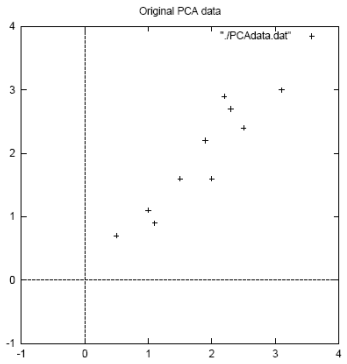
SVD разложение и обратная проекция

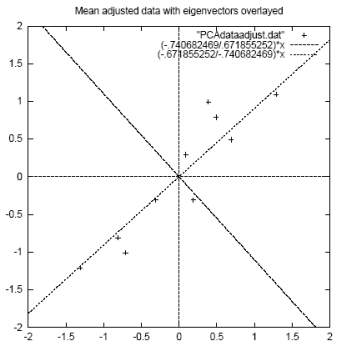
- SVD разложение матрицы: $X_{l\times p} = V_{l\times l} S_{l\times l} U^T_{p\times p}$
- SVD приближение (метод главных компонент):
  - отбрасываются с.в., соответствующие наименьшим с.з.
  - остается m-я часть главных с.в., которые характеризуют основные зависимости:

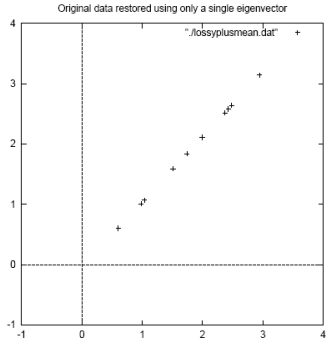
$$\min_{V,S,U} \left\| X_{l\times p} - V_{l\times m} S_{m\times m} {U_{m\times p}}^T \right\|$$

  - можем построить приближенную проекцию исходной матрицы:

$$\tilde{X} = UU^T X$$





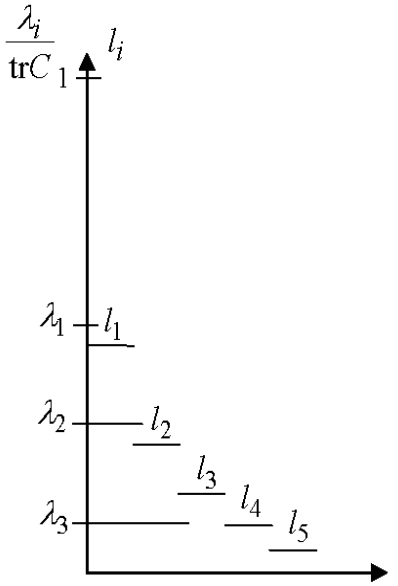


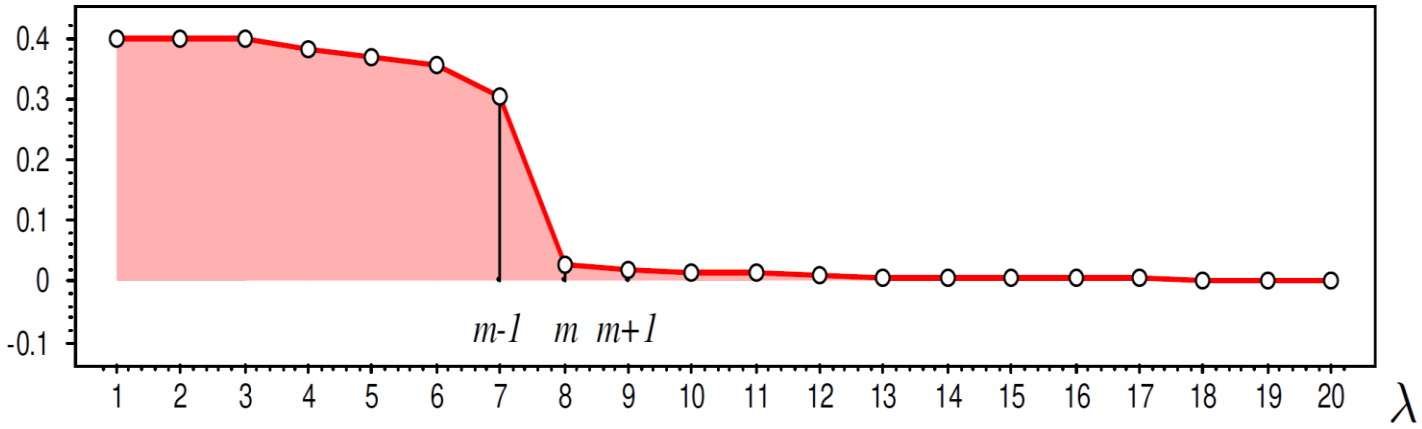
Отбор числа главных компонент

- Пропорция описанной вариации пространства признаков
- Метод Кайзера: оставляем с.в. у которых с.зн. больше среднего (или больше 1 для разложения корреляционной матрицы)
- Правило «сломанной стрости»

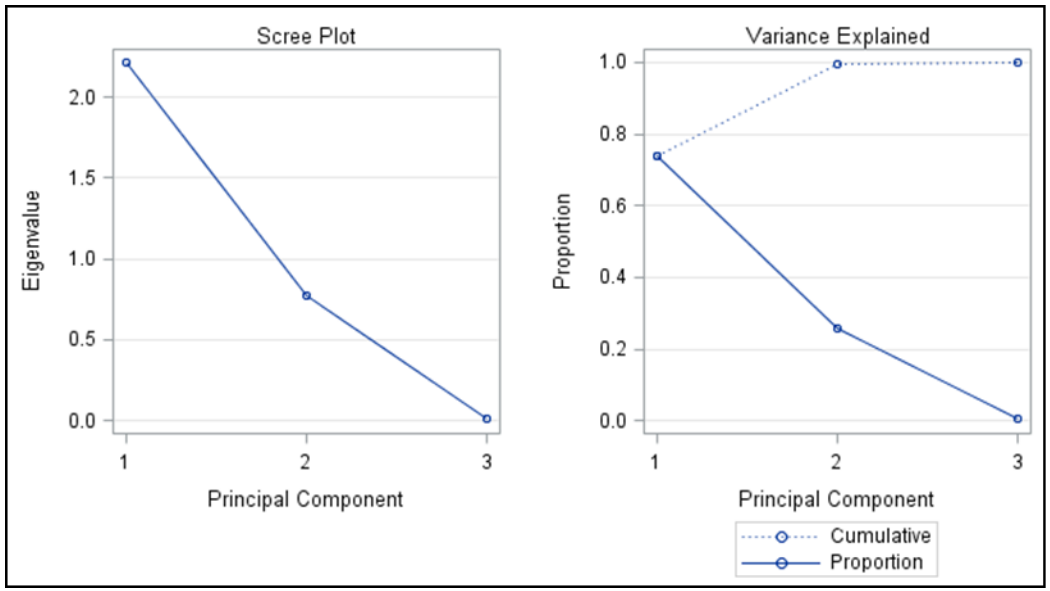
$l_i = E(L_i) = \frac{1}{p}\sum_{j=i}^{p}\frac{1}{j}$, λ_i оставляем если $\frac{\lambda_i}{tr(C)} > l_i$

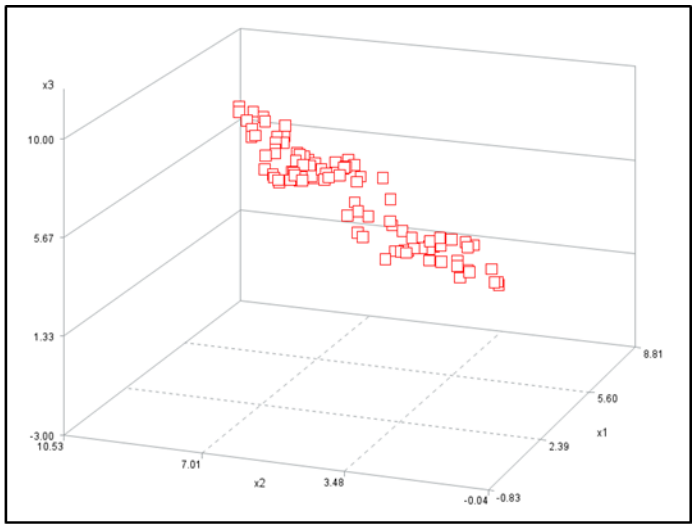
- По графикам – «скачки» и «плато» с.зн.





Пример 1

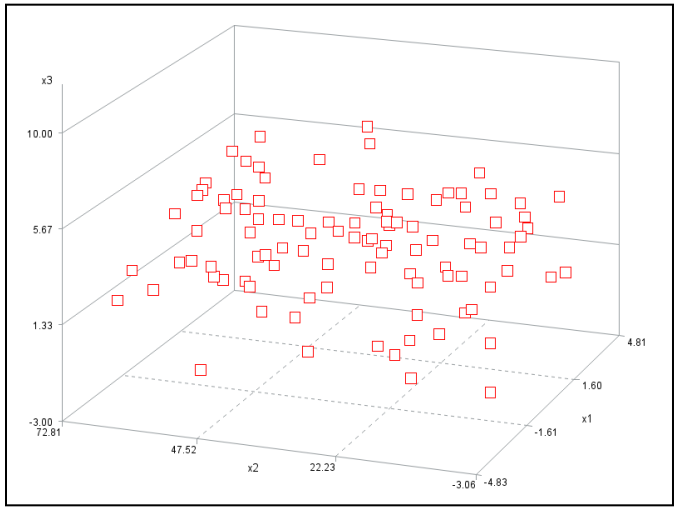




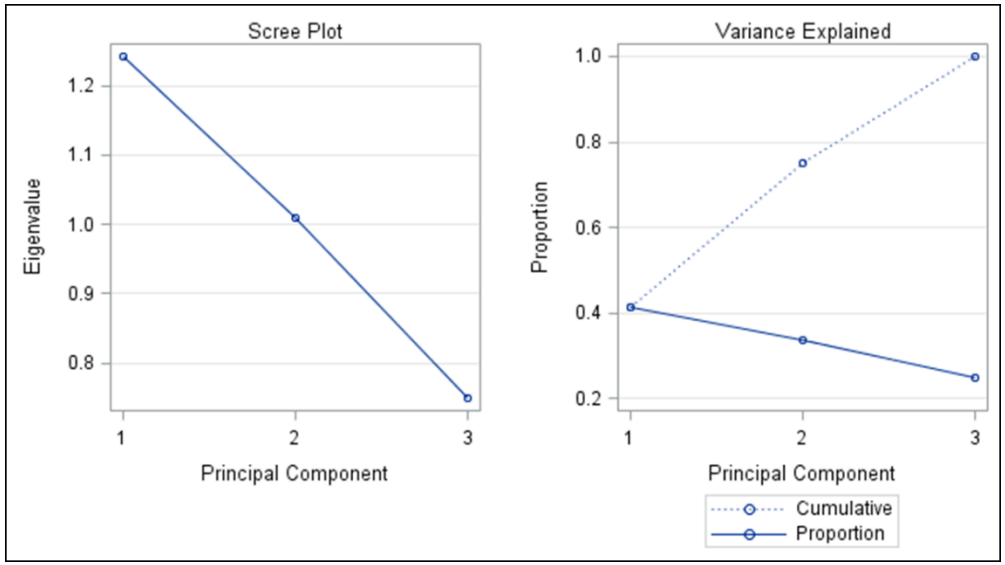
Eigenvalues of the Correlation Matrix				
	Eigenvalue	Difference	Proportion	Cumulative
1	2.21358154	1.44403082	0.7379	0.7379
2	0.76955072	0.75268297	0.2565	0.9944
3	0.01686775		0.0056	1.0000

Eigenvectors			
	Prin1	Prin2	Prin3
x1	0.650940	0.263685	0.711862
x2	0.645235	0.301851	-.701825
x3	-.399937	0.916164	0.026348

Пример 2



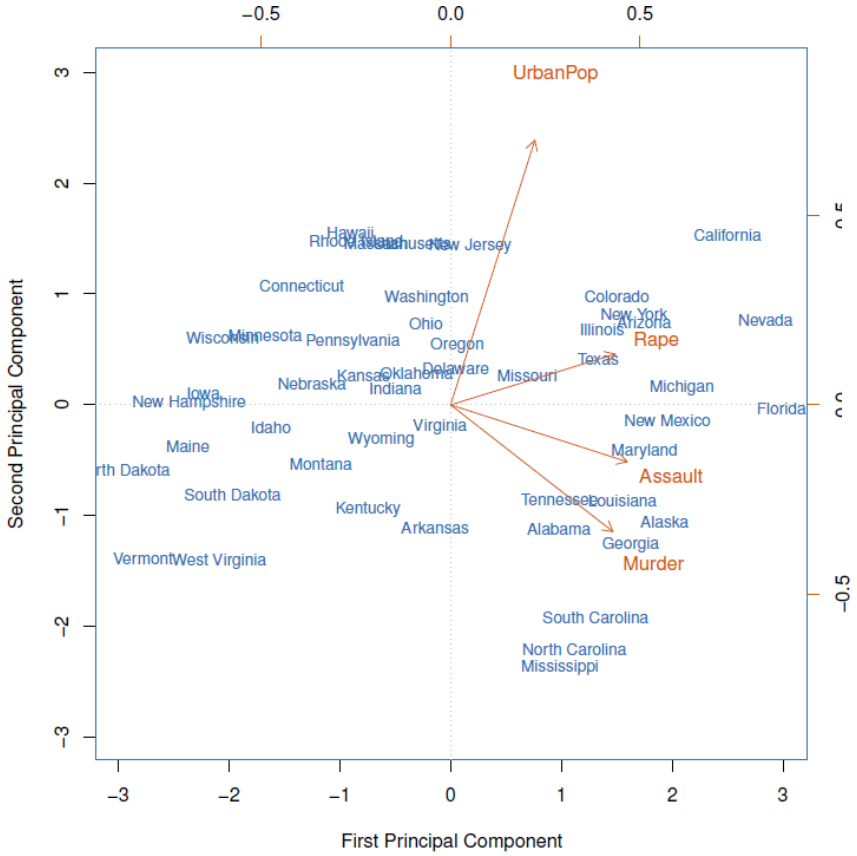
Eigenvalues of the Correlation Matrix				
	Eigenvalue	Difference	Proportion	Cumulative
1	1.24205697	0.23274599	0.4140	0.4140
2	1.00931098	0.26067894	0.3364	0.7505
3	0.74863205		0.2495	1.0000

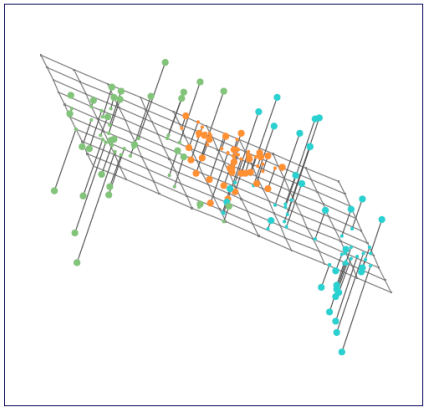


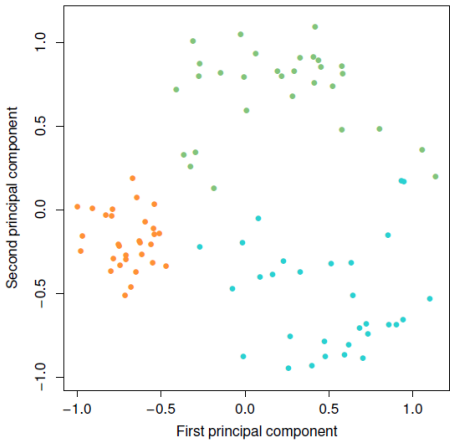
Eigenvectors			
	Prin1	Prin2	Prin3
x1	0.704619	-.156546	0.692102
x2	0.708942	0.113759	-.696032
x3	0.030228	0.981097	0.191139

Визуализация

- Одно из важнейших применений – проекция множества наблюдений на пары главных компонент







	PC1	PC2
Murder	0.5358995	-0.4181809
Assault	0.5831836	-0.1879856
UrbanPop	0.2781909	0.8728062
Rape	0.5434321	0.1673186

Пример использования (Python)

Пример:

- Дана матрица клиент-продукт:

```
import pandas as pd
```

```
assoc = pd.read_csv("assoc.csv")
```

```
assoc = assoc.groupby("CUSTOMER")["PRODUCT"].value_counts().unstack(fill_value=0)
assoc
```

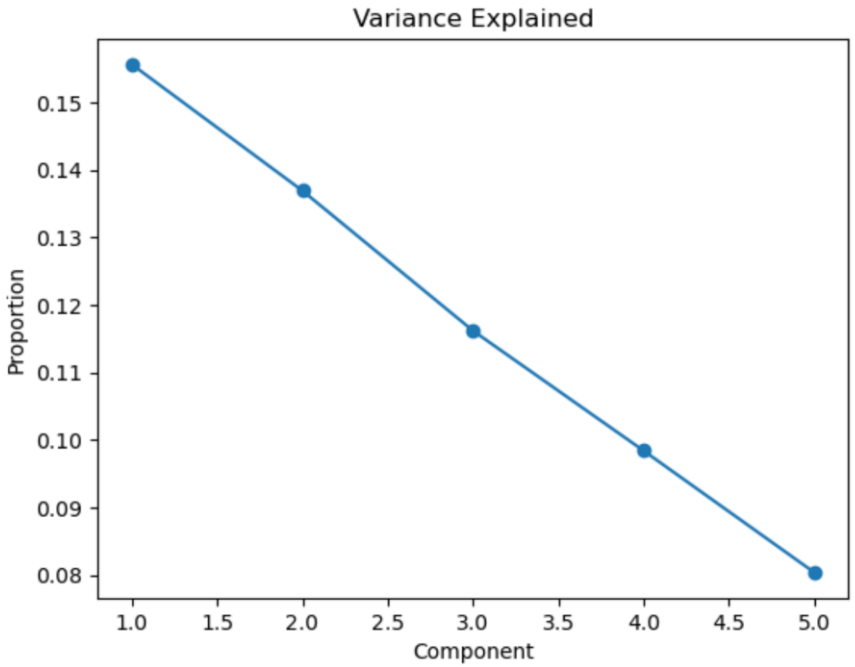
PRODUCT CUSTOMER	apples	artichok	avocado	baguette	bordeaux	bourbon	chicken	coke	corned_b	cracker	ham	heinel
0.0	0	0	0	0	0	1	0	0	1	0	1	
1.0	0	0	0	1	0	0	0	0	1	1	0	
2.0	0	1	1	0	0	0	0	0	0	1	1	
3.0	0	0	0	0	0	1	0	1	0	0	1	
4.0	1	0	1	0	0	0	0	0	1	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	

Пример использования (Python)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

```
N = 5
pca = PCA(n_components=N)
features = pca.fit_transform(assoc)
```

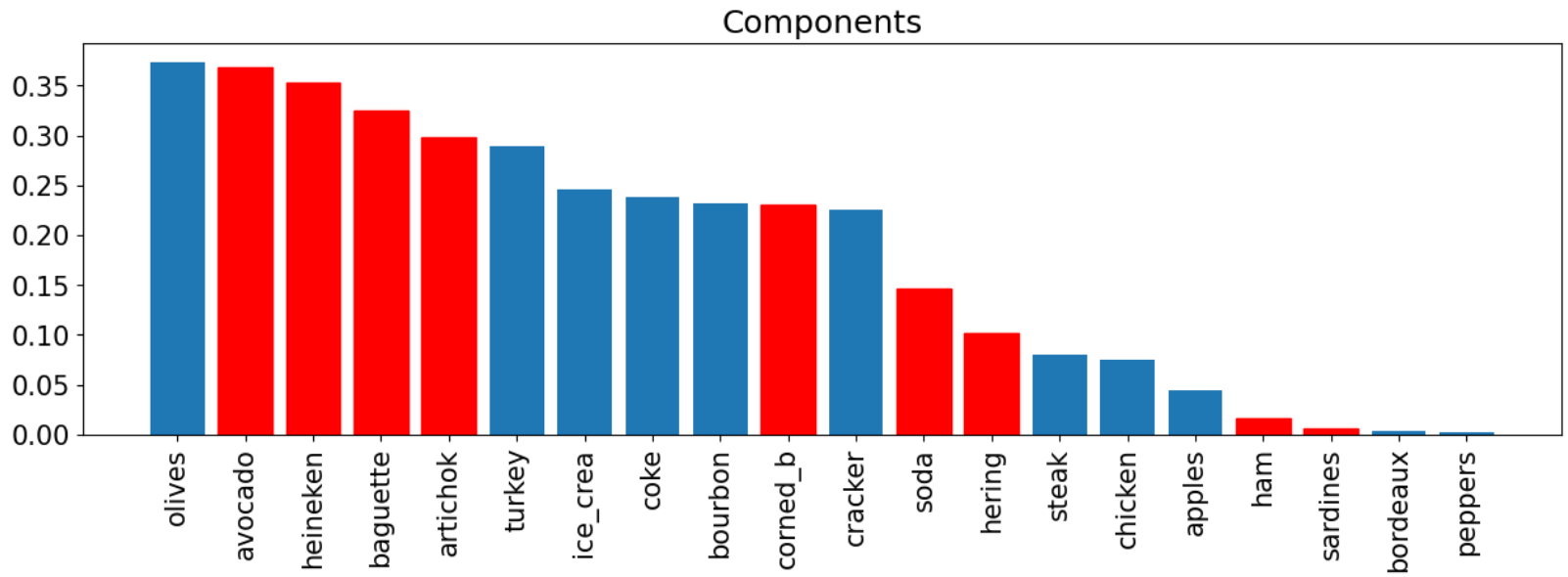
```
plt.plot(range(1, N+1), pca.explained_variance_ratio_)
plt.scatter(range(1, N+1), pca.explained_variance_ratio_)
plt.xlabel("Component")
plt.ylabel("Proportion")
plt.title("Variance Explained")
pass
```



Пример использования (Python)

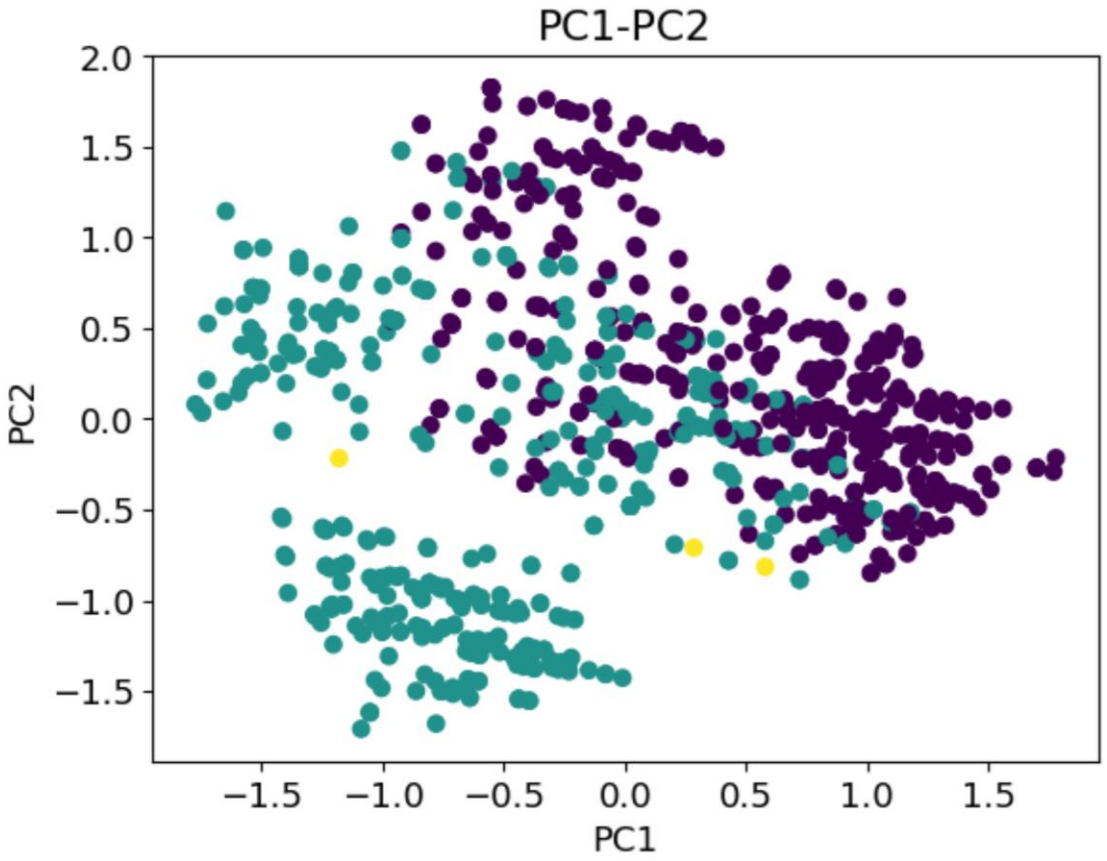
```
components = pca.components_ # [Число компонент, Число переменных]
sample = components[0] # Для примера возьмем первую
order = np.argsort(-abs(sample))
```

```
plt.xticks(rotation='vertical') # Поворачиваем текст на графике
# Нарисуем по модулю:
barplot = plt.bar(assoc.columns[order], abs(sample[order]))
# Отрицательные покрасим синими:
[x.set_color("red") for i, x in enumerate(barplot) if sample[i] > 0]
plt.title("Components")
pass
```



Пример использования (Python)

```
# Расскрасска - объем покупки оливок
plt.scatter(features[:, 0], features[:, 1], c=assoc["olives"])
plt.title("PC1-PC2")
plt.xlabel("PC1")
plt.ylabel("PC2")
pass
```



Регрессия главных компонент

- Применяем анализ главных компонентов (PCA), чтобы построить новое некоррелированное признаковое пространство меньшей размерности и строим в нем линейную регрессию МНК:

$$\min \sum_{i=1}^{l} \left(y_i - w_0 - \sum_{j=1}^{m} g_{ij} w_j \right)^2, \text{ где } g_{ij} = u_{j1} x_{i1} + \cdots + u_{jp} x_{ip}, j = \overline{1, m}$$

Похоже на регуляризацию?

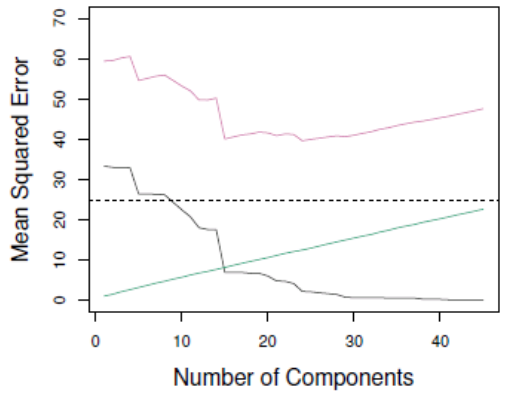
- В матричном виде:

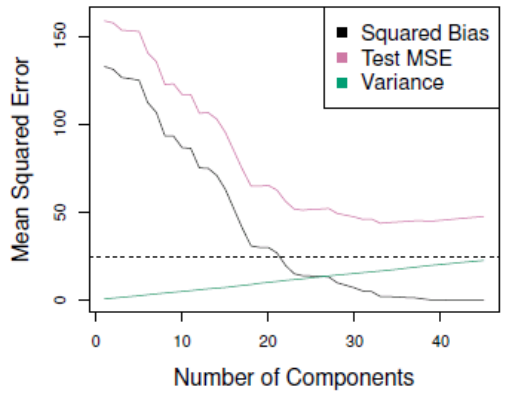
$$\left|\left| Gw - y \right|\right|^2 \to min,\ G = U^T X$$

- Можно подставить линейные равенства g_{ij} в полученную по МНК линейную регрессию $w = (G^T G)^{-1} G^T y$ и понять, что:
  - Полученная регрессия есть в том числе и линейная регрессия от исходных признаков, с коэффициентами $w_x = Uw$
  - Можно рассматривать регрессию главных компонент как одну из форм регуляризации, где ограничения на параметры модели заданы не через L_p, а равенствами через матрицу U

О выборе числа главных компонент

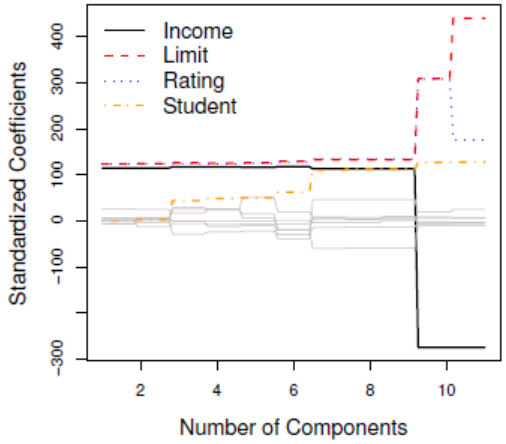
- MSE декомпозиция зависит от числа компонент:

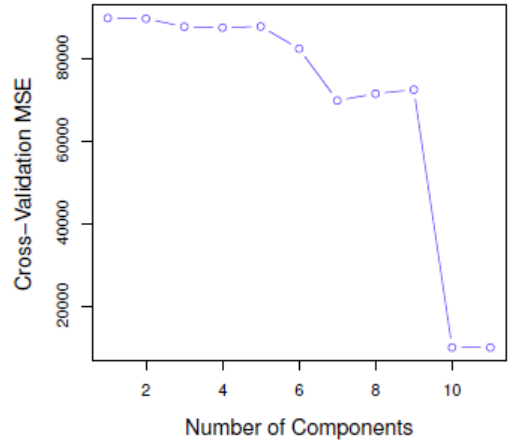




MSE на тестовой выборке
Дисперсия модели
Смещение модели

- Трассы коэффициентов





Применение регрессии главных компонент

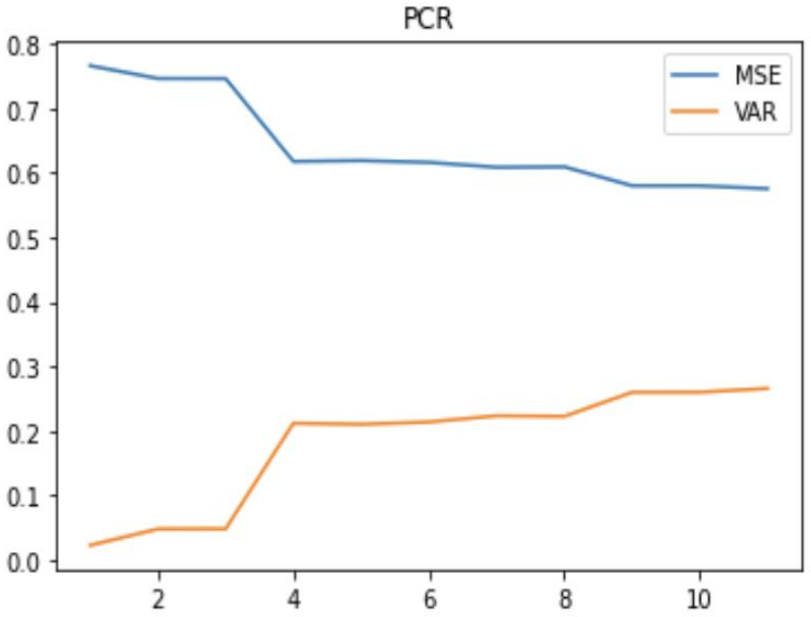
```
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_predict
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

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

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import explained_variance_score

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

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

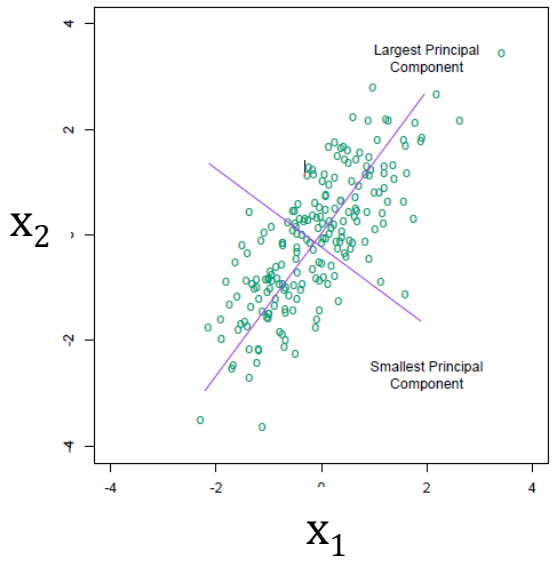


Метод частичных наименьших квадратов (PLS)

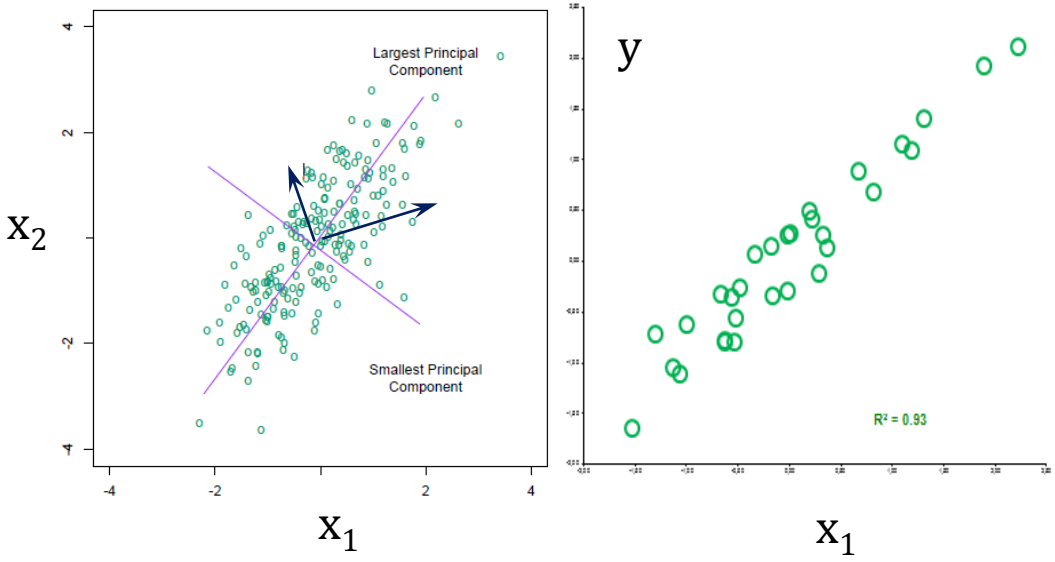
- PCR определяет линейные комбинации, или *направления*, которые наилучшим образом представляют предикторы
- Эти направления определяются *обучением без учителя*, так как отклик не используется при определении направлений главных компонент.
- Следовательно, PCR страдает от потенциально **серьезного недостатка**: нет никакой гарантии, что направления, которые наилучшим образом объясняют предикторы, также будут лучшими направлениями при использовании для прогнозирования отклика.
- PLS работает аналогично PCR, но определяет новый сокращенный набор ортогональных признаков с *учетом отклика*:

$$\max_{||z||=1} \boldsymbol{Corr^2}(\boldsymbol{y}, \boldsymbol{Xz}) Var(Xz)$$

Метод частичных наименьших квадратов (PLS)



Метод главных компонент



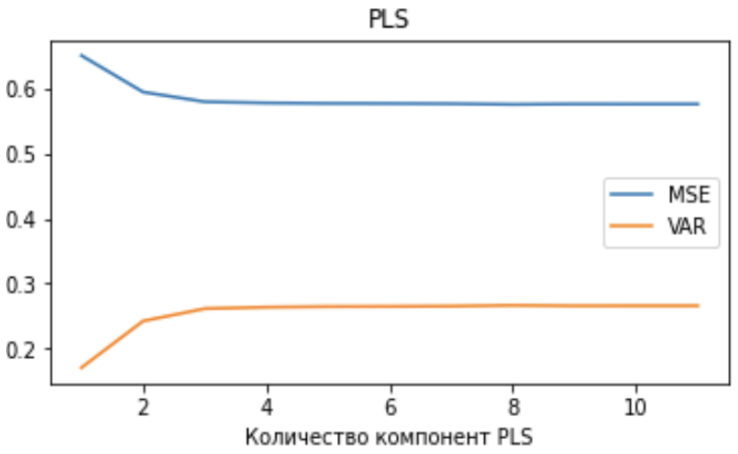
Метод наименьших частичных квадратов

Метод частичных наименьших квадратов (алгоритм SIMPLS)

- Стандартизируем X и Y
- Повторяем итерации $i = \overline{1,p}$
  - $c_{ij} = \left\langle x_j^{(i-1)}, y^{(i-1)} \right\rangle$ - расчет корреляций
  - $z_i = \sum_{j=1}^{p} c_{ij} x_j^{(i-1)}$ - расчет главной компоненты
  - $t_i = \frac{\langle z_i, y \rangle}{\langle z_i, z_i \rangle}$ - расчет проекции отклика на гл. компоненту
  - $y^{(i)} = y^{(i-1)} + t_i z_i$ - пересчет отклика
  - $x_j^{(i)} = x_j^{(i-1)} - z_i \frac{\left\langle z_i, x_j^{(i-1)} \right\rangle}{\langle z_i, z_i \rangle}$ - проекция данных

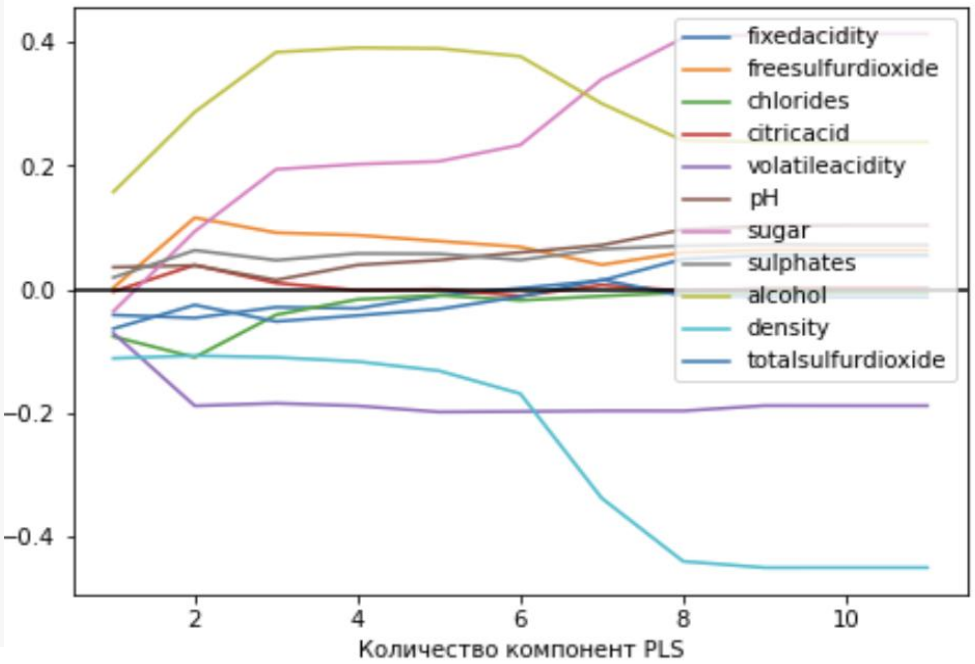
Применение PLS

```
from sklearn.cross_decomposition import PLSRegression

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)
    return mse, var

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

plt.figure(figsize=(6, 3))
plt.plot(list_components, np.array(mse_))
plt.plot(list_components, np.array(var_))
plt.xlabel('Количество компонент PLS')
plt.legend(["MSE","VAR"])
plt.title('PLS')
plt.show()
```



Применение PLS

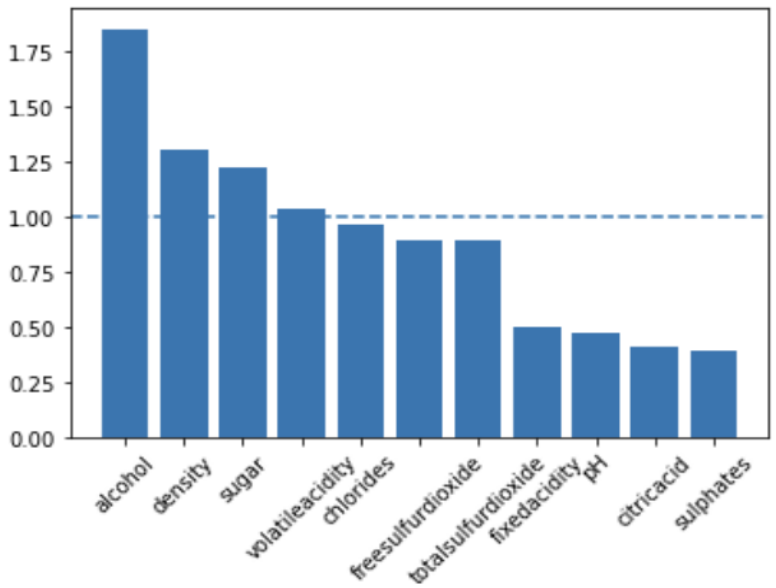
```
from sklearn.preprocessing import StandardScaler

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)
    coef = pls.coef_.reshape(-1, 1)
    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.xlabel('Количество компонент PLS')
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 u_{jk} в проекции на каждую из k главных компонент с учетом пропорции описанной ей вариации отклика SSY_k/SSY_{total} :

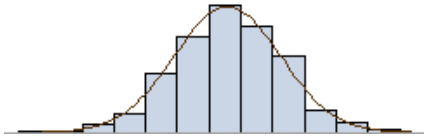
$$vip^2(x_j) = \sum_k u_{jk}^2 SSY_k/SSY_{total}$$



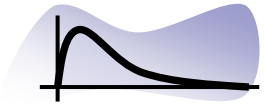

```
def vip(model):
  t = model.x_scores_
  w = model.x_weights_
  q = model.y_loadings_
  p, h = w.shape
  vips = np.zeros((p,))
  s = np.diag(t.T @ t @ q.T @ q).reshape(h, -1)
  total_s = np.sum(s)
  for i in range(p):
      weight = np.array([(w[i,j] / np.linalg.norm(w[:,j]))**2
                 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

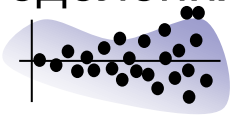
Важное предположение линейной регрессии

- Нормальное распределение ошибки с константной дисперсией:



- Часто возникающие «особенности»:
  - Несимметричные распределения отклика
  - Гетероскедастичность
  - Ограниченная область определения отклика





- Что делать?
  - Явно преобразовывать отклик: $\mathbf{E}(\mathbf{g}(\mathbf{y}) \mid \mathbf{x})$

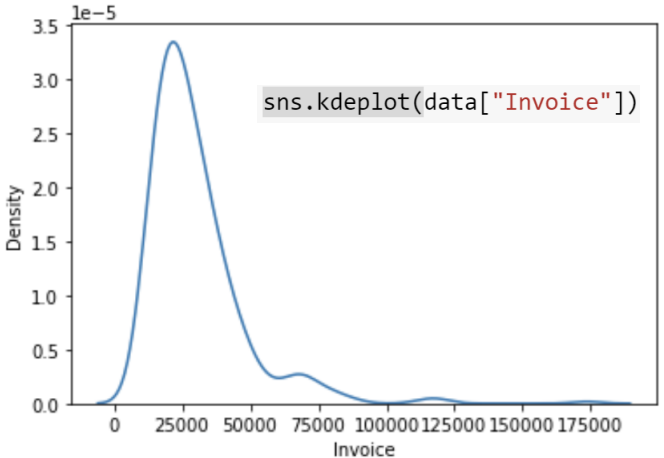
НО, в общем случае: $\mathbf{g}^{-1}(\mathbf{E}(\mathbf{g}(\mathbf{y}) \mid \mathbf{x})) \neq \mathbf{E}(\mathbf{y} \mid \mathbf{x})$

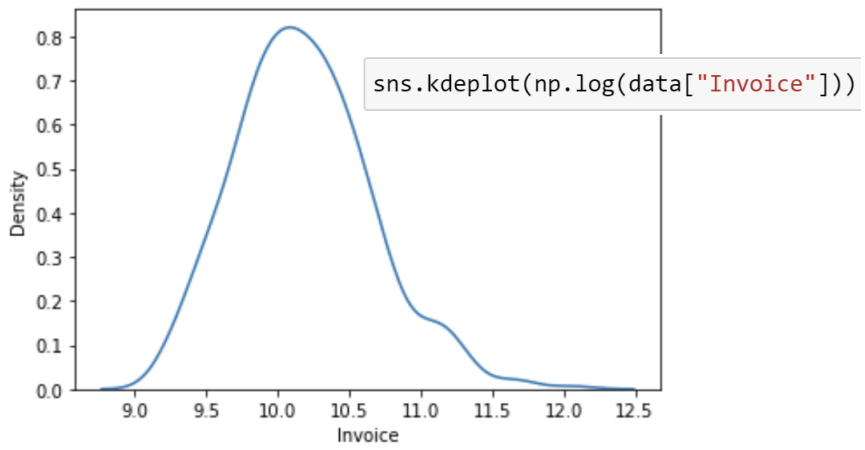
  - Использовать функцию связи: $\mathbf{g}(\mathbf{E}(\mathbf{y} \mid \mathbf{x}))$

Пример

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data=pd.read_csv("cars0.csv",delimiter=",")
data.head()
```

	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
0	Acura	MDX	SUV	Asia	All	36945.0	33337.0	3.5	6.0	265	17	23	4451	106	189
1	Acura	RSX Type S 2dr	Sedan	Asia	Front	23820.0	21761.0	2.0	4.0	200	24	31	2778	101	172
2	Acura	TSX 4dr	Sedan	Asia	Front	26990.0	24647.0	2.4	4.0	200	22	29	3230	105	183
3	Acura	TL 4dr	Sedan	Asia	Front	33195.0	30299.0	3.2	6.0	270	20	28	3575	108	186
4	Acura	3.5 RL 4dr	Sedan	Asia	Front	43755.0	39014.0	3.5	6.0	225	18	24	3880	115	197





Пример (МНК) – все плохо

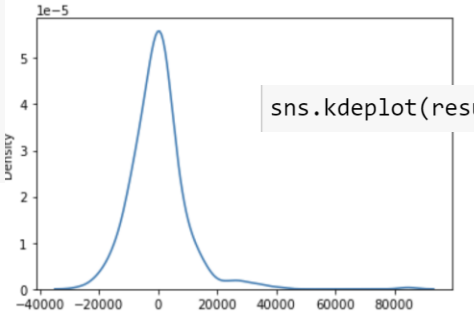
```
import statsmodels.api as sm

ols = sm.OLS(endog=data['Invoice'],
             exog=sm.add_constant(data[['Weight','Length','Horsepower']]), data=data)
results=ols.fit()
results.summary()
```

Dep. Variable:	Invoice	**R-squared:**	0.704
Model:	OLS	**Adj. R-squared:**	0.702
Method:	Least Squares	**F-statistic:**	336.3
Date:	Wed, 01 Nov 2023	**Prob (F-statistic):**	1.06e-111
Time:	01:43:01	**Log-Likelihood:**	-4531.2
No. Observations:	428	**AIC:**	9070.
Df Residuals:	424	**BIC:**	9087.
Df Model:	3		
Covariance Type:	nonrobust		

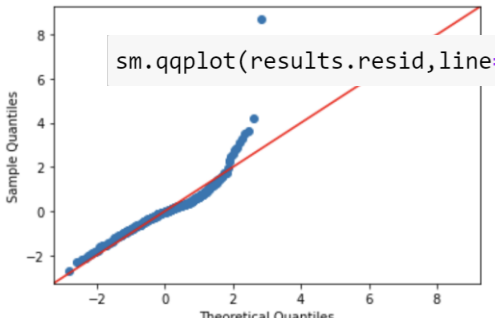
	coef	std err	t	P>|t|	[0.025	0.975]
const	2.25e+04	6682.648	3.367	0.001	9362.779	3.56e+04
Weight	0.0255	1.015	0.025	0.980	-1.970	2.021
Length	-213.1397	45.054	-4.731	0.000	-301.696	-124.583
Horsepower	218.3874	8.400	26.000	0.000	201.877	234.898

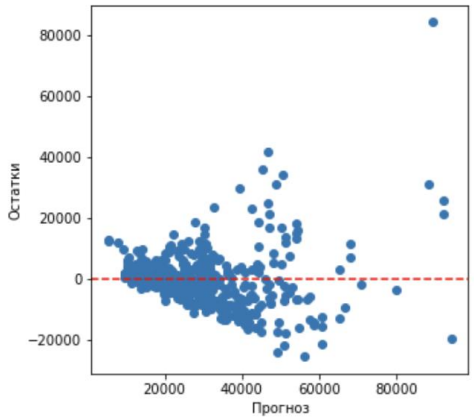
```
fig, ax = plt.subplots(figsize=(5, 5))
ax.scatter(results.predict(sm.add_constant(data[['Weight','Length','Horsepower']])),
           results.resid)
ax.set_ylabel('Остатки')
ax.set_xlabel('Прогноз')
plt.axhline(y = 0, color = 'r', linestyle = '--')
```

```
sns.kdeplot(results.resid)
```



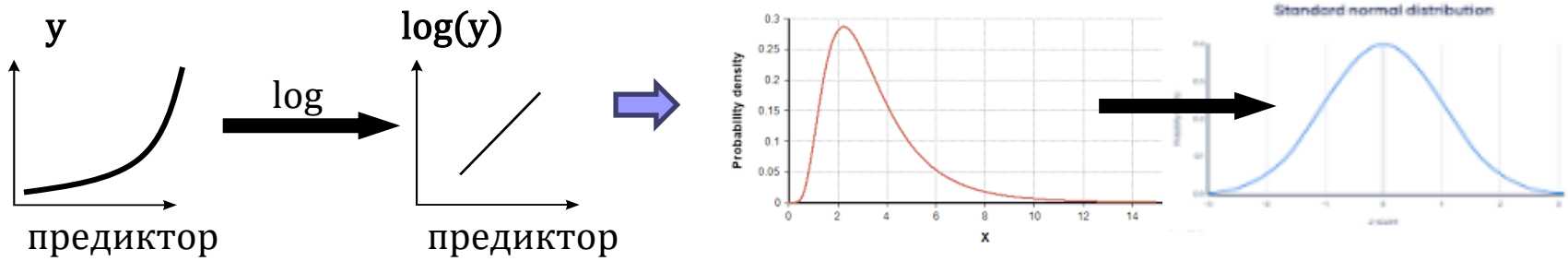

```
sm.qqplot(results.resid,line="45",fit=True)
```





Преобразование отклика и логнормальная регрессия

- Распределение отклика y логнормальное, тогда распределение с.в. $\log(y)$ – нормальное: $\log(y) \sim N(\mu, \sigma^2)$



- Связь моментов исходной с.в. y и $\log(y)$:

$$E(y) = \exp(\mu + \frac{\sigma^2}{2}),\ D(y) = \left(e^{\sigma^2} - 1\right)(E(y))^2$$

- Это значит, что можно построить МНК регрессию для прогнозирования $\log(y) = w^T x$ и получить исходный отклик:

$$\mu = E(\log(y)|x) = w^T x \Rightarrow E(y|x) = \exp\left(w^T x + \frac{\sigma^2}{2}\right)$$

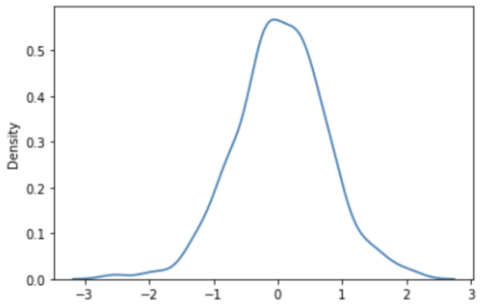
- Откуда брать σ^2? Можно взять оценку $\sigma^2 \approx MSE_{val}$, желательно на *валидационном* наборе

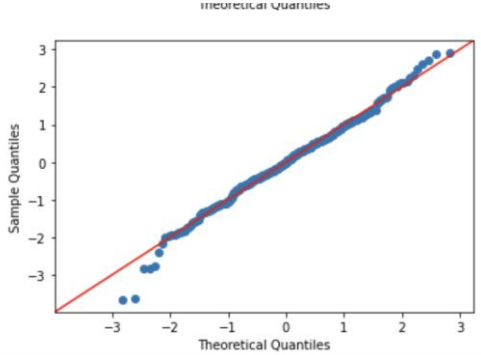
Пример (логнормальная регрессия) – лучше

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_test, y_train, y_test = train_test_split(
    sm.add_constant(data[['Weight','Length','Horsepower']]),
    np.log(data['Invoice']),test_size=0.3)
lnr = sm.OLS(endog=y_train, exog=X_train)
lnr_results=lnr.fit()
mse=mean_squared_error(y_test, lnr_results.predict(X_test))
lnr_results.summary()
```

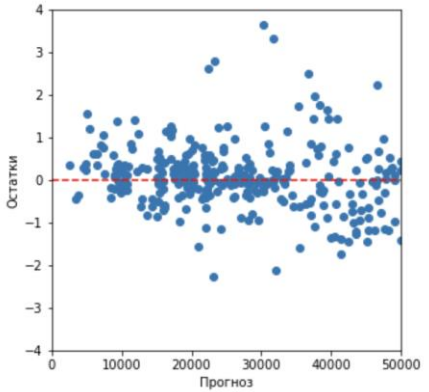
Dep. Variable:	Invoice	**R-squared:**	0.768
Model:	OLS	**Adj. R-squared:**	0.766
Method:	Least Squares	**F-statistic:**	325.9
Date:	Wed, 01 Nov 2023	**Prob (F-statistic):**	2.68e-93
Time:	01:54:31	**Log-Likelihood:**	9.5115
No. Observations:	299	**AIC:**	-11.02
Df Residuals:	295	**BIC:**	3.779
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	9.6851	0.209	46.330	0.000	9.274	10.097
Weight	0.0001	2.83e-05	4.264	0.000	6.5e-05	0.000
Length	-0.0060	0.001	-4.324	0.000	-0.009	-0.003
Horsepower	0.0055	0.000	22.407	0.000	0.005	0.006






```
fig, ax = plt.subplots(figsize=(5, 5))
ax.scatter(np.exp(mse/2+lnr_results.predict(
    sm.add_constant(data[['Weight','Length','Horsepower']]))),
    results.resid_pearson)
plt.xlim(0, 50000)
plt.ylim(-4, 4)
ax.set_ylabel('Остатки')
ax.set_xlabel('Прогноз')
plt.axhline(y = 0, color = 'r', linestyle = '--')
```



Обобщенная линейная модель

Функция связи → $g\big(\boldsymbol{E}(\boldsymbol{y}|\boldsymbol{x})\big) = w_0 + w_1 x_1 \ldots + w_k x_p = \langle x, w \rangle$

- Распределение отклика принадлежит экспоненциальному семейству $y_i \sim \mathrm{Exp}(\theta, \phi)$, где плотность определена как:

$$p(y|\theta,\phi) = exp\left(\frac{y\theta - c(\theta)}{\phi} + h(y,\phi)\right)$$

- Математическое ожидание с.в. y зависит только от θ через некоторую монотонную *функцию связи* $g(.)$ (link function) как: $\mu = E(y) = \mathrm{c}'(\theta) \Rightarrow \theta = g(\mu) = [c']^{-1}(\mu)$
- Дисперсия с.в. y есть функция от среднего: $D(y) = \phi c''(\theta)$
- Распределение отклика наблюдений может подсказать какую функцию связи и функцию потерь следует выбрать

Важные частные случаи

- Линейная регрессия: $p(y|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{(y-\mu)^2}{2\sigma^2}\right)$
- Логистическая регрессия: $p(y|\mu) = \mu^y(1-\mu)^{1-y}$
- Пуассоновская регрессия: $p(y|\lambda) = \frac{e^{-\lambda}\lambda^y}{y!}$
- Гамма регрессия: $p(y|\nu,\mu) = \frac{1}{\Gamma(\nu)y}\left(\frac{y\nu}{\mu}\right)^{\nu} e^{-\frac{y\nu}{\mu}}$

Регрессия	Отклик	Параметр θ (среднее)	Параметр ϕ	Дисперсия	Каноническая функция связи
Линейная	непрерывный неограниченный	μ	σ	σ^2	тождество $g(\mu)= \mu$
Логистическая	бинарный категориальный	μ	1	(1- μ) μ	логит $g(\mu)= \log(\mu/(1- \mu))$
Пуассоновская	«Счетчик» - дискретный положительный	λ	1	λ	логарифм $g(\mu)= \log(\mu)$
Гамма	непрерывный положительный	μ	ν	μ/ ν^2	обратная $g(\mu)= 1/\mu$

Примеры вывода функции связи

- Суть: приведение распределения к каноническому виду $\mathrm{Exp}(\theta, \phi)$
- Линейная регрессия (нормальное распределение):

$$p(y|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) = \exp\left(\frac{\boldsymbol{y\mu - \frac{1}{2}\mu^2}}{\sigma^2} - \frac{y^2}{2\sigma^2} - \frac{1}{2}\log(2\pi\sigma^2)\right)$$

$$\theta = g(\mu) = \mu,\ c(\theta) = \tfrac{1}{2}\mu^2 = \tfrac{1}{2}\theta^2$$

- Пуассоновская регрессия (распределение Пуассона):

$$p(y|\lambda) = \frac{e^{-\lambda}\lambda^y}{y!} = \exp\left(\frac{\boldsymbol{y\log(\lambda) - \lambda}}{1} - \log(y)!\right)$$

$$\theta = g(\lambda) = \log(\lambda),\ c(\theta) = \lambda = e^{\theta}$$

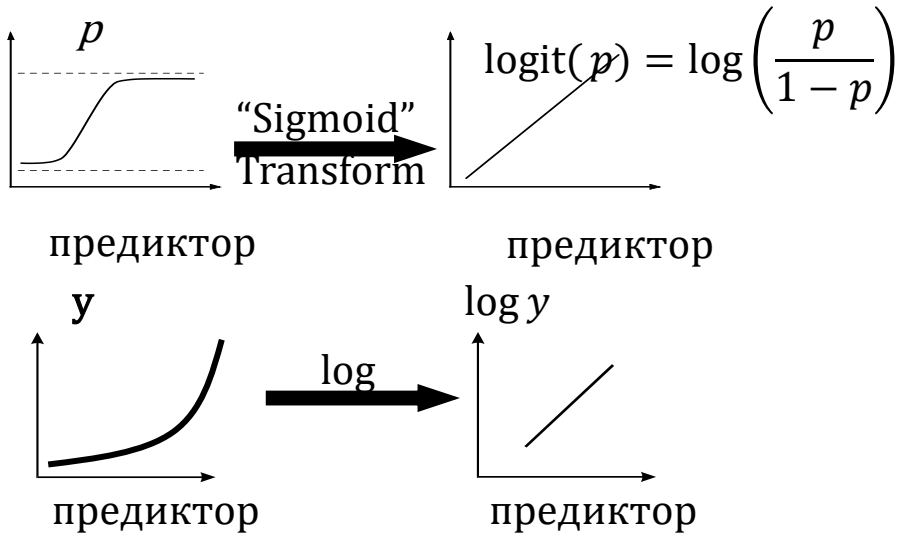
- Логистическая регрессия (распределение Бернулли):

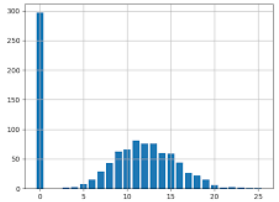
$$p(y|\mu) = \mu^y(1-\mu)^{1-y} = \exp\left(\boldsymbol{y\log\left(\frac{\mu}{1-\mu}\right) - \log(1-\mu)}\right)$$

$$\theta = g(\mu) = \log\left(\tfrac{\mu}{1-\mu}\right),\ c(\theta) = -\log(1-\mu) = \log(1+e^{\theta})$$

Не все так однозначно

- На практике часто используют неканонические функции связи
- Например, для логистической регрессии:
  - Каноническая logit
  - probit: $\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\mu} z^2 dz$
  - log-log: $\log(-\log(1-\mu))$
- Для гамма регрессии:
  - Каноническая обратная
  - log, тождественная и др.
- Для «счетчиков»:
  - Чрезмерная дисперсия - может не выполнятся условие $E(y) = D(y) = \lambda$ и тогда используют отрицательно биномиальное распределение, где дисперсия моделируется как функция от среднего и квадрата среднего
  - Может быть «смесь» счетчиков
  - “zero inflated” – смесь 0 и пуассоновского счетчика





Пример гамма регрессии

```
gamma_model = sm.GLM(data['Invoice'],
    sm.add_constant(data[['Weight','Length','Horsepower']]),
    family=sm.families.Gamma())
gamma_results = gamma_model.fit()
gamma_results.summary()
```

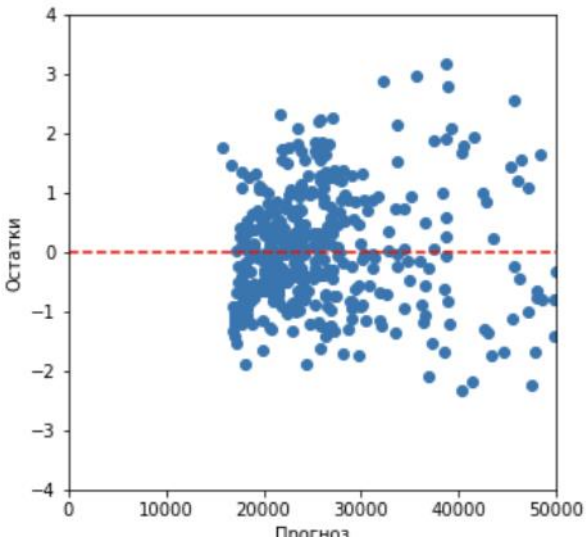
Dep. Variable:	Invoice	**No. Observations:**	428
Model:	GLM	**Df Residuals:**	424
Model Family:	Gamma	**Df Model:**	3
Link Function:	inverse_power	**Scale:**	0.11306
Method:	IRLS	**Log-Likelihood:**	-5686.6
Date:	Wed, 01 Nov 2023	**Deviance:**	310.53
Time:	02:03:07	**Pearson chi2:**	47.9
No. Iterations:	8	**Pseudo R-squ. (CS):**	-74.85
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	4.818e-05	6.76e-06	7.124	0.000	3.49e-05	6.14e-05
Weight	-6.088e-09	5.99e-10	-10.164	0.000	-7.26e-09	-4.91e-09
Length	2.391e-07	4.43e-08	5.402	0.000	1.52e-07	3.26e-07
Horsepower	-1.467e-07	2.88e-09	-50.999	0.000	-1.52e-07	-1.41e-07

Как считать?
Ответ позже

Статистика Уальда
(аналогично
Стьюденту для МНК)

```
fig, ax = plt.subplots(figsize=(5, 5))
ax.scatter(gamma_results.predict(data[['Weight','Length','Horsepower']]),
           results.resid_pearson)
plt.xlim(0, 50000)
plt.ylim(-4, 4)
ax.set_ylabel('Остатки')
ax.set_xlabel('Прогноз')
plt.axhline(y = 0, color = 'r', linestyle = '--')
```



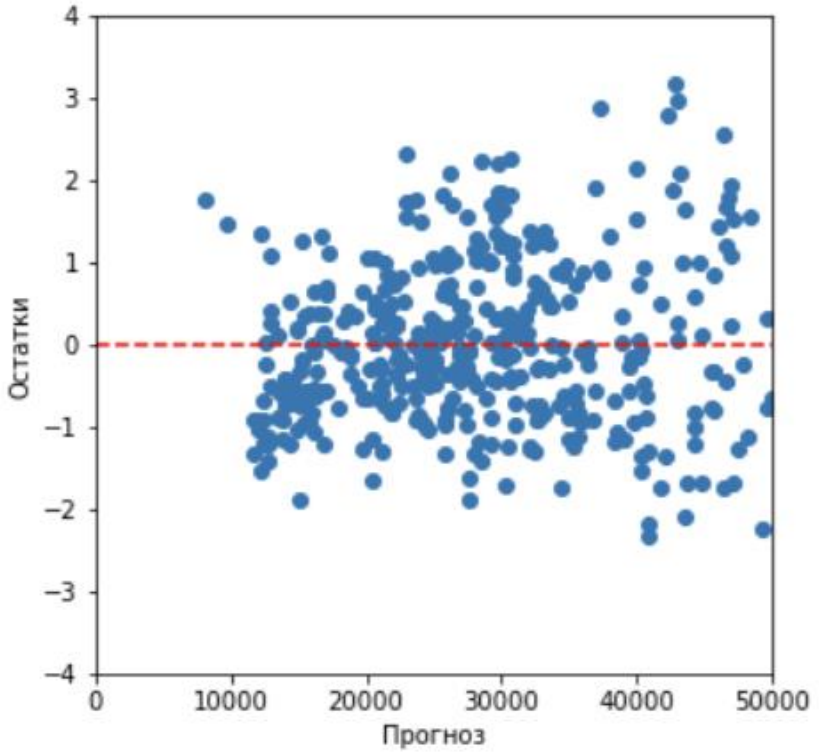
гетероскедастичность?

Пример гамма регрессии с неканонической тождественной функцией связи

```
gamma_model = sm.GLM(data['Invoice'],
    sm.add_constant(data[['Weight','Length','Horsepower']])
    family=sm.families.Gamma(sm.families.links.identity()))
gamma_results = gamma_model.fit()
gamma_results.summary()
```

Dep. Variable:	Invoice	**No. Observations:**	428
Model:	GLM	**Df Residuals:**	424
Model Family:	Gamma	**Df Model:**	3
Link Function:	identity	**Scale:**	0.066351
Method:	IRLS	**Log-Likelihood:**	-4359.6
Date:	Wed, 01 Nov 2023	**Deviance:**	24.571
Time:	02:06:57	**Pearson chi2:**	28.1
No. Iterations:	19	**Pseudo R-squ. (CS):**	0.9438
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.359e+04	4377.250	5.389	0.000	1.5e+04	3.22e+04
Weight	2.8085	0.849	3.307	0.001	1.144	4.473
Length	-209.3271	31.087	-6.734	0.000	-270.256	-148.398
Horsepower	161.8258	8.531	18.969	0.000	145.105	178.547

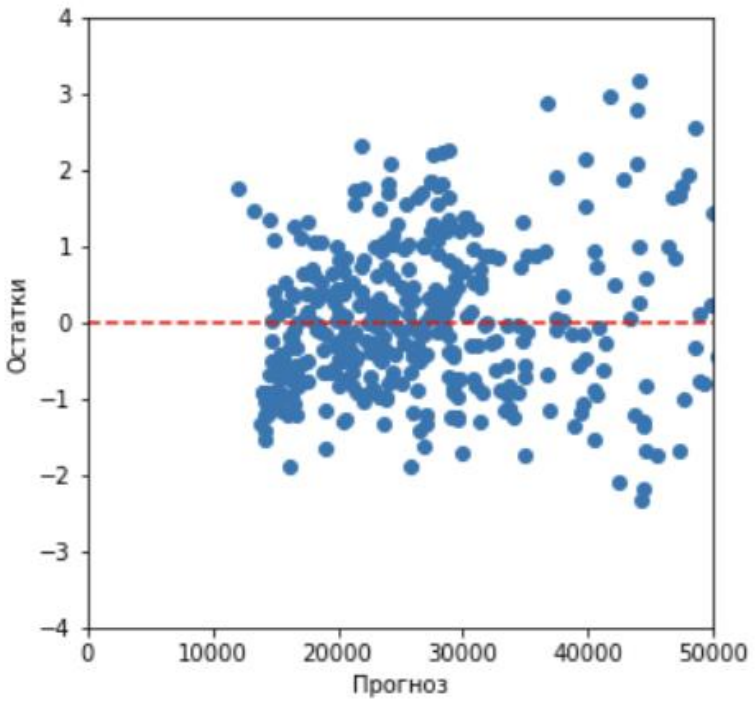


Пример гамма регрессии с неканонической функцией связи log

```
gamma_model = sm.GLM(data['Invoice'],
    sm.add_constant(data[['Weight','Length','Horsepower']]),
    family=sm.families.Gamma(sm.families.links.Log()))
gamma_results = gamma_model.fit()
gamma_results.summary()
```

Dep. Variable:	Invoice	**No. Observations:**	428
Model:	GLM	**Df Residuals:**	424
Model Family:	Gamma	**Df Model:**	3
Link Function:	Log	**Scale:**	0.059580
Method:	IRLS	**Log-Likelihood:**	-4346.8
Date:	Wed, 01 Nov 2023	**Deviance:**	23.319
Time:	02:10:09	**Pearson chi2:**	25.3
No. Iterations:	12	**Pseudo R-squ. (CS):**	0.9614
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	9.6571	0.169	57.017	0.000	9.325	9.989
Weight	0.0001	2.57e-05	4.863	0.000	7.47e-05	0.000
Length	-0.0058	0.001	-5.077	0.000	-0.008	-0.004
Horsepower	0.0055	0.000	25.850	0.000	0.005	0.006



Максимизация правдоподобия для GLM методом Ньютона-Рафсона

- Принцип максимума правдоподобия:

$$L(w) = -\log \prod_{i=1}^{l} p(y_i|\theta_i, \phi_i) = -\sum_{i=1}^{l} [y_i\theta_i - c(\theta_i)]/\phi_i \to \min_w \quad ,$$

где $\theta_i = w^T x_i$

- Метод Ньютона-Рафсона (t– номер итерации):

$$w^{t+1} = w^t - \eta_t \, (\nabla^2 L(w^t))^{-1} \nabla L(w^t)$$

- Градиент $\nabla L\left(w^t\right)$:

$$\frac{\partial L(w)}{\partial w_j} = \sum_{i=1}^{l} \frac{y_i - c'(w^T x_i)}{\phi_i} x_i$$

- Матрица Гессе $\nabla^2 L(w^t)$:

$$\frac{\partial^2 L(w)}{\partial w_j \partial w_k} = -\sum_{i=1}^{l} \frac{c''(w^T x_i)}{\phi_i} x_i x_k$$

Метод IRLS (Iteratively reweighted least squares)

- Обозначения:
  - Взвешенная (по наблюдениям) матрица признаков $\tilde{X} = W_t X$,
  - где X исходная матрица данных,
  - $W_t = diag\left(\sqrt{\frac{c''(\theta_i)}{\phi_i}}\right)$ – веса наблюдений на t-ой итерации
  - $\tilde{y_i} = \frac{y_i - c'(\theta_i)}{\sqrt{\phi_i c''(\theta_i)}}$ – модифицированные отклики
- Метод Ньютона-Рафсона принимает вид:

$$w^{t+1} = w^t - \eta_t \underbrace{\left(\mathrm{X^T W_t W_t X}\right)^{-1} \mathrm{X^T W_t}}_{\left(\tilde{X}^T \tilde{X}\right)^{-1} \tilde{X}^T} \left(\underbrace{\sqrt{\frac{\phi_i}{c''(\theta_i)}} \frac{y_i - c'(\theta_i)}{\phi_i}}_{\tilde{y}_i} \right)$$

- На каждом шаге - МНК линейной регрессии с взвешенными наблюдениями и модифицированными откликами:

$$\left\| \tilde{X} - \tilde{y} w \right\|^2 \to \min_w$$

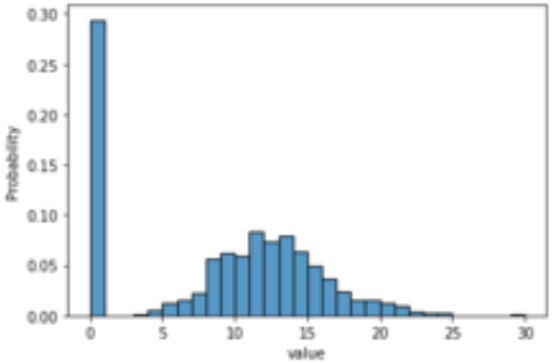
Особенности поиска решения

- При небольшой выборке IRLS – лучший вариант
- Но на больших выборках используют методы:
  - градиентные (в том числе стохастические)
  - квазиньютоновские (в том числе lbfgs)
- Есть варианты борьбы с переобучением:
  - L_1 и L_2 регуляризация
  - пошаговый отбор переменных (вместо тестов Фишера или Стьюдента – тест Уальда, информационные критерии и кросс-валидация работают как и для МНК)
- Для оценки важности переменных используются:
  - стандартные ошибки расчета коэффициентов (за рамками нашего курса)
  - статистика Уальда для оценки важности коэффициентов:

$$\frac{w_i}{SE(w_i)} \sim N(0,1)$$

Zero – inflated модели

- Наличие пика распределения:
  - обычно в 0, но может быть и нет
  - пиков может быть несколько, например в 1 и 0
  - Много 0 еще не значит Zero – inflated –



может быть просто маленькое мат. ожидание
с чрезмерной дисперсией, Iдолжно быть две моды – в 0 и не в 0

- Моделируются как порождающая модель смеси распределений:

$$p(x) = \sum_{j=1}^{k} w_j \varphi(x, \theta_j), \quad \sum_{j=1}^{k} w_j = 1, \quad w_j \geq 0$$

  - k – число компонент смеси (обычно 2)
  - $w_j = P(j)$ – априорная вероятность j-ой компоненты (может задаваться априори, но обычно подгоняется)
  - $\varphi(x, \theta_j) = p(x|j)$ – функция правдоподобия j-ой компоненты, для не 0 обычно Пуассоновское или отрицательное биномиальное, но может быть и биномиальное, и даже непрерывное

Проверка на Zero – inflated модели

- Процедура:
  - для линейных смешанных моделей тестовое распределение рассматривается как смесь по компонентам исходной смеси распределений χ^2
  - есть подходы на основе анализа остатков и более «эвристический» подход: если ваша модель систематически прогнозирует меньше 0, чем надо, то надо проверить ZI версию модели
- Если одна или несколько компонент в смеси константные, то:
  - задача проверки существенно упрощается, например, для случая установки одного компонента дисперсии в 0, тестовое распределение $\pi\chi_1^2 + (1-\pi)\chi_0^2$
  - оценка параметров смеси может сводится к оценке параметров условных распределений вида $\varphi(x, \theta_j) = p(x|j) = p(x|\theta_j, x > 0)$ при

В самом общем случае - EM-алгоритм

- Задача максимизации логарифма правдоподобия

$$L(w,\theta) = \ln \prod_{i=1}^{l} p(x_i) = \sum_{i=1}^{l} \ln \left[\sum_{j=1}^{k} w_j \varphi(x_i, \theta_j) \right] \to max_{w,\theta}$$

  - при ограничениях $\sum_{j=1}^{k} w_j = 1; w_j \geq 0$
  - вводим **скрытые переменные** $g_{ij} = P(j|x_i)$, их семантика – распределение ненаблюдаемых «меток» компонент (кластеров) для каждого наблюдения, позволяет максимизировать «взвешенное» правдоподобие **отдельно по каждой компоненте**
- Итерационный алгоритм Expectation-Maximization:
  - Начальное приближение параметров (w, θ);
  - **E-шаг** $(w, \theta) \to G = (g_{ij})$**:** оценка скрытых переменных
  - **M-шаг** $(w, \theta, G) \to (w, \theta)$: максимизация взвешенного правдоподобия отдельно по компонентам (w, θ)
  - Пока w, θ и G не стабилизируются.

EM-алгоритм

- Теорема (необходимые условия экстремума):
  - точка $(w_j, \theta_j)_{j=1}^{k}$ локального экстремума логарифмического правдоподобия $L(w, \theta)$ удовлетворяет системе уравнений относительно w_j, θ_j, g_{ij}:
  - E-шаг: $g_{ij} = \frac{w_j \varphi(x_i, \theta_j)}{\sum_{s=1}^{k} w_s \varphi(x_i, \theta_s)}, \ i = 1, \dots, l, \ j = 1, \dots, k;$
  - M-шаг: $\theta_j = \arg\max_{\theta} \sum_{i=1}^{l} \boldsymbol{g_{ij}} \ln \varphi(x_i, \theta), \ j = 1, \dots, k;$
  - $w_j = \frac{1}{l} \sum_{i=1}^{l} g_{ij}, \quad j = 1, \dots, k.$
- Вероятностная интерпретация:
  - **E-шаг** – формула Байеса: $g_{ij} = P(j|x_i) = \frac{P(j)p(x_i|j)}{p(x_i)} = \frac{w_j \varphi(x_i, \theta_j)}{p(x_i)} = \frac{w_j \varphi(x_i, \theta_j)}{\sum_{s=1}^{k} w_s \varphi(x_i, \theta_s)},$ при условии нормировки $\sum_{j=1}^{k} g_{ij} = 1$
  - **M-шаг** – это максимизация **взвешенного** правдоподобия, с весами объектов g_{ij} для j-ой компоненты смеси:

$$\theta_j = \arg\max_{\theta} \sum_{i=1}^{l} g_{ij} \ln \varphi(x_i, \theta), \ w_j = \frac{1}{l} \sum_{i=1}^{l} g_{ij}$$

Tweedie GLM регрессии

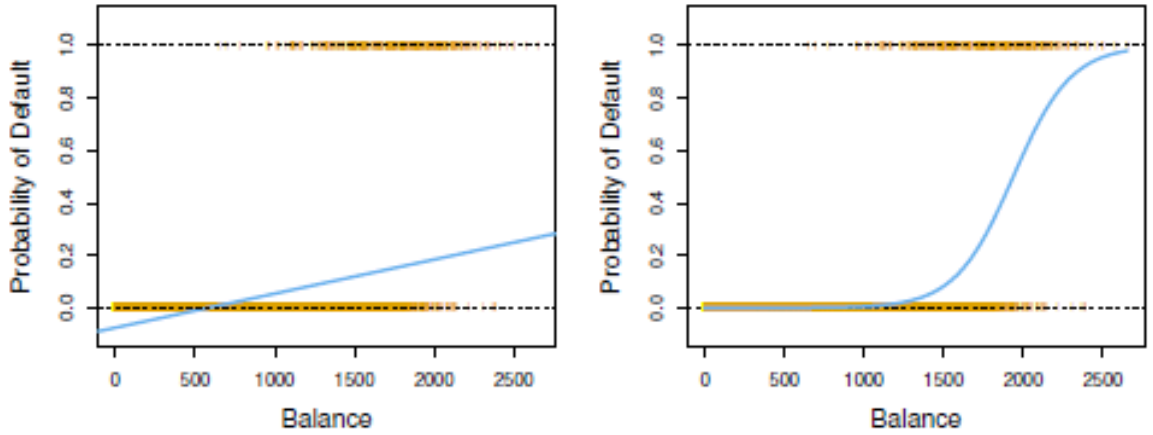
- Твиди распределение (основные факты):
  - Экспоненциальное семейство
  - Три параметра p - степень, μ – параметр положения, σ^2 - разброса
  - $Y \sim Tw_p(\mu, \sigma^2) \Rightarrow \mu = E(Y), D(Y) = \sigma^2 \mu^p$
  - Очевидно, что если $p = 0$ – нормальное распределение, $p = 1$ – Пуассона, $p = 2$ – Гамма, $p = 3$ – обратное гауссово распределение
  - Для $p \neq 0, 1, 2$ статистика отклонения (и функция потерь)

$$\mathrm{D} = 2\left(\frac{\max(y_i, 0)^{2-p}}{(1-p)(2-p)} - \frac{y_i a(x_i)^{1-p}}{1-p} - \frac{a(x_i)^{2-p}}{2-p}\right)$$

  - Часто применяется для моделирования «пуассоновских» сумм (число слагаемых из распределения пуассона) гамма, нормально или «обратно гауссово» распределенных случайных величин
  - А также для пуассоновско-гамма смесей, в том числе для zero-inflated gamma распределений

Логистическая регрессия

- Почему нельзя моделировать вероятность как непрерывный отклик с помощью линейной регрессии?



- Как представить категориальный отклик в виде числовой переменной?
- Если отклик закодирован (1=Yes, 0=No), а прогноз 1.1 или -0.4, что это означает?
- Если переменная имеет только два значения (или несколько), имеет ли смысл требовать постоянство дисперсии или нормальность ошибок?
- Вероятность ограничена, а линейная функция нет. Принимая во внимание ограниченность вероятности, можно ли предполагать линейную связь между предиктором и откликом?

Логистическая регрессия

Уравнение регрессии:

$$\text{logit}(p_i) = \mu = w_0 + w_1 x_{1i} + \cdots + w_p x_{pi}$$

Вероятность

параметр

предиктор

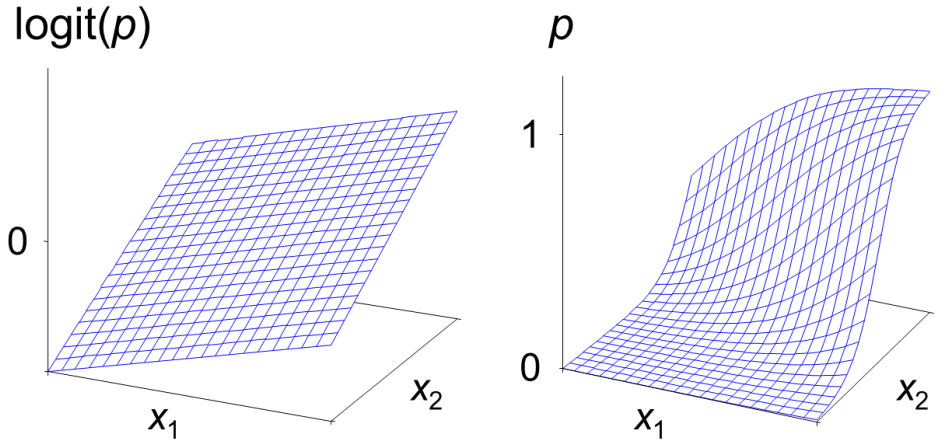
$$p_i = p(y = 1|x) = 1 - p(y = -1|x)$$

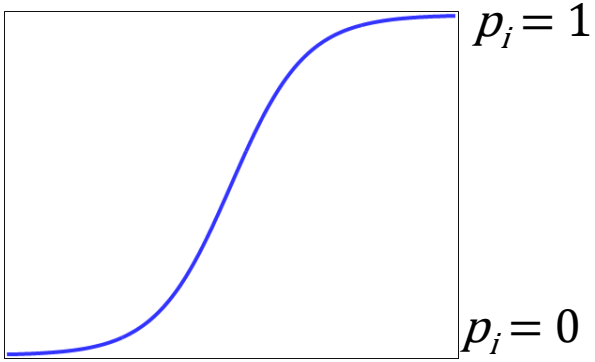
Функция связи (логит) и обратная ей (логистическая):

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1 - p_i}\right) = \mu \Rightarrow$$

$$\Rightarrow \quad p_i = \sigma(\mu) = \frac{1}{1+e^{-\mu}} = \frac{1}{1+e^{-x^T w}}$$

Основное предположение линейной логистической регрессии (линейная зависимость логита вероятности от предикторов):





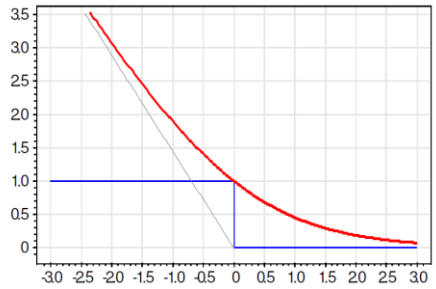
меньше $\leftarrow \mu \rightarrow$ больше

Ограничивает значение отклика

Функция потерь логистической регрессии

- **Функция потерь** (логарифмическая) является аппроксимацией негладкой функции потерь sign(.):

$$L(y, x, w) = \log\left[1 + \exp\left(-yw^T x\right)\right] \geq$$
$$\geq \text{ sign}(yw^T x)$$



- Градиент $\nabla Q(w)$ и матрица Гессе $\nabla^2 Q(w)$ для метода Ньютона-Рафсона:

$$w^{t+1} = w^t - \eta_t \left(\nabla^2 Q(w^t)\right)^{-1} \nabla Q(w^t)$$

$$\frac{\partial Q(w)}{\partial w_j} = \sum_{i=1}^{l} (1 - \sigma_i) y_i x_i, \quad \frac{\partial^2 Q(w)}{\partial w_j \partial w_k} = -\sum_{i=1}^{l} (1 - \sigma_i) \sigma_i y_i x_i x_k$$

где $\sigma_i = \sigma\left(y_i w^T x_i\right)$, $\sigma(z) = \frac{1}{1+e^{-z}}$ - сигмоидальная функция

IRLS для логистической регрессии

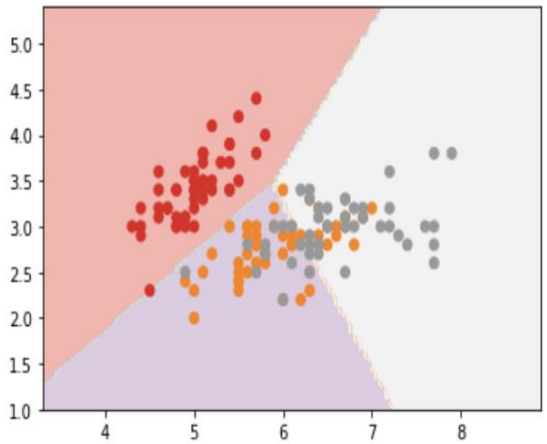
- На каждом шаге:
  - МНК линейной регрессии с взвешенными наблюдениями и модифицированными остатками, старающийся улучшить эмпирический риск на самых «сложных» примерах:

$$Q(w) = \sum_{i=1}^{l}(1-\sigma_i)\sigma_i\left(w^T x_i - \frac{y_i}{\sigma_i}\right)^2 \to \min_w \quad \leftrightarrow \quad \left\|\tilde{X} - \tilde{y}w\right\|^2 \to \min_w$$

- где:
  - Взвешенная (по наблюдениям) матрица признаков $\tilde{X} = W_t X$
  - X исходная матрица данных,
  - $W_t = diag\big((1-\sigma_i)\sigma_i\big)$ – веса наблюдений на t-ой итерации,
  - поскольку $\sigma_i = P(y_i|x_i)$- вероятность правильной классификации x_i, то чем ближе x_i к границе, тем больше вес $(1-\sigma_i)\sigma_i$ и «сложнее» пример
  - $\tilde{y_i} = \frac{y_i}{\sigma_i}$ – модифицированные отклики, чем выше вероятность ошибки тем больше $\frac{1}{\sigma_i}$

Многоклассовая логистическая регрессия и функция softmax

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.inspection import DecisionBoundaryDisplay

iris = datasets.load_iris()
X = iris.data[:, :2]
Y = iris.target

logreg = LogisticRegression()
logreg.fit(X, Y)

DecisionBoundaryDisplay.from_estimator(
    logreg, X, cmap="Pastel1")
plt.scatter(X[:, 0], X[:, 1], c=Y, cmap="Set1")
plt.show()
```



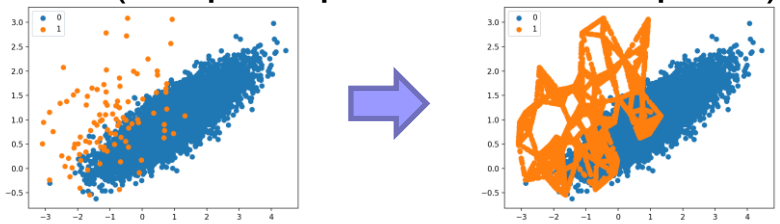
- Логистическая регрессия с двумя классами обобщается на случай K классов (многомерная логистическая функция):

$$p(y = k|x) = \frac{e^{w_k^T x}}{\sum_{j=1}^{K} e^{w_j^T x}}$$

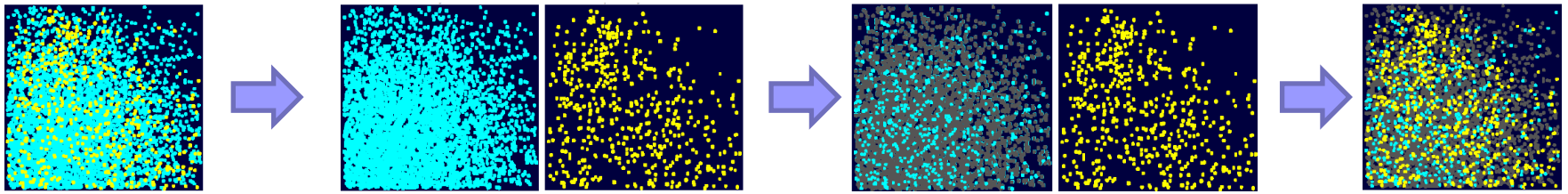
- Для *каждой* пары классов существует своя граница - линейная разделяющая функция, где вероятности классов совпадают
- Многоклассовая логистическая регрессия также называется *мультиномиальной регрессией, а* многомерная логистическая функция -softmax, *которая* «нормализует» K-мерный вектор так, чтобы сумма координат = 1

«Балансировка» выборки

- Варианты борьбы с дисбалансом:
  - Разные **веса у наблюдений** в функции потерь (обратно пропорционально общему числу наблюдений класса)
  - **Сдвиг границы** принятия решения в дискриминантной функции в сторону редкого класса пропорционально отношению размеров
  - «Балансировка» **oversampling** – с помощью некой стратегии генерируем случайные наблюдения для выборки, увеличиваем маленький класс (например, SMOTE алгоритм):



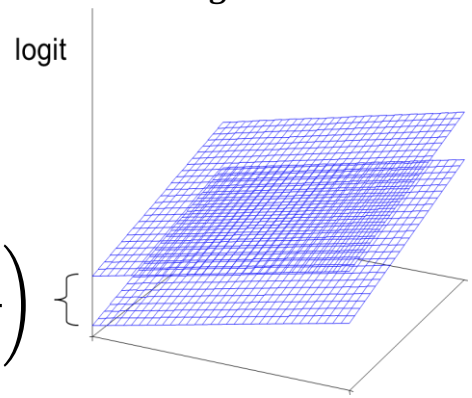
  - «Балансировка» **undersampling** – с помощью случайной выборки уменьшаем большой класс



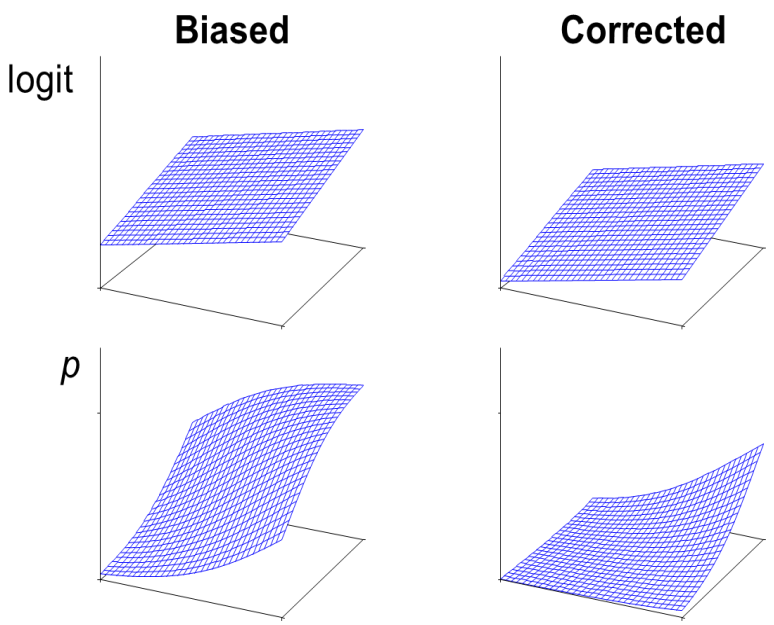
Корректировка после undersamplig

- Два способа корректировки:
  - Включить параметр «сдвига» в уравнение модели

$$g(x)^{\text{adj}} = g(x)_{logit} + b$$



$$b = \log\left(\frac{\pi_0\rho_1}{\pi_1\rho_0}\right)$$



  - Скорректировать вероятности на выходе модели:

$$p_1^{adj} = \frac{p_1\pi_1\rho_0}{p_1\pi_1\rho_0 + (1-p_1)\pi_0\rho_1}$$

π_1, π_0 - до undersampling

ρ_1, ρ_0 - после undersampling