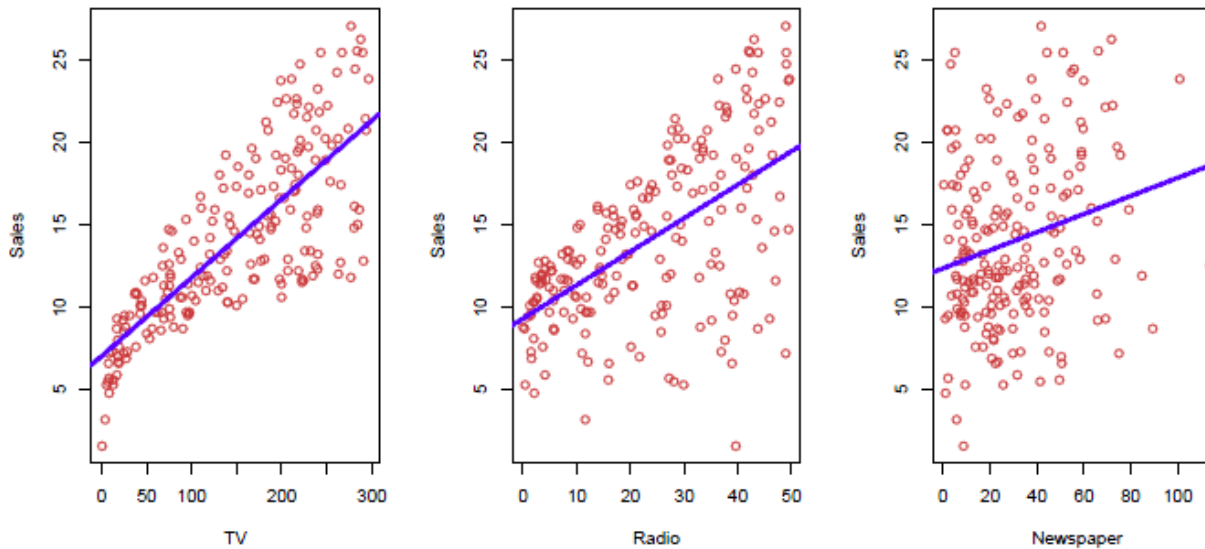


## Лекция 6: задача прогнозирования, проклятие размерности, переобучение

# Демонстрационный пример: что такое прогнозирование?



- Показаны *зависимости* продаж от каналов рекламы: ТВ, радио и газет— синяя прямая линейной регрессии отдельно для каждого параметра.
- Можно ли спрогнозировать продажи на основе этого всего в совокупности?
- Возможно мы можем сделать лучший прогноз, используя модель

$$\text{Sales} \approx f(\text{TV}, \text{Radio}, \text{Newspaper})$$

# Демонстрационный пример: обозначения

- Здесь Sales – это отклик или целевая переменная, которую мы хотим спрогнозировать. Обычно обозначаем отклик как  $Y$ .
- TV - это признак или вход; обозначим его как  $X_1$ . Признак Radio как  $X_2$ , и так далее.
- Тогда весь входной вектор можно обозначить как

$$X = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix}$$

- Можно описать всю модель как  $Y = f(X) + \epsilon$
- где  $\epsilon$  отражает ошибки измерения и другие отклонения.

# Задача «обучения с учителем»

- Множество «размеченных» примеров (прецедентов):
  - обучающая выборка или тренировочный набор:

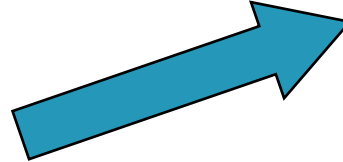
$$Z = \{(x_i, y_i)\}_{i=1}^n \in X \times Y$$

- $X$ : «сигнал», «объект», «ситуация»
  - $Y$ : «отклик», «прогнозируемая величина»
- Неформальная постановка задачи:

$$f_Z : X \rightarrow Y$$

- Два этапа: обучение и прогнозирование

# Обучение



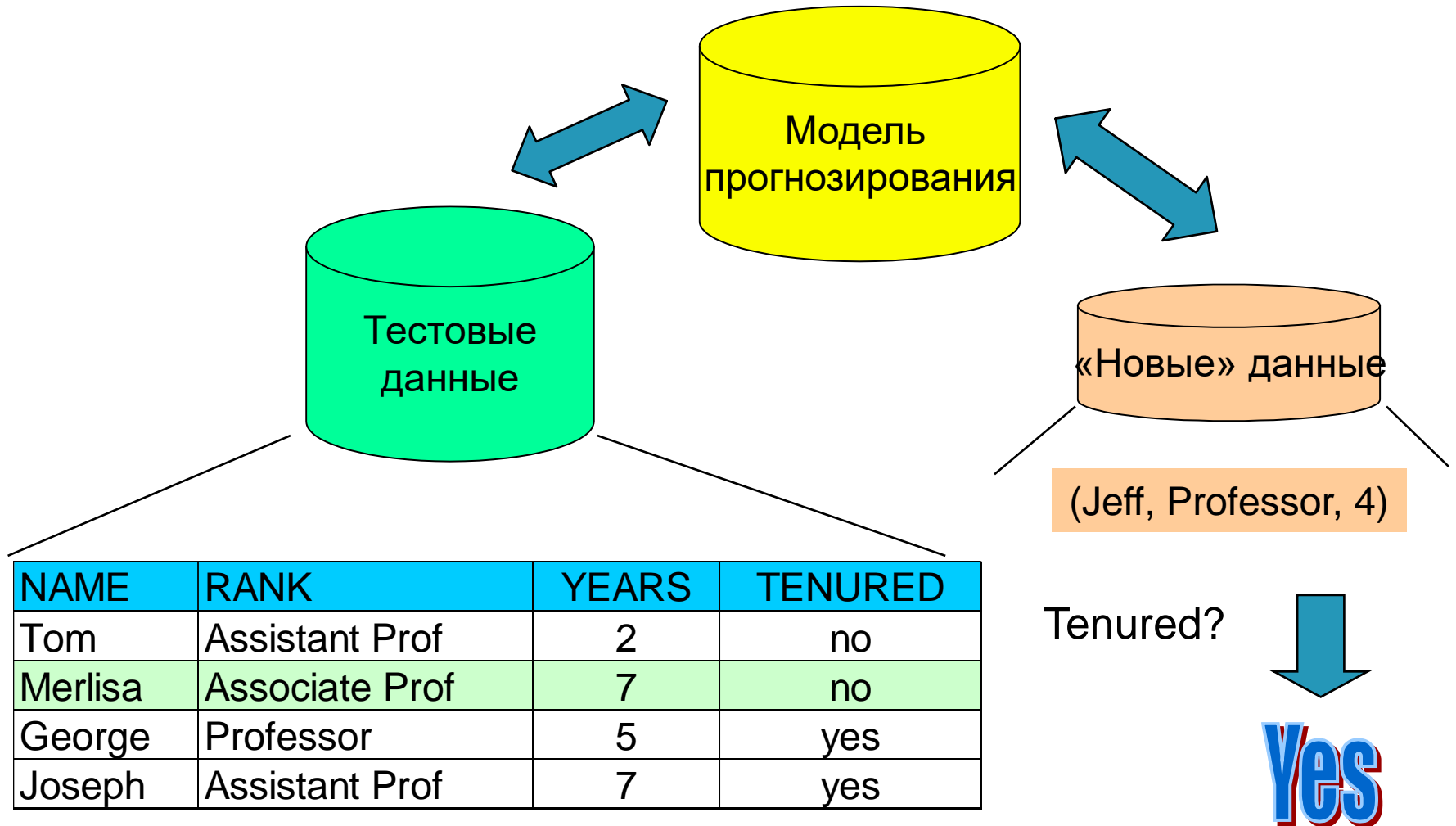
Метод машинного обучения



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

IF rank = 'professor'  
OR years > 6  
THEN tenured = 'yes'

# Оценка и прогнозирование



# Типы задач прогнозирования

- Определяются типом допустимых значений «отклика»  $y_i$  и той оценкой качества, которая используется для выбора модели
- Бинарная классификация (разделение):
  - $y_i$  - бинарная,  $f$  – бинарная функция
- Регрессия:
  - $y_i$  - вещественное число,  $f$  – вещественная функция
- Классификация:
  - $y_i$  - дискретная величина (метка класса), на  $Y$  нет порядка,  $f$  – дискретная функция
- Много-темная (multi-label) классификация:
  - $y_i$  – множество неупорядоченных дискретных величин (меток классов),  $f$  – бинарная вектор-функция ( $i$ -й разряд – «да»/«нет» для  $i$ -го класса)
- Порядковый (ординальный) прогноз:
  - $y_i$  – множество упорядоченных дискретных величин (меток классов),  $f$  – вещественная вектор-функция ( $i$ -й разряд – ранг для  $i$ -го класса)

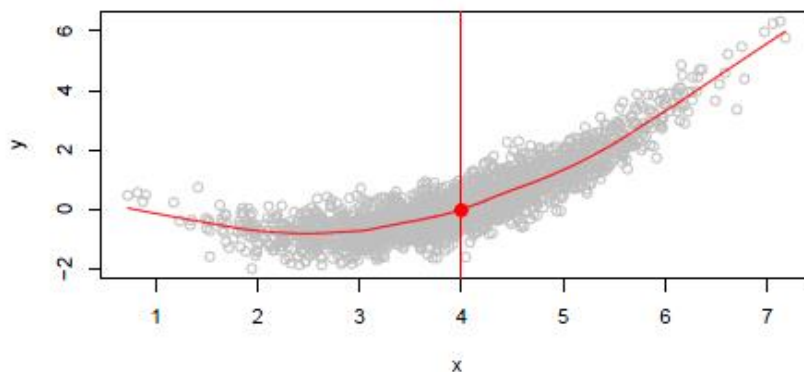
# Применение методов прогнозирования в задачах ИАД

- Прогнозирование ради прогнозирования:
  - Автоматическая классификация и прогнозирование (обучились и применяем модель для решения прикладной задачи)
  - Выявление и описание основных зависимостей, т.е. как значения характеристик примера влияют на отклик (важны интерпретируемость и визуализируемость модели)
- Предобработка данных:
  - «Условная» дискретизация (разбиение значений свойств примеров на интервалы с учетом отклика)
  - Обработка пропущенных значений (импутация)
- Поиск исключений и артефактов:
  - Что не соответствует прогнозу, то аномалия
  - Поиск и построение моделей «редких» (или малых) классов
- Области применения:
  - Везде, где необходим прогноз или классификация



# Что означает, что $f(X)$ дает «хороший» прогноз?

- Имея функцию  $f$  можно найти  $Y$  для новой точки  $X = x$ .
- Мы можем оценить, какая компонента  $X_j$  важна в объяснении  $Y$
- В зависимости от сложности функции  $f$ , мы можем понять как каждый компонент вектора  $X$  влияет на  $Y$ .



- Существует ли идеальная  $f(X)$ ? Какое «хорошее» значение  $f(X)$  для выбранного  $X$  (например,  $X = 4$ )? Может быть много значений  $Y$  для  $X$ . Хорошее значение таково, что  $f(4) = E(Y|X = 4)$
- $E(Y|X = 4)$  – ожидаемое значение (среднее) из  $Y$  для заданного.
- $f(x) = E(Y|X = x)$  называется функцией регрессии.

# Функция регрессии $f(x)$

- Аналогично определяется для вектора  $X$ , например:

$$f(x) = f(x_1, x_2, x_3) = E(Y|X_1 = x_1, X_2 = x_2, X_3 = x_3)$$

- Является оптимальным предиктором  $Y$  относительно среднеквадратичной ошибки прогнозирования:  $f(x) = E(Y|X = x)$  это функция которая минимизирует (для нормальной ошибки!)

$$E[(Y - g(X))^2|X = x]$$

по всем функциям  $g$  во всех точках  $X = x$ .

- $\epsilon = Y - f(x)$  - это *несокращаемая* ошибка, обычно существует распределение возможных значений.
- Для любой оценки  $\hat{f}(x)$  функции  $f(x)$ , мы имеем:

$$E[(Y - \hat{f}(X))^2|X = x] = \underbrace{[f(x) - \hat{f}(x)]^2}_{\text{Reducible}} + \underbrace{\text{Var}(\epsilon)}_{\text{Irreducible}}$$

# Другие функции потерь

- Функция потерь  $L = Y \times Y \rightarrow R$  характеризует отличие правильного ответа от спрогнозированного
- Примеры:

- Классификация и регрессия:

$$L(y, y') = [y \neq y'], L(y, y') = |y - y'|,$$

$$L(y, y') = (y - y')^2, L(y, y') = [|y - y'| > \delta]$$

- Много-темная классификация:

$$HL = |y \nabla y'|, a \nabla b = (a \cup b) \setminus (a \cap b), a \subseteq Y, b \subseteq Y$$

- Ранжирование:

$$RL = \frac{|\{(l, s) \in y \times \bar{y} : y_l \leq y_s\}|}{|y| |\bar{y}|}$$

- Оценка качества прогноза:

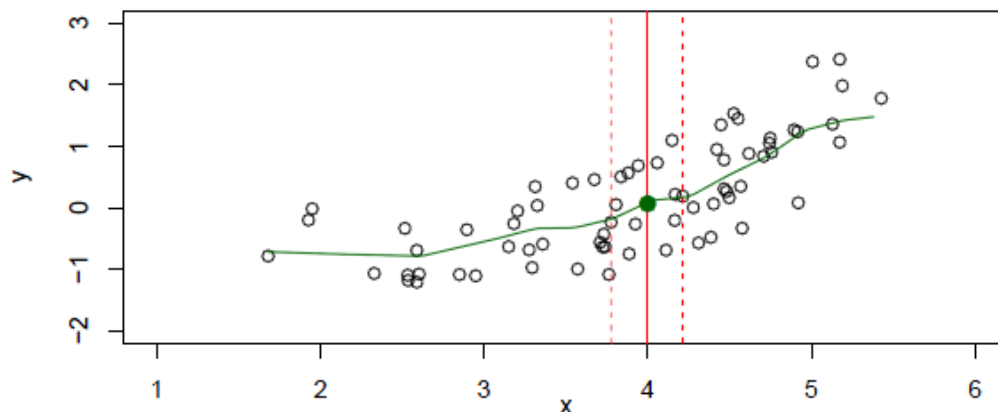
- Усреднение потерь по множеству примеров

# Как оценить $f$ ?

- Обычно мы имеем немного точек с одинаковым  $X$ .
- Таким образом мы не можем вычислить  $E(Y|X = x)$ !
- Определим

$$\hat{f}(x) = \text{Ave}(Y|X \in \mathcal{N}(x))$$

где  $\mathcal{N}(x)$  – некоторая окрестность точки  $x$ .



- Усреднение по ближайшим соседям может быть достаточно хорошо для малых  $p$  (число признаков) и больших  $N$  (число наблюдений).
- Методы ближайших соседей могут плохо работать при больших  $p$ .

# Метод K ближайших соседей

## ■ Общая схема работы:

- Каждый пример – точка в пространстве, все примеры хранятся
- Вводится метрика расстояния с учетом нормирования координат
- Ищется K (от 1 до ...) ближайших соседей
- Прогноз вычисляется как функция от откликов найденных соседей по одному из алгоритмов:

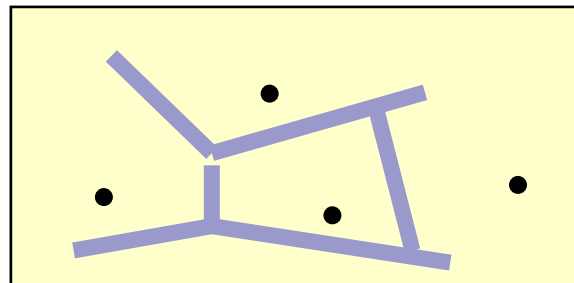
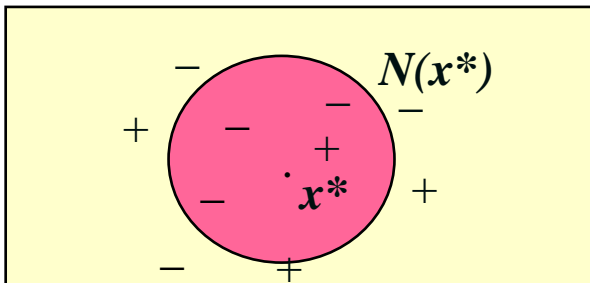
$$y^* = F_{x_i \in N(x^*)}(y_i)$$

## ■ Метод KNN:

- Для задачи регрессии отклик считается как среднее по откликам всех соседей:
- Для классификации выбирается самый частый класс:

$$y^* = \frac{1}{K} \sum_{x_i \in N(x^*)} y_i$$

$$y^* = \arg \max_{c \in C, x_i \in N(x^*)} [y_i = c]$$



# Метод «взвешенных» К ближайших соседей

## ■ Метод KWNN:

- На базе KNN, но помимо распределения «отклика» учитываются и расстояния до соседей в окрестности
- Учет происходит за счет «взвешенного» голосования для классификации:

$$y^* = \arg \max_{c \in C, x_i \in N(x^*)} \left[ \frac{w_i |y_i = c|}{\sum_{x_j \in N(x^*)} w_j} \right]$$

- И «взвешенного» среднего для регрессии

$$y^* = \frac{\sum_{x_i \in N(x^*)} w_i y_i}{\sum_{x_i \in N(x^*)} w_i}$$

- весовой коэффициент обратно пропорционален квадрату расстояния или пропорционален корреляции с откликом

# Метод К ближайших соседей с адаптивным расстоянием

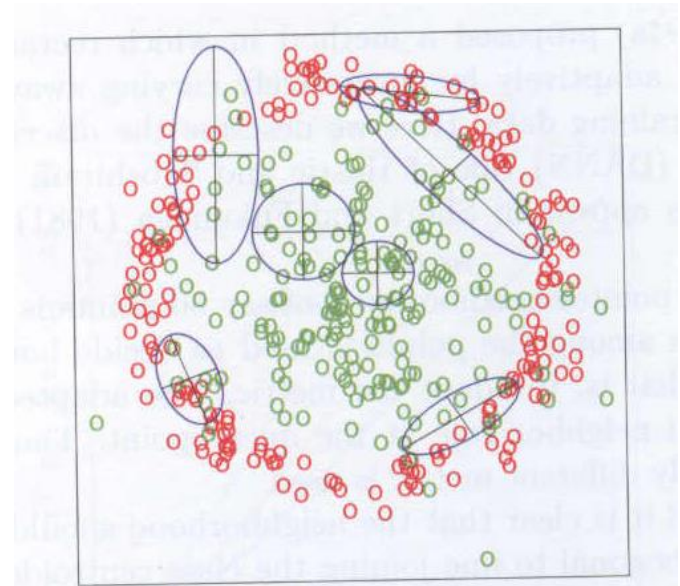
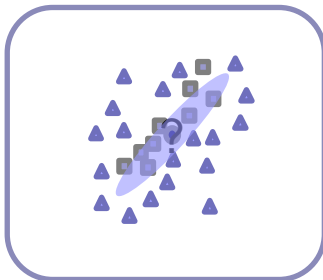
## ■ Метод DANN:

- На базе KNN, но используется локальный дискриминантный анализ для адаптации метрики расстояния с учетом структуры распределения соседей в окрестности:

## ■ Параметры алгоритма:

$$d^{(l)}(x^*, x_i) = (x^* - x_i)^T \Sigma^{(l)} (x^* - x_i)$$

- $K_M$  – число соседей для оценки метрики (нужно побольше)
- $K$  – число соседей для прогноза (лучше поменьше)
- $\varepsilon$  – «смягчающий» параметр



# Метод К ближайших соседей с адаптивным расстоянием

## ■ Процедура:

1. Инициализация метрики единичной матрицей  $\Sigma = I$
2. Поиск К ближайших соседей вокруг  $x^*$  в метрике  $\Sigma$ .
3. Расчет  $W$  - взвешенной суммы внутриклассовых ковариационных матриц:

$$W = \sum_{c \in C} \sum_{x_k \in N(x^*), y_k = c} \pi_k (x_k - \bar{x}_c)(x_k - \bar{x}_c)^T$$

4. ... и  $B$  - взвешенной суммы межклассовых ковариационных матриц:

$$B = \sum_K \pi_k (x_k - \bar{x})(x_k - \bar{x})^T$$

5. Новая метрика:

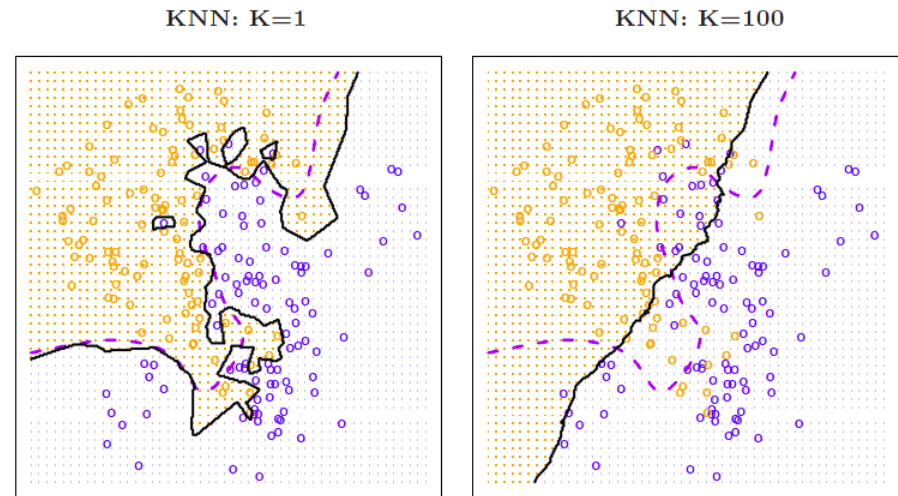
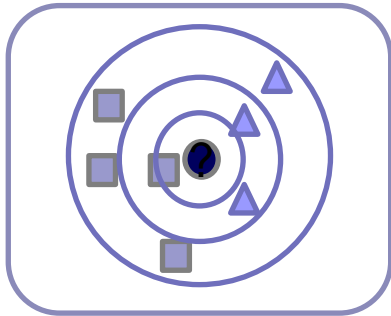
$$\Sigma^{(l+1)} = W^{-1/2} [W^{-1/2} B W^{-1/2} + \varepsilon I] W^{-1/2}$$

6. Повторить шаги 2-5 заданное число раз
7. Применить стандартный KNN



# Выбор параметра K

- Важность K:
- $k = 1$ : Результат = квадрат
- $k = 5$ : Результат = треугольник
- $k = 7$ : Снова квадрат



## ■ Выбор $k$ :

- Если  $k$  мал, то чувствительность к шуму, и негладкие границы классов
- Если  $k$  велико, то окрестность может сильно «задеть» соседний класс, зато гладкие границы. При классификации надо использовать нечетный  $k$ , чтобы не было «ничьей»
- Выбирается кросс-валидацией или на валидационном наборе
- Стандартная эвристика  $k = \sqrt{n}$

# Свойства методов KNN

## ■ Основные свойства:

- ☐ «Ленивый классификатор» - не надо ничего обучать
- ☐ Качество классификации зависит, в основном, от структуры данных, от параметров в меньшей степени
- ☐ Обязательно нужна хорошая метрика и нормированные атрибуты

## ■ Достоинства:

- ☐ Простой и легко реализуемый
- ☐ Один из самых точных
- ☐ Легко адаптируется под сложные типы «откликов», включая ранжирование, многотемность и т.д.
- ☐ Можно интегрировать экспертные знания, задавая веса у примеров, или параметры у метрики

## ■ Недостатки:

- ☐ «черный ящик» - результат не интерпретируемый совсем
- ☐ Достаточно вычислительно трудоемкий, проблема использования индексов для сложных структур  $X$
- ☐ **«Проклятие размерности»**

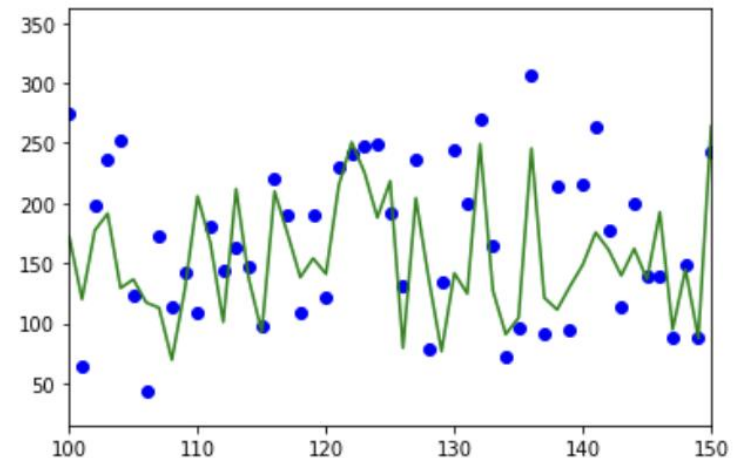
# Пример (Python)

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets import load_diabetes
```

```
N = 200
data = load_diabetes()
X, X_test = data.data[:N], data.data[N:]
y, y_test = data.target[:N], data.target[N:]
```

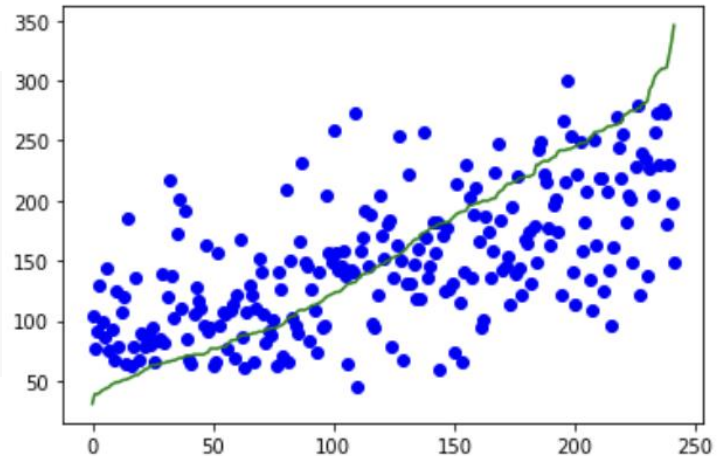
```
# weights="uniform" is default
# weights="distance" is for KNN
# weights as user function: distances -> weight (implement DANN)
KNN = KNeighborsRegressor(n_neighbors=5, weights="distance")
KNN.fit(X, y)
pass
```

```
plt.scatter(range(len(y_test)), y_test, color="blue")
plt.plot(KNN.predict(X_test), color="green")
plt.xlim([100, 150])
```

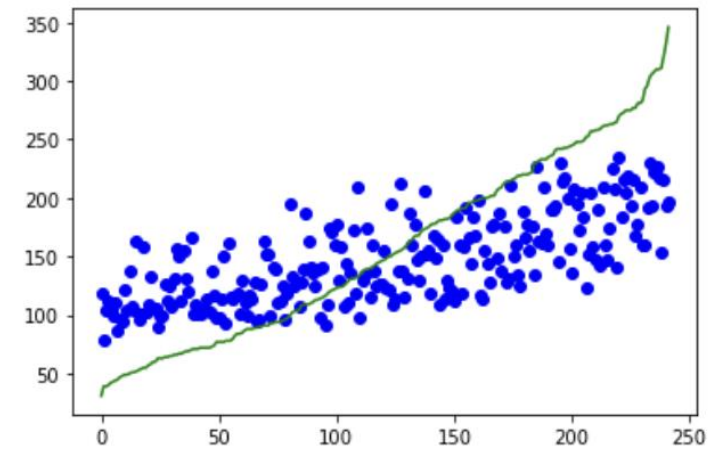


# Пример (Python)

```
KNN = KNeighborsRegressor(n_neighbors=3, weights="distance")
KNN.fit(X, y)
y_pred=KNN.predict(X_test)
rs=pd.DataFrame([y_pred, y_test]).T
rs.sort_values(1,inplace=True)
plt.scatter(range(len(rs[0])), [rs[0]], color="blue")
plt.plot(range(len(rs[1])),rs[1], color="green")
```

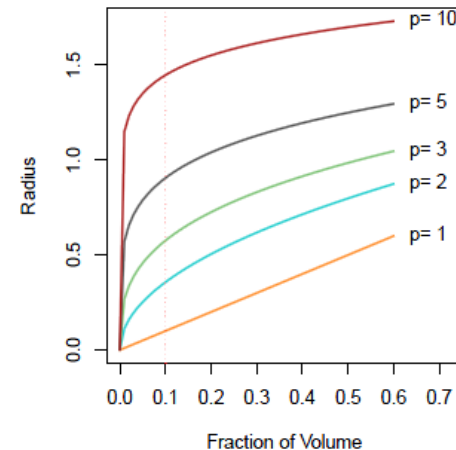
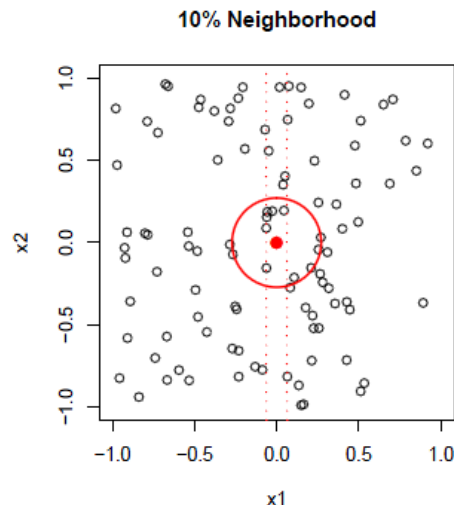


```
KNN = KNeighborsRegressor(n_neighbors=30, weights="distance")
KNN.fit(X, y)
y_pred=KNN.predict(X_test)
rs=pd.DataFrame([y_pred, y_test]).T
rs.sort_values(1,inplace=True)
plt.scatter(range(len(rs[0])), [rs[0]], color="blue")
plt.plot(range(len(rs[1])),rs[1], color="green")
```

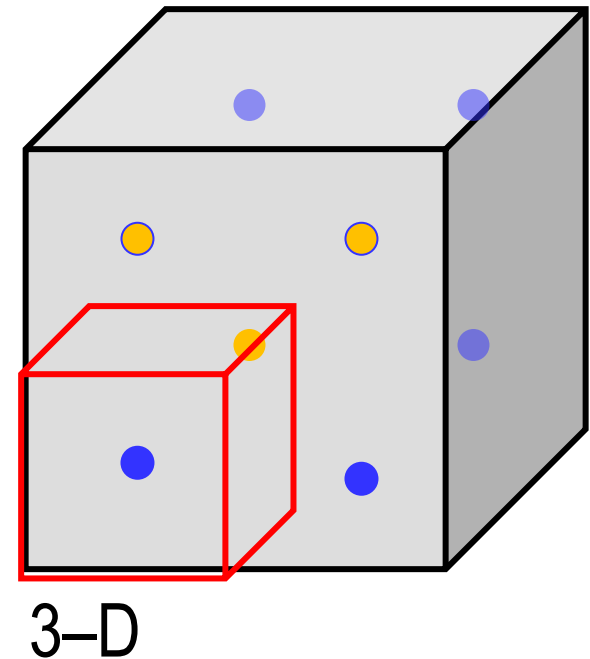
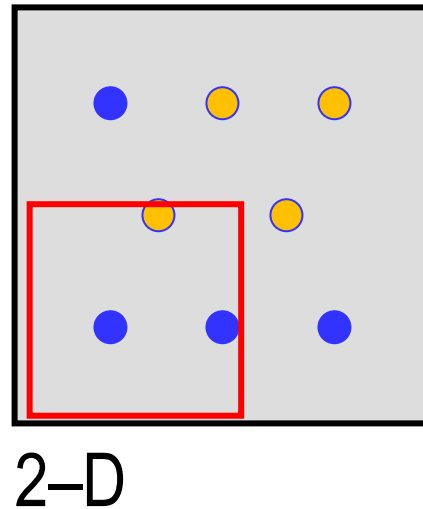
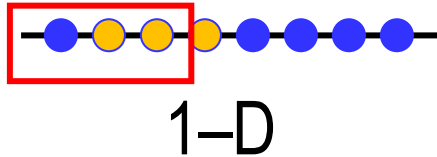


# Проклятие размерности

- Ближайшие соседи как правило расположены далеко при больших размерностях.
  - Нам нужно получить значительную часть из  $N$  значений  $y_i$ , чтобы снизить дисперсию - например, 10%.
  - 10% соседей для случая больших размерностей не может быть локализована, так что мы уже можем сделать оценку  $E(Y|X = x)$  на основе локального усреднения.



# Модельный пример, демонстрирующий проклятие размерности



- $r = K/N$
- $E_p(r) = r^{1/p}$
- $E_{10}(0.01) = 0.63$
- $E_{10}(0.1) = 0.8$

# Параметрические модели

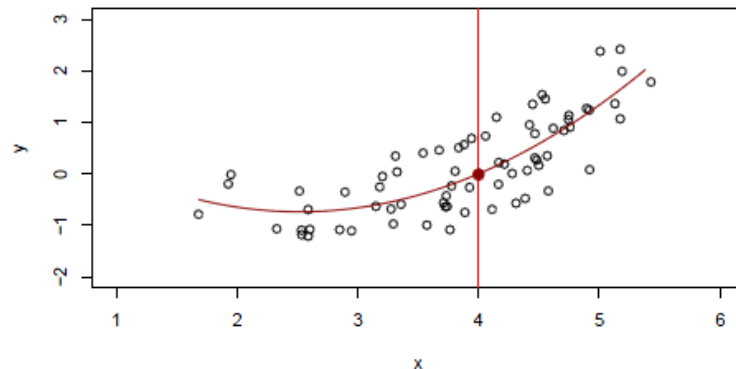
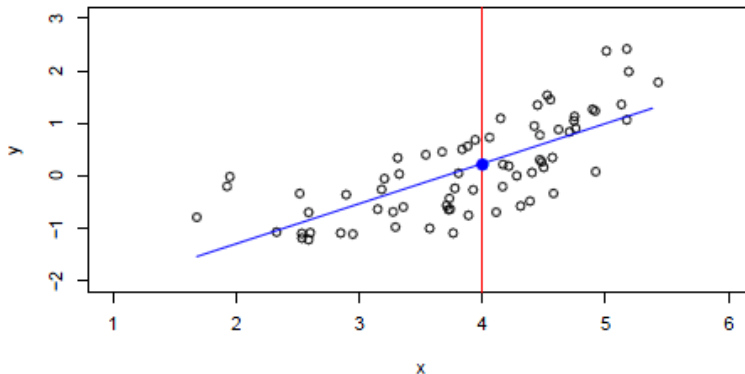
Линейная модель представляет собой важный пример параметрической модели:

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

- Линейная модель определяется  $p + 1$  параметрами

$$\beta_0, \beta_1, \dots, \beta_p.$$

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



# Линейная регрессия

- Задача регрессии:

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

- Уравнение линейной регрессии:

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

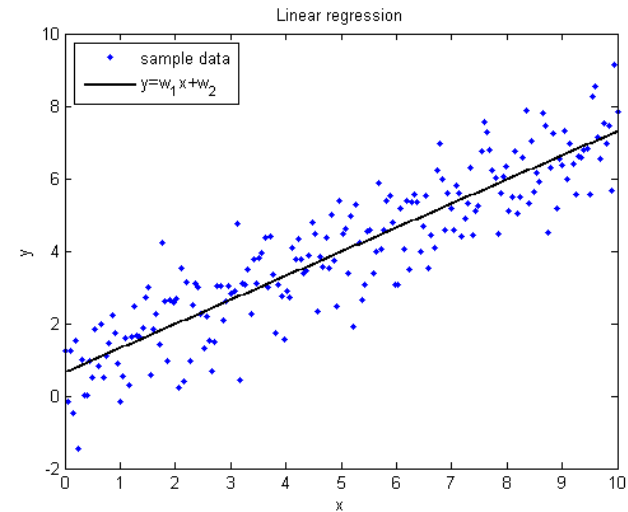
- $\varepsilon = N(0, \sigma^2)$  - шум
- $Y$  –отклик
- $X=(X_1, \dots, X_p)$  - регрессоры (предикторы)
- $b$  – параметры модели

- Линеаризируемые регрессии:

- Степенная ,Экспоненциальная
- Гиперболическая, и другие

- Цель регрессионного анализа:

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



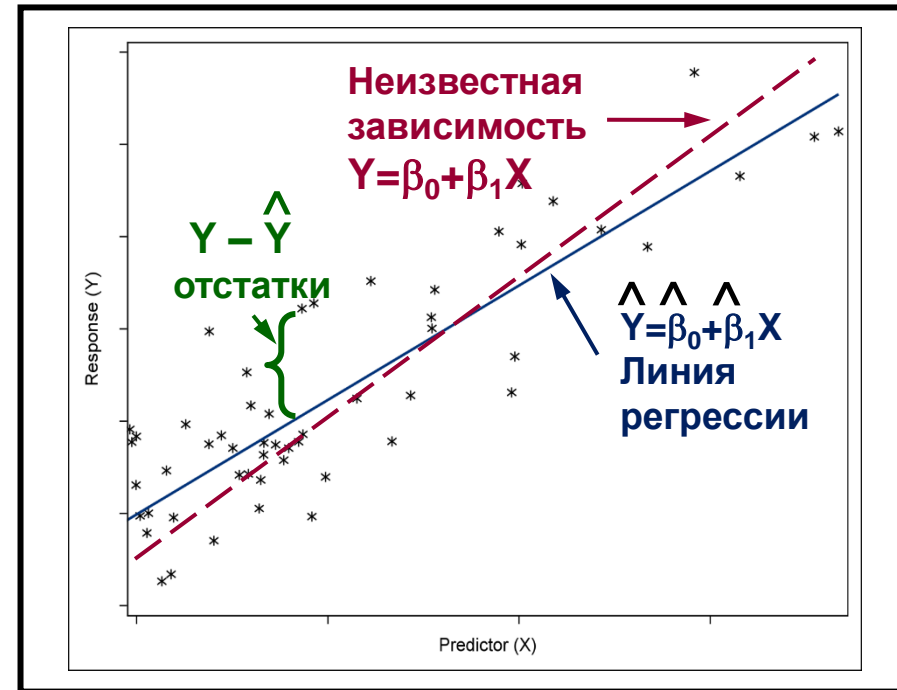
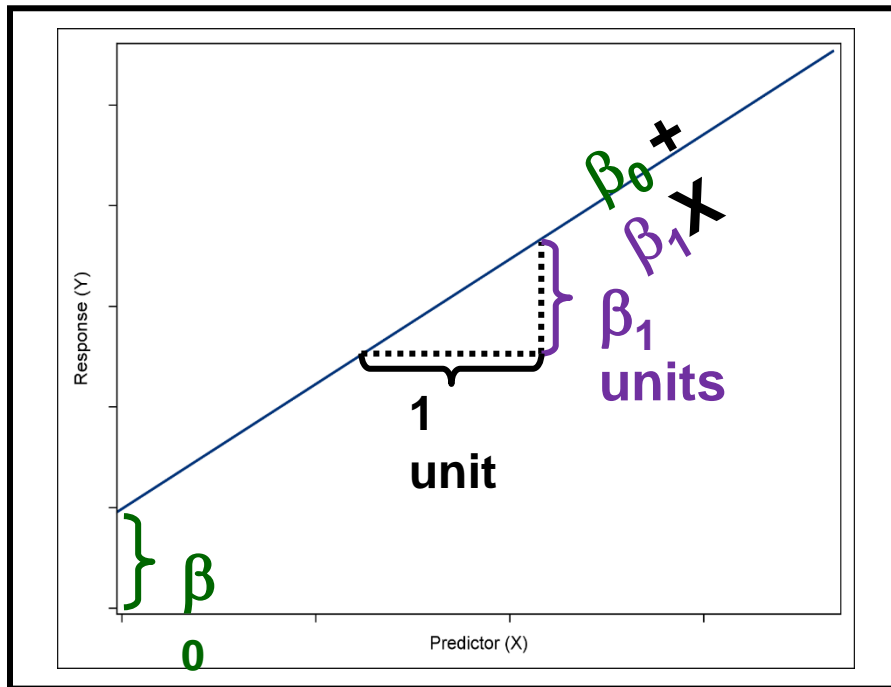
$$y = ax_1^{b_1} x_2^{b_2} \dots x_p^{b_p} \varepsilon,$$

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

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



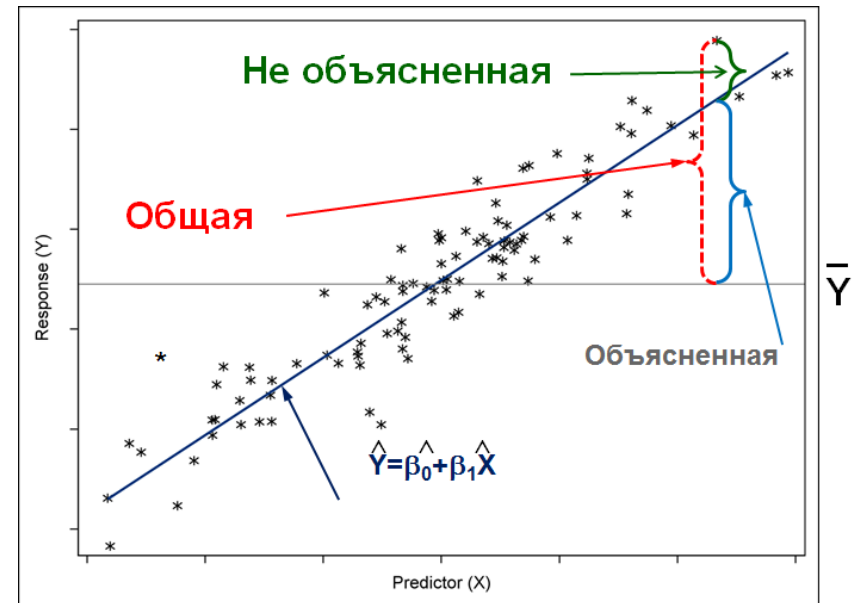
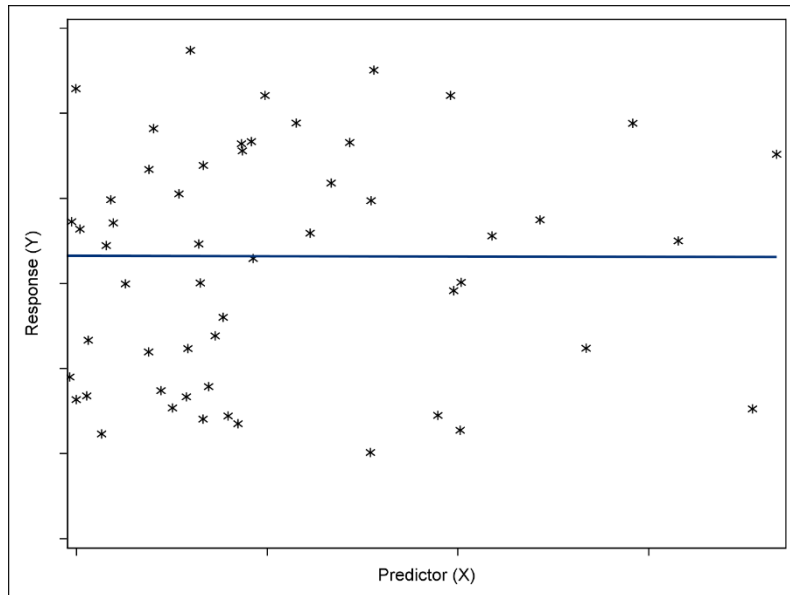
# Простая линейная регрессия



Предположения:

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

# Базовая модель (Нулевая гипотеза)



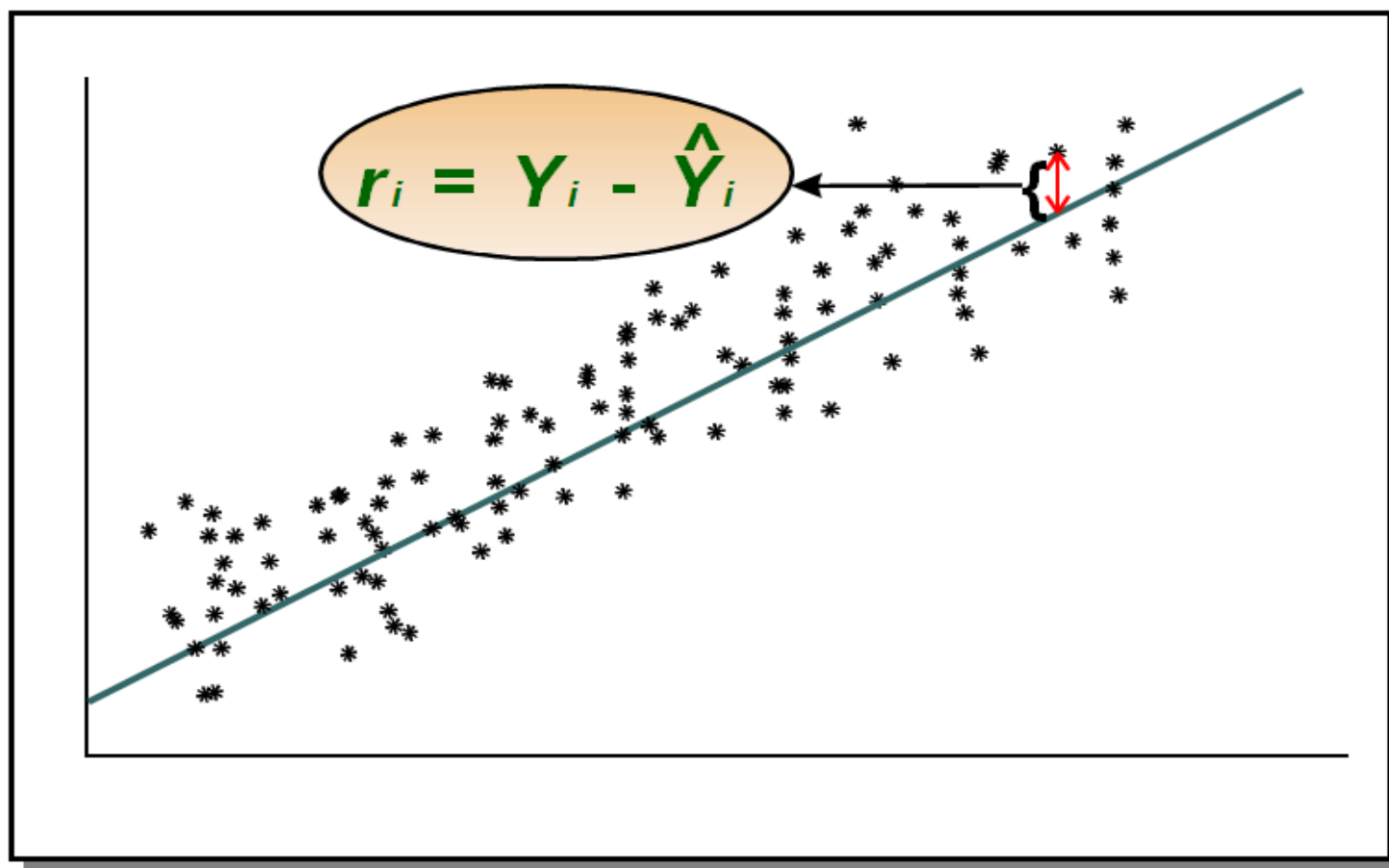
## ■ Нулевая гипотеза:

- Регрессионная модель приближает наблюдаемые данные не лучше базовой модели – константы ( $\beta_1=0$ )

## ■ Альтернативная гипотеза:

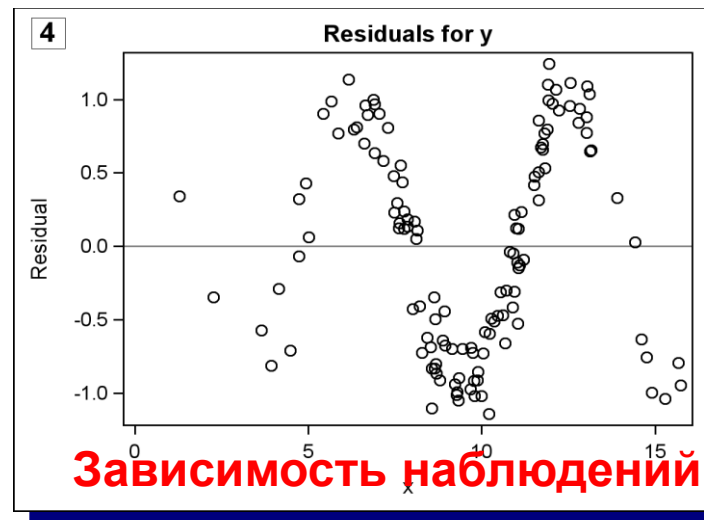
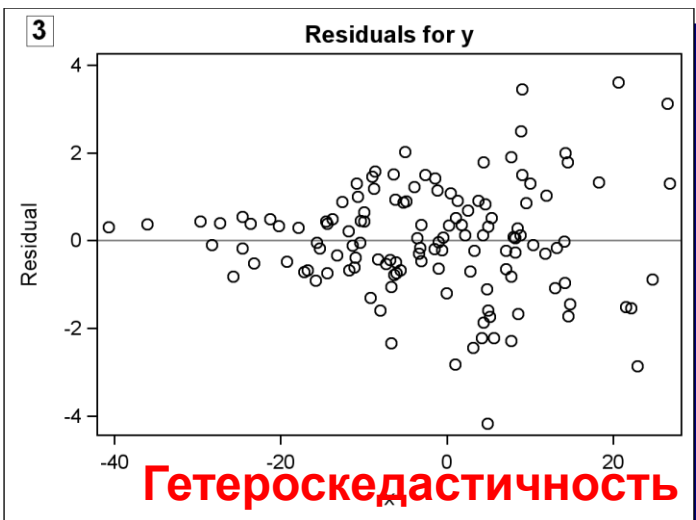
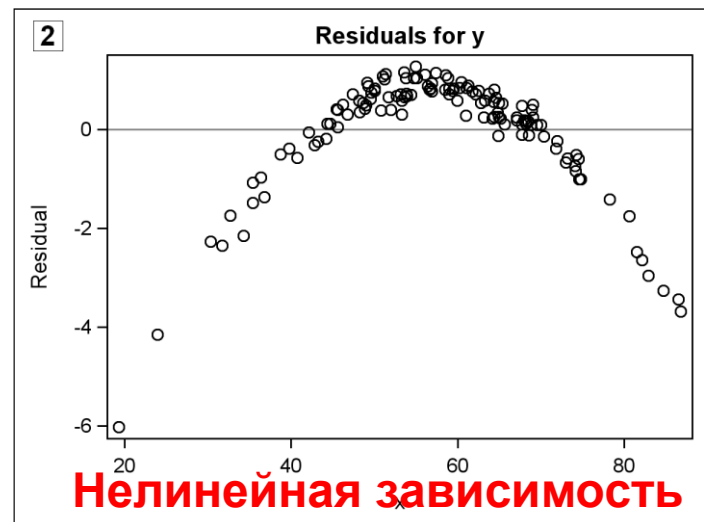
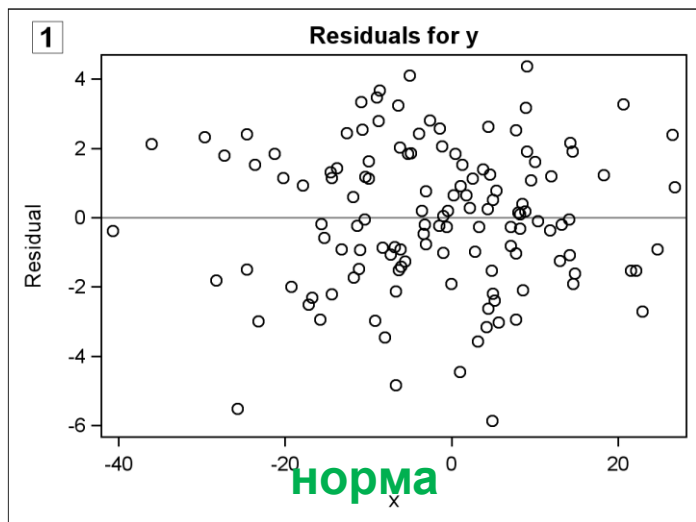
- Регрессионная модель лучше приближает наблюдаемые данные чем базовая модель – константа ( $\beta_1 \neq 0$ )

# Проверка предположений модели с помощью графиков остатков



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

# Графики остатков

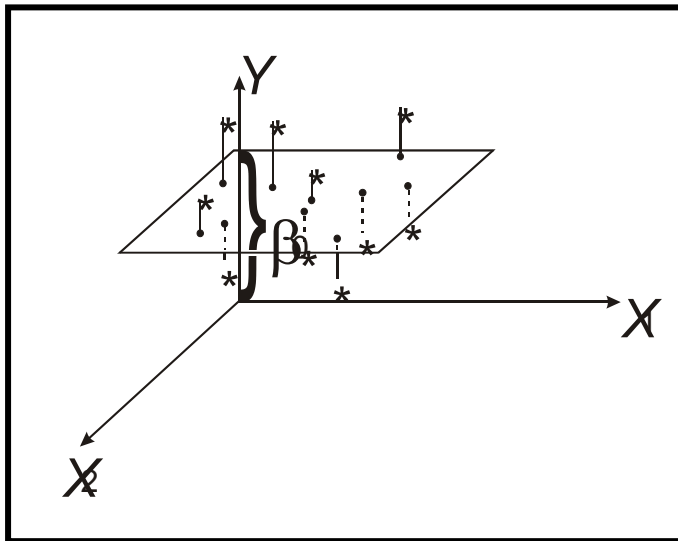


# Множественная линейная регрессия

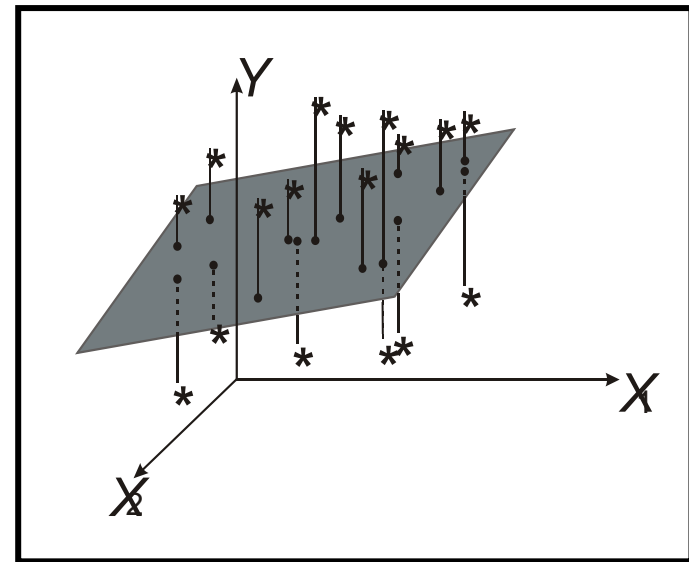
- Пример линейной модели с двумя переменными

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon, \text{ где}$$

$Y$  – отклик,  $X_1$  и  $X_2$  предикторы,  $\varepsilon$  - ошибка,  $\beta_0$ ,  $\beta_1$ , и  $\beta_2$ -параметры (неизвестные)



Нет зависимости

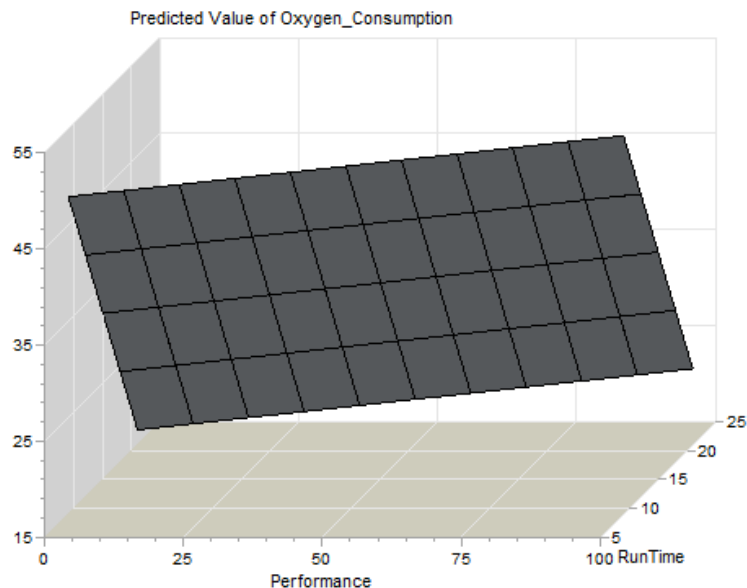


Есть зависимость

# Множественная линейная регрессия

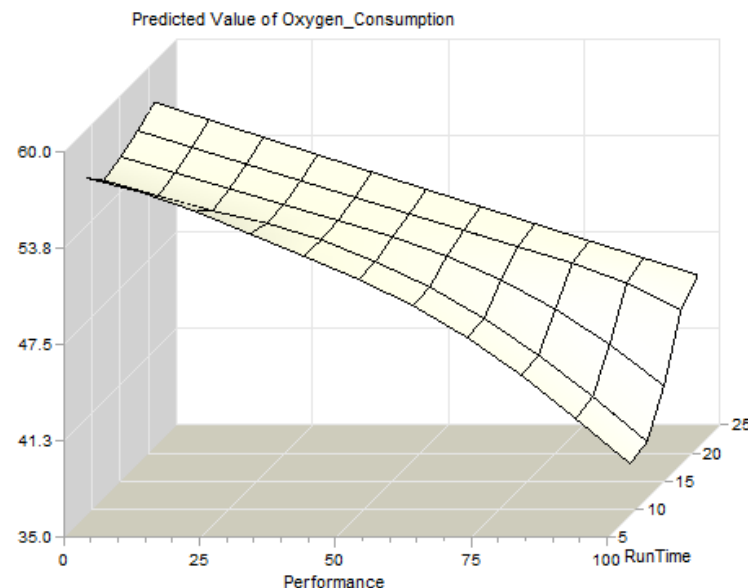
- В общем случае ищем зависимость как линейную комбинацию  $k$  предикторов  $X_1 - X_k$ :

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_k X_k + \varepsilon$$



$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$$

Линейная модель с  
линейными эффектами



$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2 + \beta_4 X_2^2 + \varepsilon$$

Линейная модель с нелинейными  
эффектами

# Метод наименьших квадратов и проблема мультиколлинеарности

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

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

- В матричной форме:

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

- Единственное оптимальное решение (если матрица данных не сингулярная)

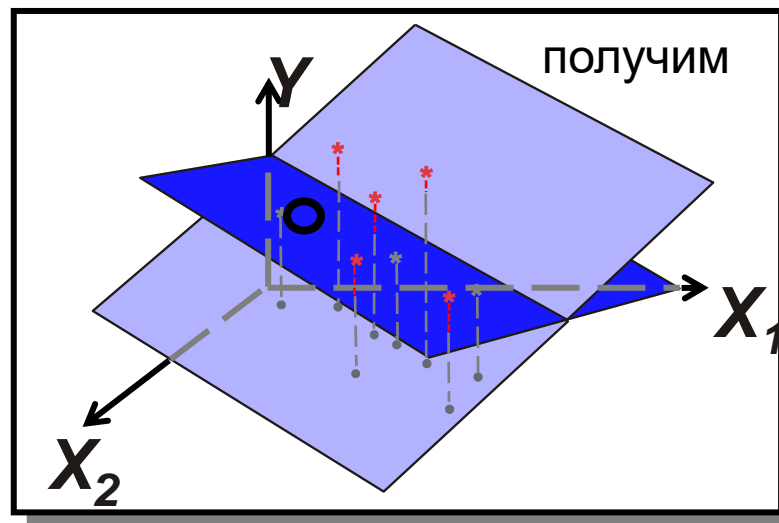
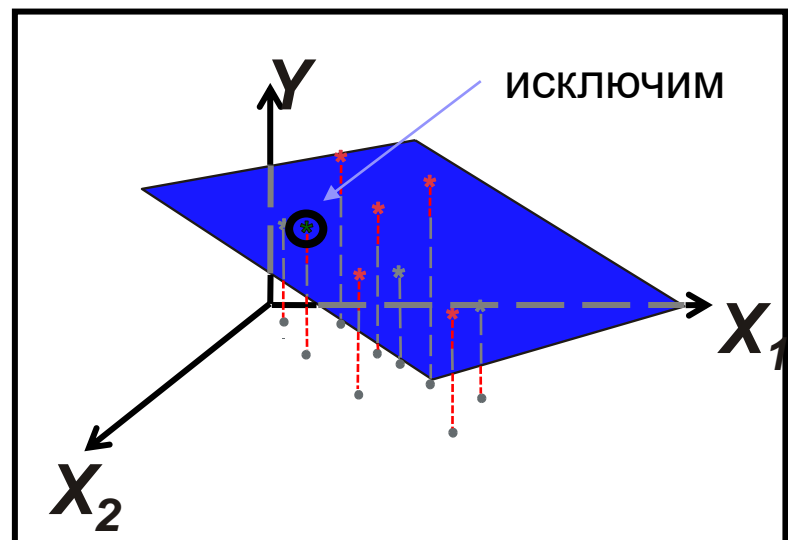
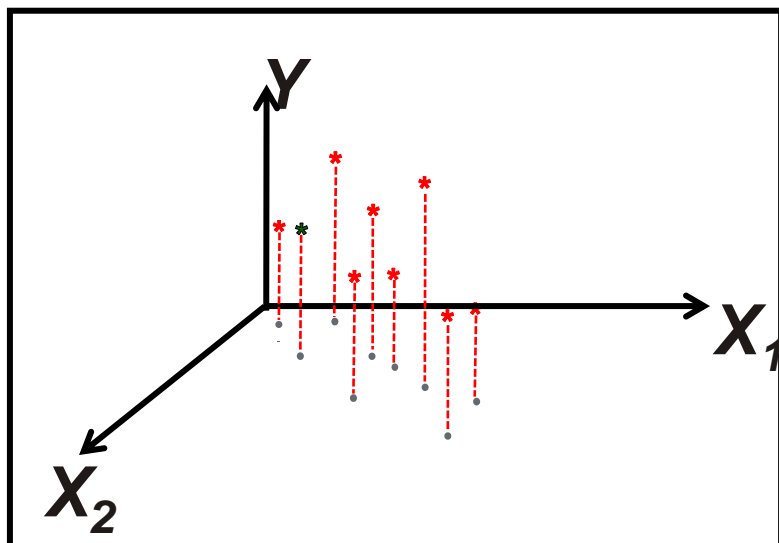
- Недостатки:  $B = (X^T X)^{-1} X^T \bar{y}$

- ☐ Сингулярная матрица данных из-за коррелированных факторов
- ☐ Большое число регрессоров – плохая точность и интерпретируемость

- Основные подходы:

- ☐ Поиск и удаление зависимых и незначимых факторов
- ☐ Использование «смещенных» регуляризированных моделей
- ☐ переход к новым независимым факторам, например, с помощью метода главных компонент

# Иллюстрация мультиколлинеарности



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



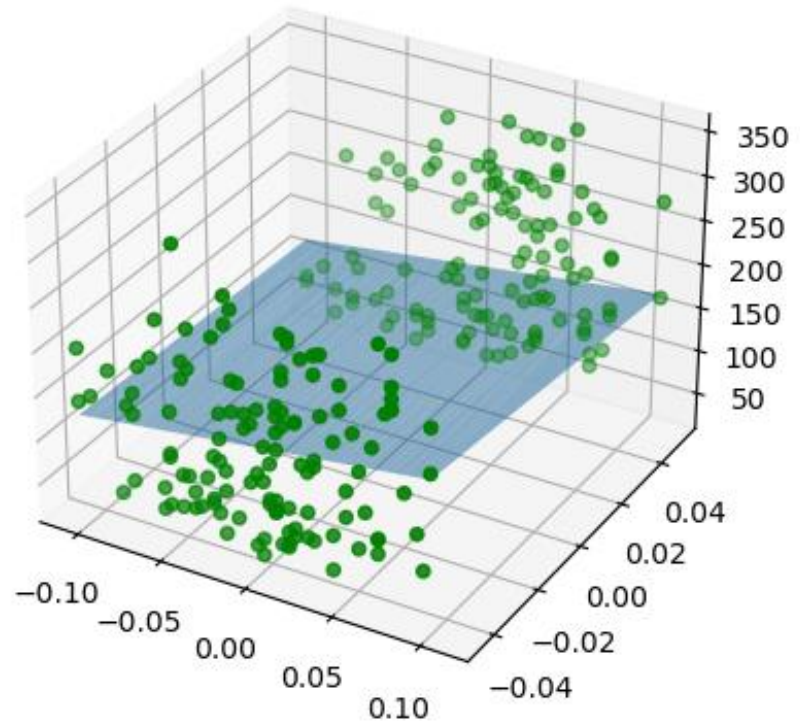
# Пример (Python)

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

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

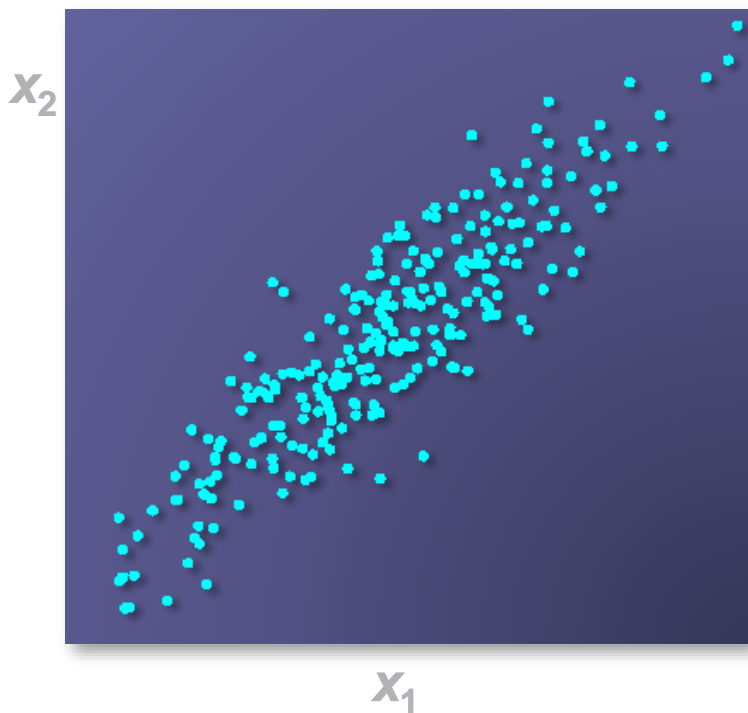
```
regr = LinearRegression()  
regr.fit(X_2d, y)  
pass
```

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

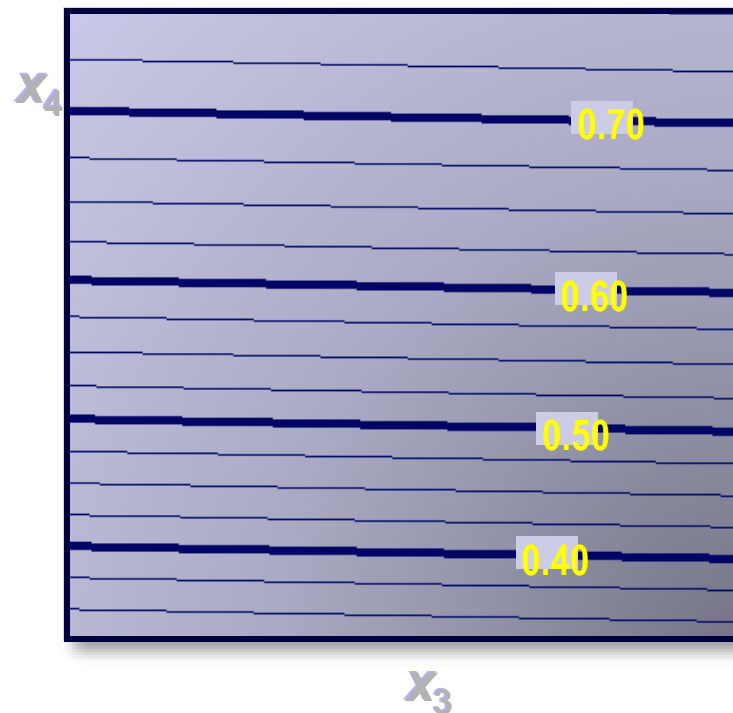


# Проблемы входных переменных и для KNN и для МНК

Зависимость



Не релевантность  
отклику



Выхода два: либо преобразование либо исключение

# Простые методы сокращения размерности

**Дано:** входные переменные  $\{x_1, \dots, x_n\}$  и выходная (числовая или бинарная)  $y$

**Задача:** оставить только значимые и независимые  $x_i$

Работает в два этапа:

1. Удаляет все  $x_i$ , где  $R^2(x_i) < T1$  - *(удаление незначимых)*
2. Forward stepwise регрессия  $f(x_{i_1}, \dots, x_{i_k})$  пока  $R^2(f(x_{i_1}, \dots, x_{i_k})) - R^2(f(x_{i_1}, \dots, x_{i_{k-1}})) > T2$   
- *(удаление зависимых)*

Преобразования переменных:

- Дискретизация непрерывных
- Группировка категориальных

# Пример (Python)

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

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

```
VT = VarianceThreshold(threshold=0.002)
```

```
X_ = VT.fit_transform(X)
```

```
X.shape, X_.shape
```

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

```
# KNN.score(X, y) returns R2 - performed until tol is satisfied
```

```
# direction: forward/backward - add or remove features
```

```
# for newer scikit-learn versions:
```

```
"""
```

```
selector = SequentialFeatureSelector(KNN, n_features_to_select="auto", tol=0.5, direction="backward")
```

```
"""
```

```
# for older scikit-learn versions:
```

```
selector = SequentialFeatureSelector(KNN, n_features_to_select=3, direction="backward")
```

```
selector.fit(X_, y)
```

```
X_ = selector.transform(X_)
```

Недостаток – не сохраняет значимости на каждой итерации

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

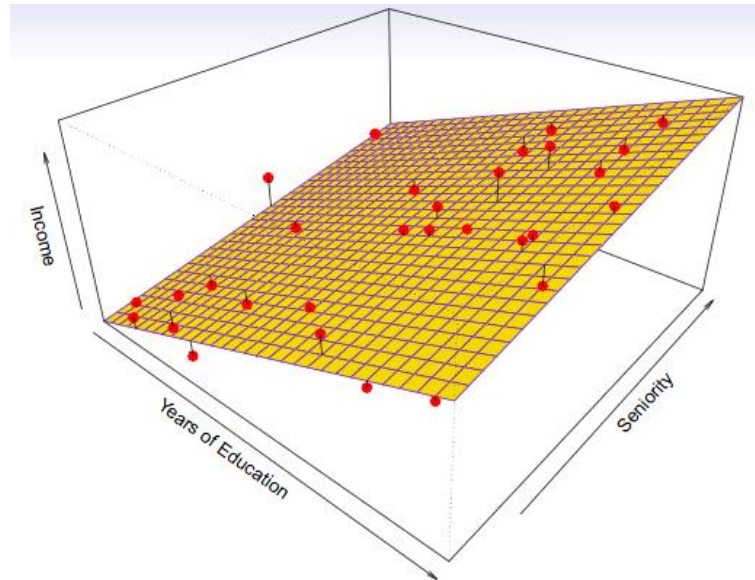
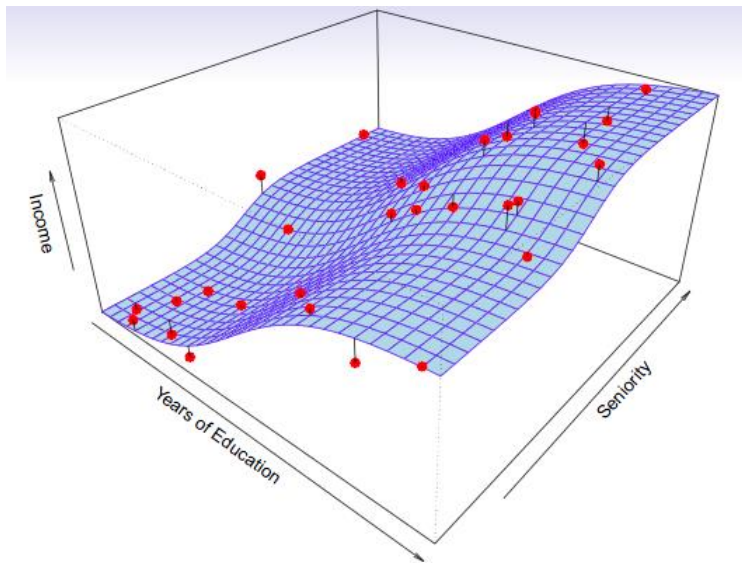
# Простые методы сокращения размерности

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

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

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

# Проблема недообучения и переобучения



Модельный пример.

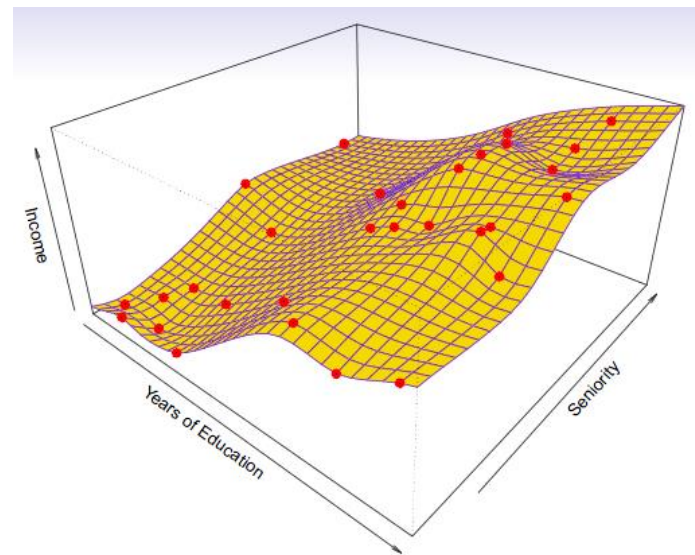
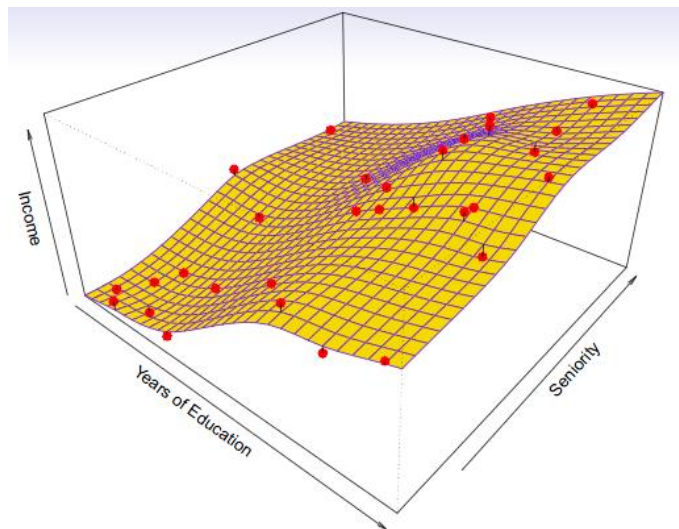
- Красные точки - наблюдения, синяя поверхность – истинная зависимость  $\text{income} = f(\text{education}, \text{seniority}) + \epsilon$

- Желтая поверхность линейная модель

$$\hat{f}_L(\text{education}, \text{seniority}) = \hat{\beta}_0 + \hat{\beta}_1 \times \text{education} + \hat{\beta}_2 \times \text{seniority}$$

- Плохая точность приближения

# Проблема недообучения и переобучения



Модельный пример.

- Более сложные модели (сплайны или полиномиальные регрессии или нейронные сети или еще что-то)
- Справа модель не допускает ошибок на обучающем наборе.
- Это хорошо? Нет!

# Переобучение

- Основная проблема методов машинного обучения!!!
- По сути:
  - Высокая точность на тренировочном наборе и плохая на тестовом
- Причины:
  - Сложность модели: например, для параметрических моделей много степеней свободы (параметров модели) или слишком сложное уравнение
  - Шум и выбросы в тренировочной выборке
  - Малый объем или неравномерность тренировочной выборки
- Обобщающая способность:
  - способность метода машинного обучения правильно прогнозировать «отклик» для объектов и ситуаций, которых не было в тренировочном наборе
  - метод называется состоятельным, если он с большой вероятностью делает маленькую ошибку на данных, которых не было в обучающей выборке
  - Как оценить?



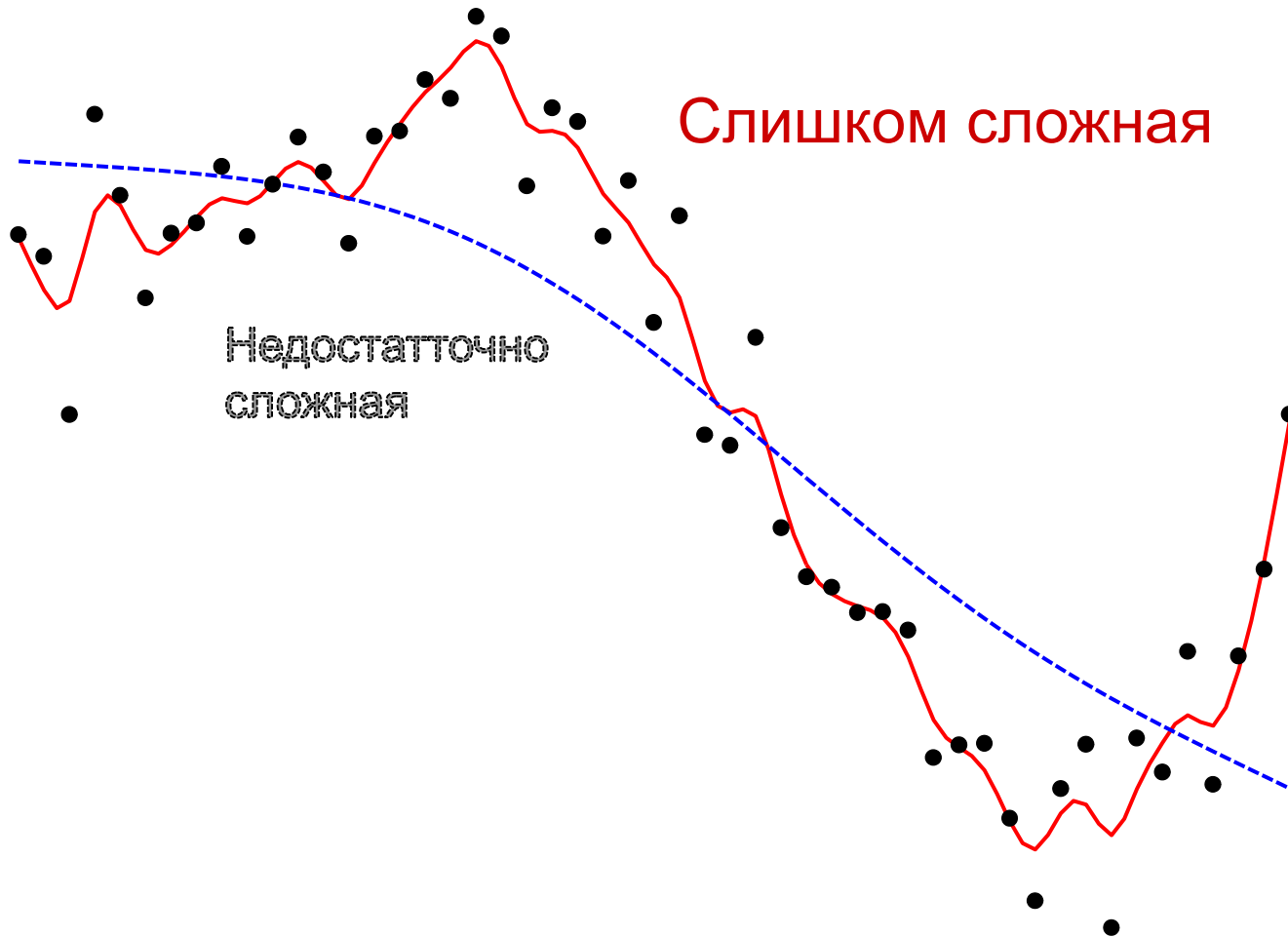
# Сложность модели



# Сложность модели



# Сложность модели



# Экспериментальная оценка качества модели

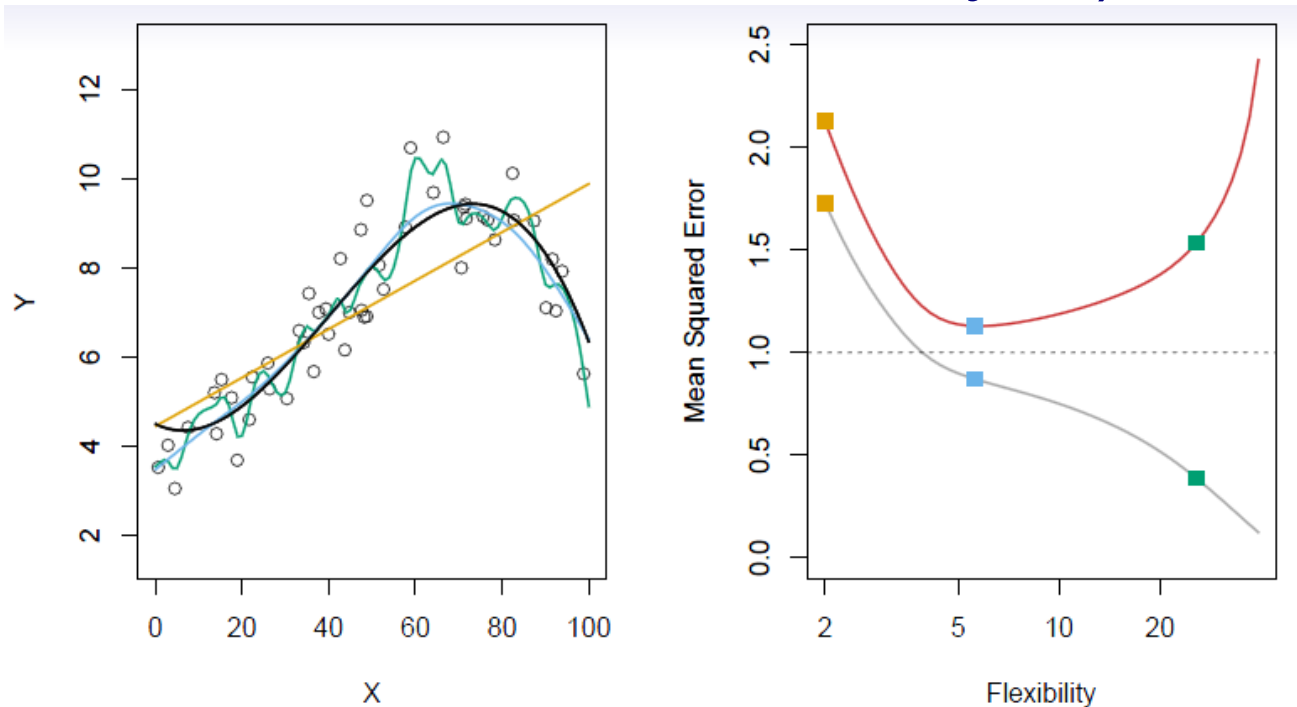
- Предположим, что мы строим модель  $\hat{f}(x)$  на обучающем наборе данных  $Tr = \{x_i, y_i\}_1^N$  и хотим, чтобы она была наилучшей.
  - Мы можем вычислить среднеквадратичную ошибку прогнозирования для  $Tr$ .

$$MSE_{Tr} = Ave_{i \in Tr} [y_i - \hat{f}(x_i)]^2$$

- Оценка может быть смещена в сторону более сложных моделей.
  - Вместо этого мы можем, если возможно вычислить оценку, используя тестовый набор данных  $Te = \{x_i, y_i\}_1^M$ :

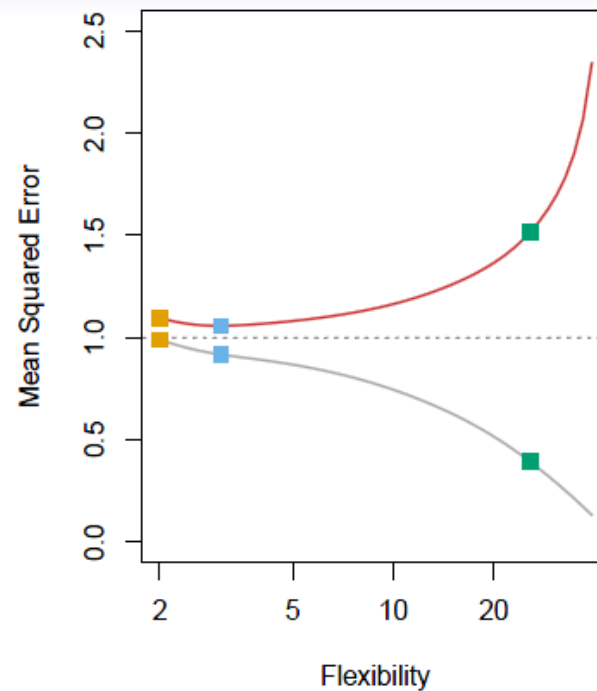
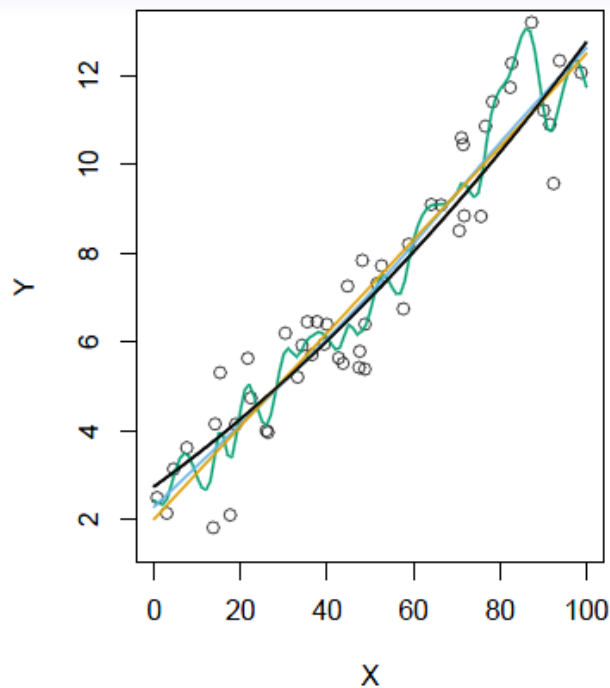
$$MSE_{Te} = Ave_{i \in Te} [y_i - \hat{f}(x_i)]^2$$

## Оценка качества модели (сложная зависимость, много шума)



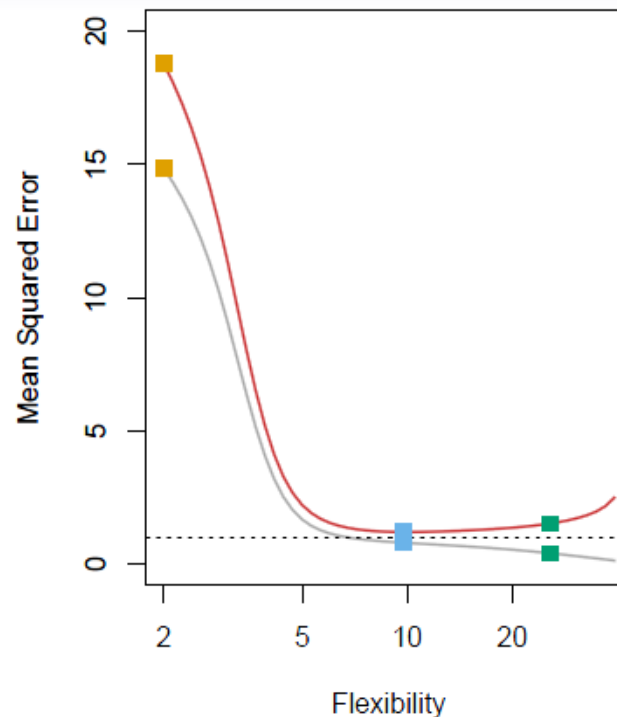
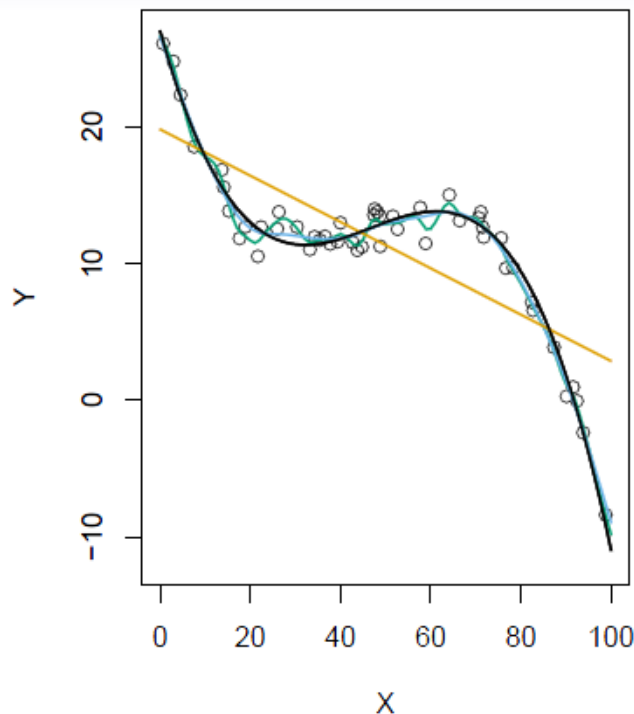
- Кривая, обозначенная черным цветом, - истинные значения.
- Красная кривая на правом рисунке –  $MSE_{Te}$ , серая кривая –  $MSE_{Tr}$ .
- Оранжевая, голубая и зеленая кривые соответствуют подгонке моделей различной сложности.
- Простые модели недообучены, сложные модели переобучены

# Оценка качества модели (простая зависимость, много шума)



- Простые модели дают высокую обобщающую способность
- Сложные модели переобучены

# Оценка качества модели (сложная зависимость, мало шума)



- Простые модели недообучены
- Сложные обладают хорошей обобщающей способностью

# Некоторые интуитивно понятные компромиссы

- Точность прогноза vs интерпретируемость.
  - Линейные модели легко интерпретируемы, тогда как более гибкие модели как правило - нет.
- Хорошее качество подгонки vs переобучение или недообучение.
  - Как определить, в какой момент подгонка наиболее точная?
- Простота vs черный ящик.
  - Мы часто предпочитаем более простую модель с участием меньшего количества переменных по сравнению с прогнозированием черным ящиком с участием их всех.



# Компромис отклонения и смещения

Пусть мы строим модель  $\hat{f}(x)$  на некотором обучающем наборе  $Tr$ , и пусть  $(x_0, y_0)$  - некоторый тестовый образец. Если истинная модель  $Y = f(X) + \epsilon$  ( $f(x) = E(Y|X = x)$ ), то

$$E \left( y_0 - \hat{f}(x_0) \right)^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon).$$

Заметим, что

$$\text{Bias}(\hat{f}(x_0)) = E[\hat{f}(x_0)] - f(x_0).$$

Как правило, когда сложность  $\hat{f}(x)$  увеличивается, дисперсия возрастает, а смещение уменьшается. Таким образом, выбор сложности, основанный на средних ошибках на тестах, представляет собой *компромисс отклонения смещения*.

# MSE декомпозиция (примеры)

$$D = f(x) + \varepsilon$$

$D$  – наблюдения,  $f(.)$  – истинная зависимость,  $\varepsilon$  – шум  $N(0, \sigma)$

• **K-NN:**

$$\hat{D}(x) = \frac{1}{k} \sum_{i \in N_k(x)} D_i, \text{Var}(D) = \sigma^2, \text{Var}(\hat{D}(x)) = \frac{1}{k^2} \sum_{i \in N_k(x)} \text{Var}(D_i) = \frac{\sigma^2}{k},$$

$$\left[ E(\hat{D}(x)) - f(x) \right]^2 = \left[ \frac{1}{k} \sum_{i \in N_k(x)} E(D_i) - f(x) \right]^2,$$

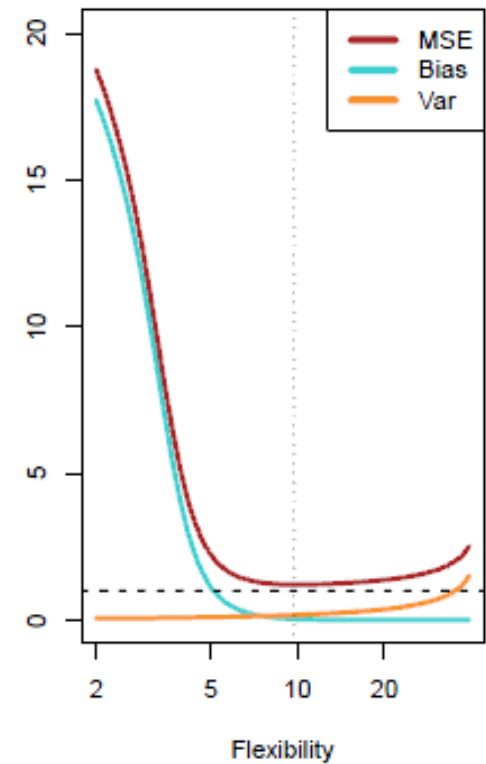
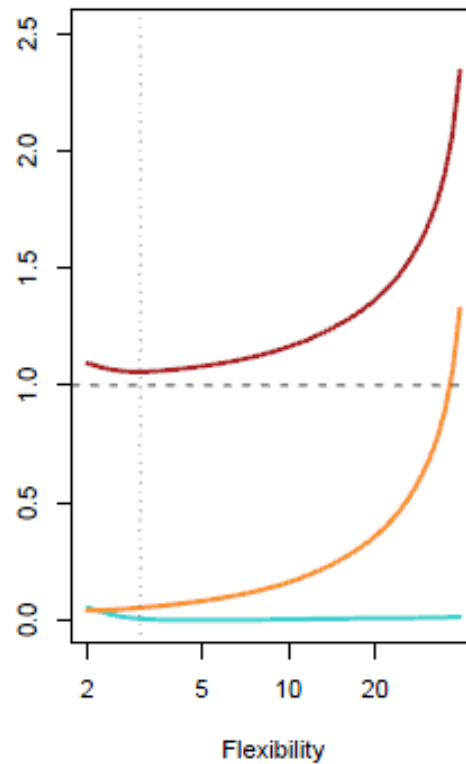
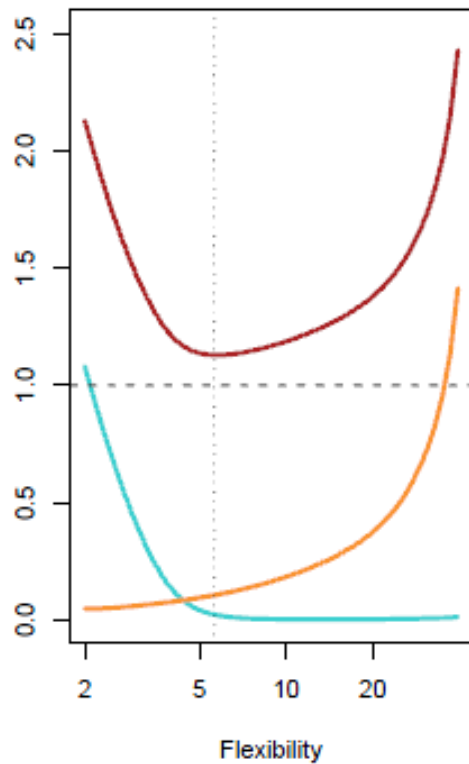
$$MSE = \sigma^2 + \frac{\sigma^2}{k} + \left[ \frac{1}{k} \sum_{i \in N_k(x)} f(x_i) - f(x) \right]^2$$

• **Линейная регрессия:**

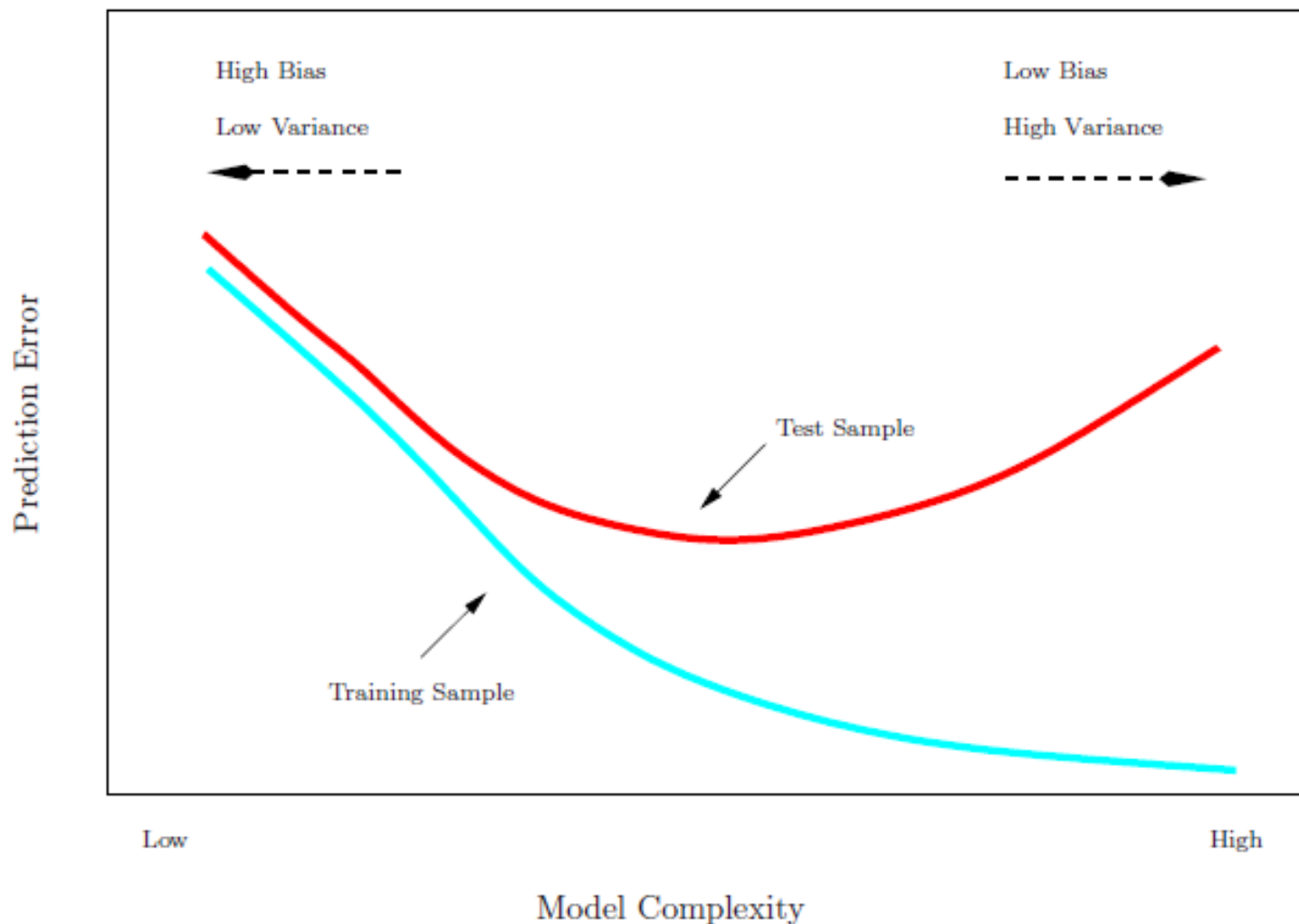
$$\hat{D}(x) = x^T (X^T X)^{-1} X^T \bar{D}, \text{Var}(D) = \sigma^2, \text{Var}(\hat{D}(x)) = \frac{p}{N} \sigma^2,$$

$$MSE = \sigma^2 + \frac{p}{N} \sigma^2 + \frac{1}{N} \sum_x \left[ E[\hat{D}(x)] - f(x) \right]^2$$

# Компромисс отклонения смещения для трех примеров



# Качество на обучающем и тестовом наборе



# Валидация, кросс-валидация и бутстреппинг

- Эти методы позволяют:
  - оценить ошибки прогнозирования тестового набора
  - стандартное отклонение и смещения оценок параметров модели
  - выбрать лучшую модель
- Различия между *ошибкой тестирования* и *ошибкой обучения*:
  - Ошибка тестирования - это усредненная ошибка, которая возникает в результате применения метода статистического обучения для прогнозирования отклика на новом наблюдении, которое не было задействовано в процессе обучения.
  - Ошибка обучения вычисляется после применения метода статистического обучения к наблюдениям, используемым в обучении.

# Применение валидационного набора

- Разделим случайным образом имеющийся набор образцов на две части: обучающую и валидационную выборки.



- Построим модель на обучающем наборе и используем ее для прогнозирования откликов наблюдений в валидационном наборе.
- Полученная ошибка на валидационном множестве дает оценку тестовой ошибки. Ошибка, как правило, оценивается с использованием MSE в случае количественного отклика и ошибки неправильной классификации в случае категориального отклика.

# Использование валидационного набора данных

*Training Data*

	<i>inputs</i>			<i>target</i>

*Validation Data*

	<i>inputs</i>			<i>target</i>

Основные методы генерации валидационного набора как и в sampling:

- Случайная выборка
- Стратифицированная выборка (сохраняем распределение выбранных переменных)
- Кластерная выборка (сохраняем пропорции кластеров)

# Оценка моделей

*Training Data*

	<i>inputs</i>			<i>target</i>

*Validation Data*

	<i>inputs</i>			<i>target</i>



Оценка качества моделей  
на валидационном наборе

Сложность модели      Валидационная оценка



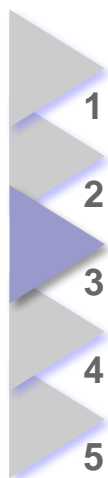
# Выбор модели

*Training Data*

	<i>inputs</i>			<i>target</i>

*Validation Data*

	<i>inputs</i>			<i>target</i>



**Самая простая модель среди  
самых лучших на  
валидационном наборе**

*Сложность модели*      *Валидационная оценка*

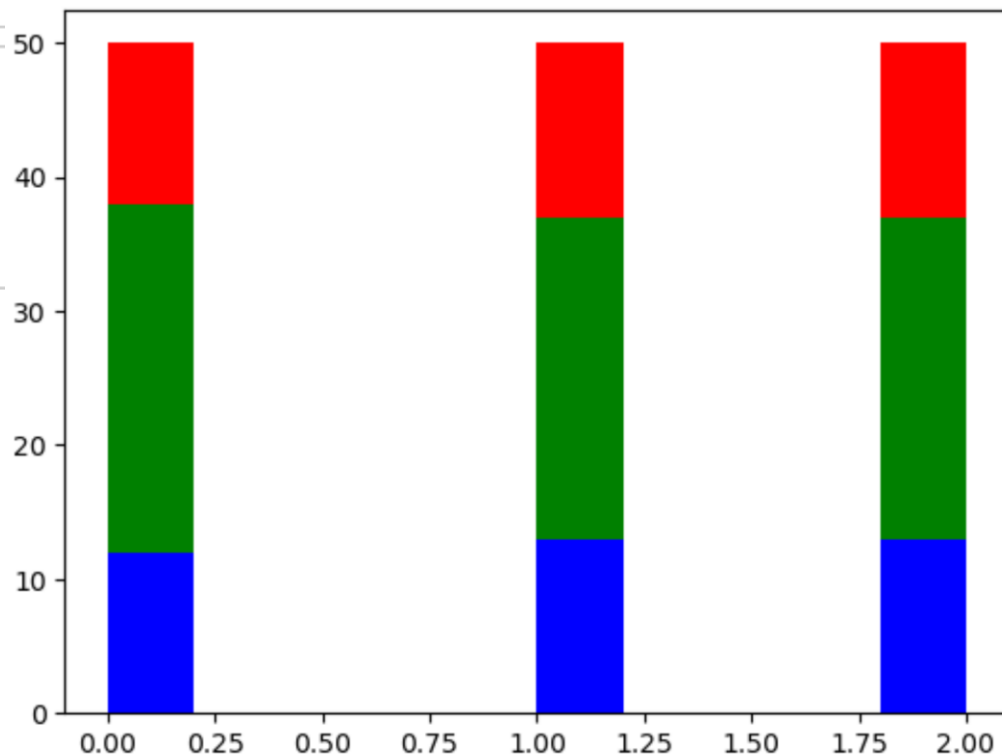
# Пример (Python)

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
```

```
iris = load_iris()
X_train, X_test, y_train, y_test, = train_test_split(iris.data, iris.target, test_size=0.25,
    shuffle=True, # Random select
    stratify=iris.target, # Stratification
    random_state=123)
```

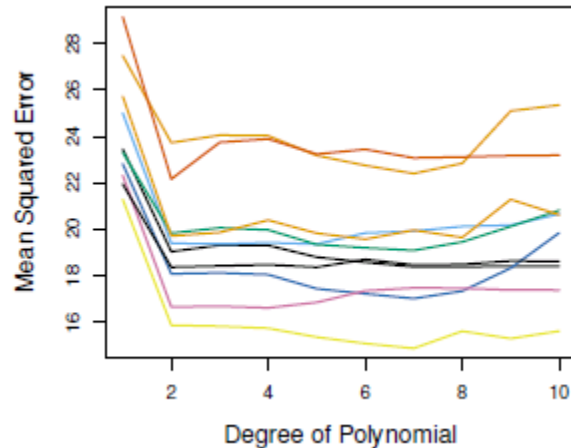
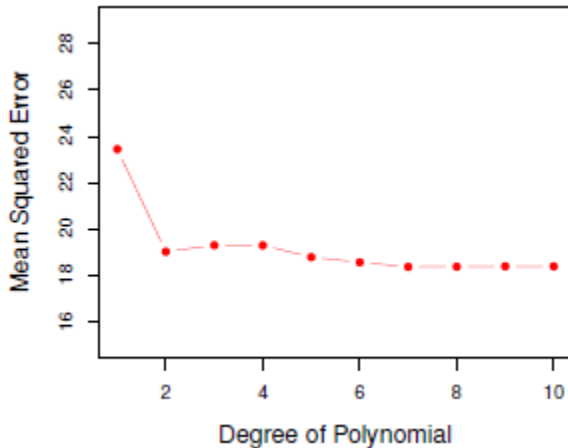
```
plt.hist(iris.target, color="red")
plt.hist(y_train, color="green")
plt.hist(y_test, color="blue")
pass
```

Для кластерной выборки  
передаем в качестве stratify  
метки кластеров



# Пример

- Хотим сравнить регрессионные модели с разными степенями полинома
- Разделим случайным образом 392 наблюдения на две группы: обучающий набор, содержащий 196 объектов и валидационный набор, содержащий оставшиеся 196 объектов.



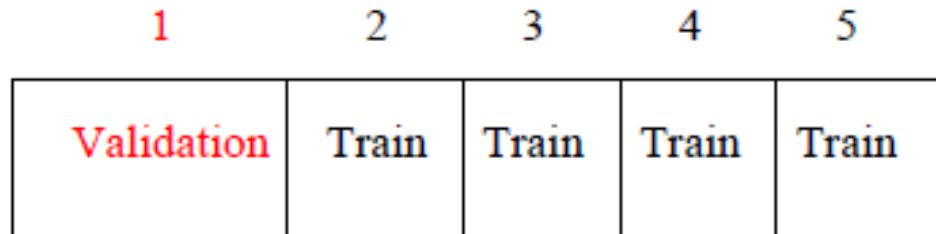
*Слева показано одиночное разбиение, справа - множественное*

# Недостатки подхода применения валидационного набора

- Если плохое разбиение:
  - Валидационная оценка ошибки тестирования может сильно варьироваться в зависимости от того, какие именно наблюдения включены в обучающий набор, а какие в валидационный.
- Не вся информация используется при обучении:
  - При валидационный подходе только подмножество наблюдений (те, которые включены в обучающий набора, а не в валидационный) используются для построения модели.
- Чрезмерный оптимизм:
  - Ошибка на валидационном наборе может иметь тенденцию *переоценивать* ошибку тестирования

# Кросс-валидация

- Широко используемый подход для оценки ошибки тестирования.
- Оценки могут быть использованы для:
  - выбора оптимальной модели,
  - оценки тестовой ошибки результирующей выбранной модели.
- Идея - разделить данные на  $K$  частей равного размера. Мы удаляем часть  $k$ , строим модель на оставшихся частях, а затем получаем прогнозы для удаленной  $k$ -ой части.



- Это делается в свою очередь для каждой части  $k = 1, 2, \dots, K$ , а затем результаты объединяются.

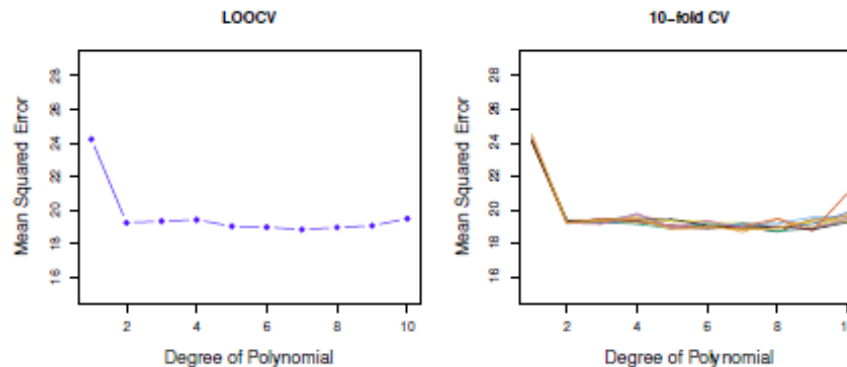
# Кросс-валидация для оценки ошибки

- Обозначим  $K$  частей как  $C_1, C_2, \dots, C_K$ , где  $C_k$  - это индексы наблюдений в части  $k$ . Есть  $n_k$  наблюдения в части  $k$ : если  $N$  кратно  $K$ , то  $n_k = n/K$ .

- Вычислим

$$CV_{(K)} = \sum_{k=1}^K \frac{n_k}{n} \text{MSE}_k$$

где  $\text{MSE}_k = \sum_{i \in C_k} (y_i - \hat{y}_i)^2 / n_k$  и  $\hat{y}_i$  - подгонка для наблюдения  $i$ , полученная на данных с удаленной частью  $k$ .

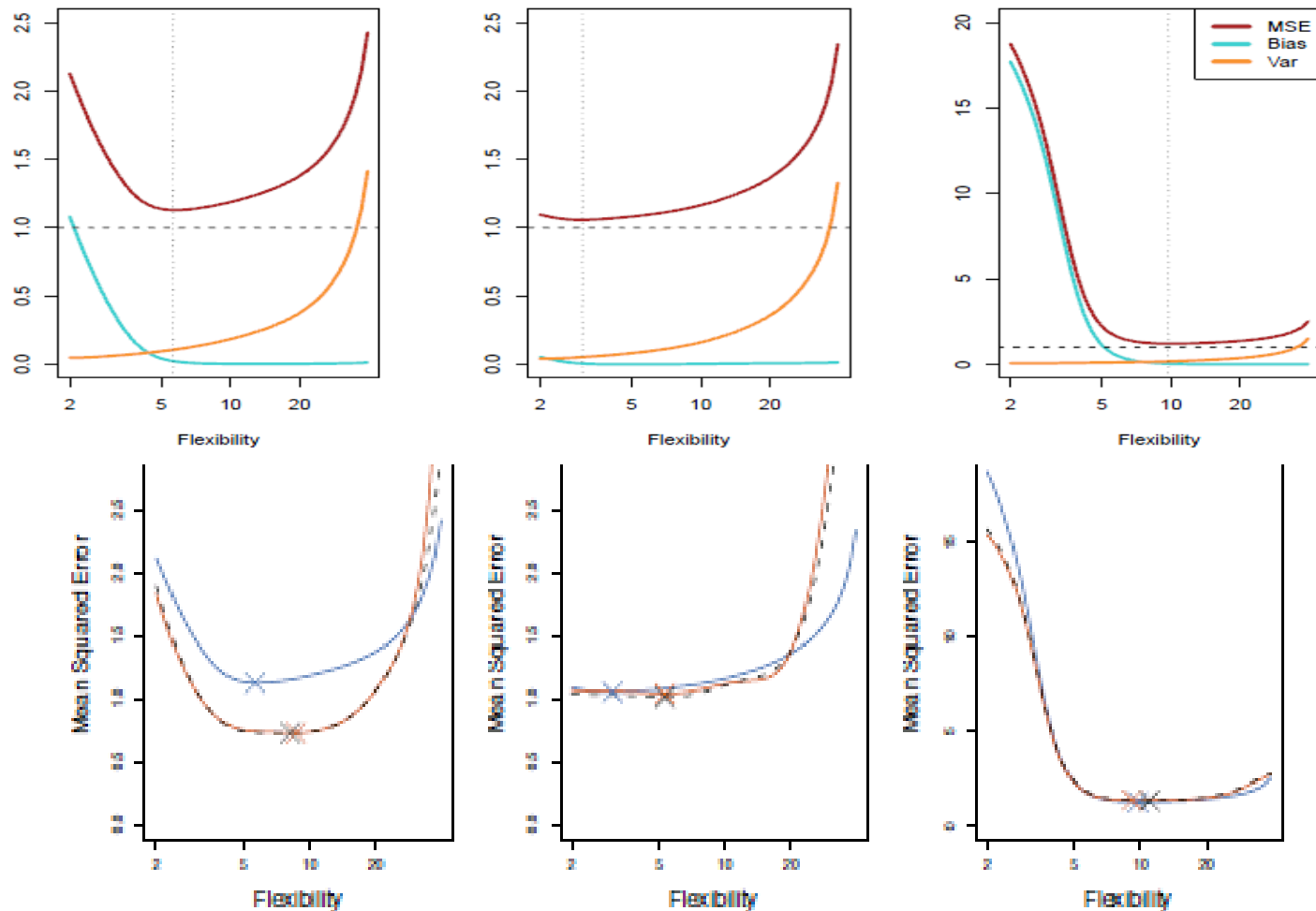


- При  $K = n$  имеем  $n$  папок или кросс-валидацию с попеременным исключением одной из частей (*leave-one out cross-validation*, LOOCV).

# Кросс-валидация для оценки метопараметров и выбора модели

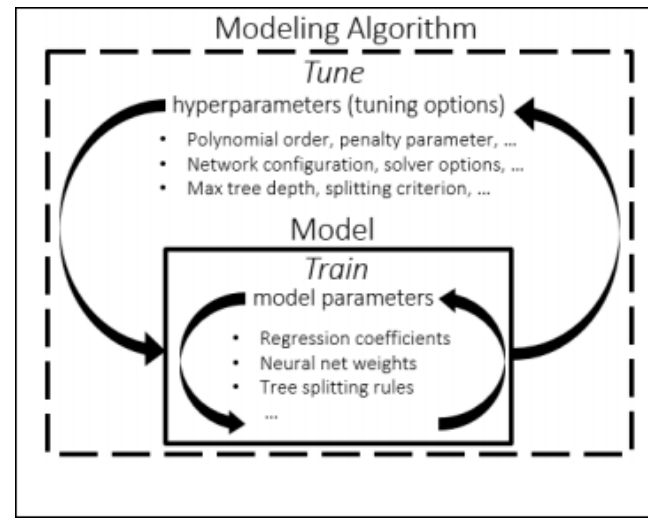
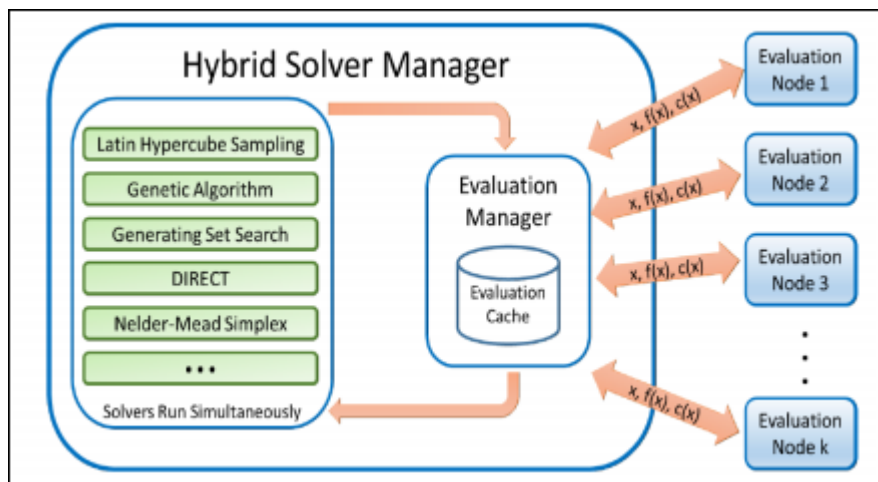
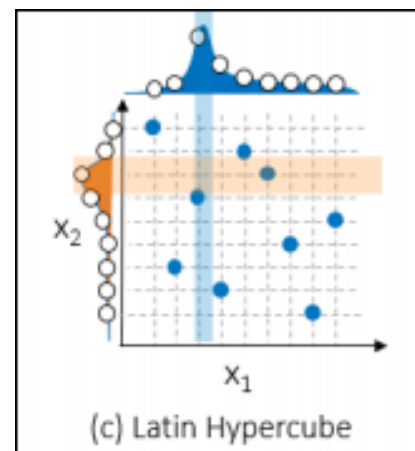
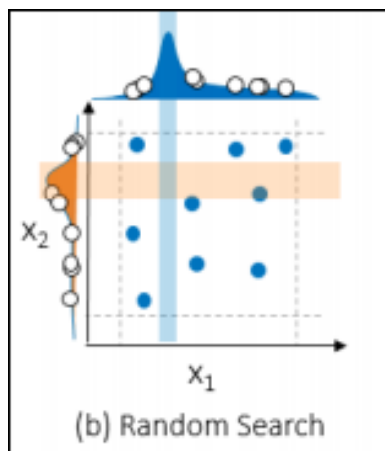
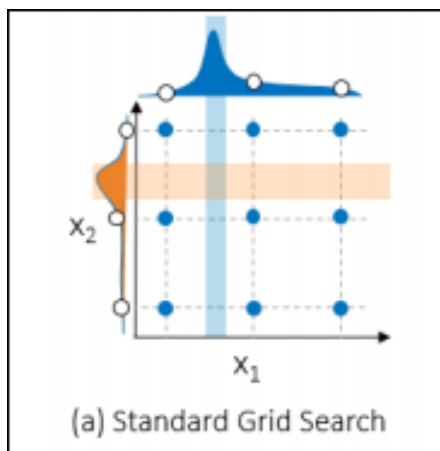
- Зачастую кросс-валидацию используют не для оценки ошибки, а для выбора метопараметров
  - Запускают кросс-валидацию для разных значений метопараметров
  - Рассчитывают кросс-валидационные ошибки для каждого варианта
  - Выбирают лучшее значение метопараметра по кросс-валидационной ошибке
  - Перестраивают модель на всей выборке с этим значением метопараметра

# Кросс-валидация для оценки метопараметров модели





# Кросс-валидация и валидация для автотьюнинга метапараметров



# Пример (Python)

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
```

```
iris = load_iris()
```

```
VT = VarianceThreshold() # Preprocessing
KNN = KNeighborsRegressor() # Regressor
```

```
# Combined model - encapsulates all stages
model = Pipeline([("VT", VT), ("KNN", KNN)])
```

```
# Parameters to cycle through
# Pipeline parameters are passed as <STAGE>__<PARAMETER NAME>
parameters = {"KNN__n_neighbors": range(2, 11),
              "VT__threshold": [0, 0.5]}
```

```
# 5-fold cross-validation
GSCV = GridSearchCV(model, parameters, cv=5)
GSCV.fit(iris.data, iris.target)
pass
```

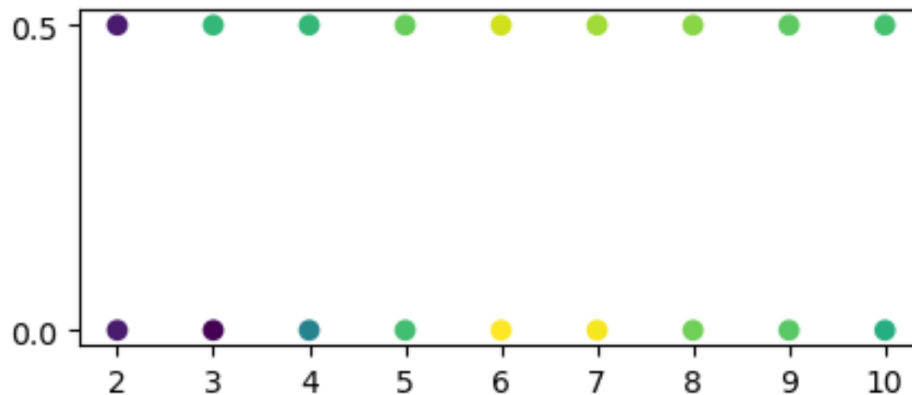
# Пример (Python)

```
GSCV.best_params_
```

```
{'KNN__n_neighbors': 6, 'VT__threshold': 0}
```

```
pred = GSCV.predict(iris.data) # GSCV is equal to the best estimator
```

```
plt.scatter(*pd.DataFrame(GSCV.cv_results_["params"]).T.values, c=GSCV.cv_results_["mean_test_score"])  
plt.yticks(parameters["VT__threshold"])  
plt.gcf().set_size_inches(5, 2)  
pass
```



# Пример – Grid Search (Python)

```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.datasets import fetch_california_housing
```

```
X, y = fetch_california_housing(return_X_y=True)
```

```
N = 5000
X, y = X[:N], y[:N]
X.shape, y.shape
```

```
((5000, 8), (5000,))
```

```
scaler = StandardScaler()
VT = VarianceThreshold() # Preprocessing
KNN = KNeighborsRegressor() # Regressor
```

```
# Combined model - encapsulates all stages
model = Pipeline([("scaler", scaler), ("VT", VT), ("KNN", KNN)])
```

# Пример – Grid Search (Python)

```
# Parameters to cycle through  
# Pipeline parameters are passed as <STAGE>__<PARAMETER NAME>  
parameters = {"KNN__n_neighbors": range(2, 20),  
              "VT__threshold": [0, 1]}
```

```
# 5-fold cross-validation  
GSCV = GridSearchCV(model, parameters, cv=5)  
GSCV.fit(X, y)  
pass
```

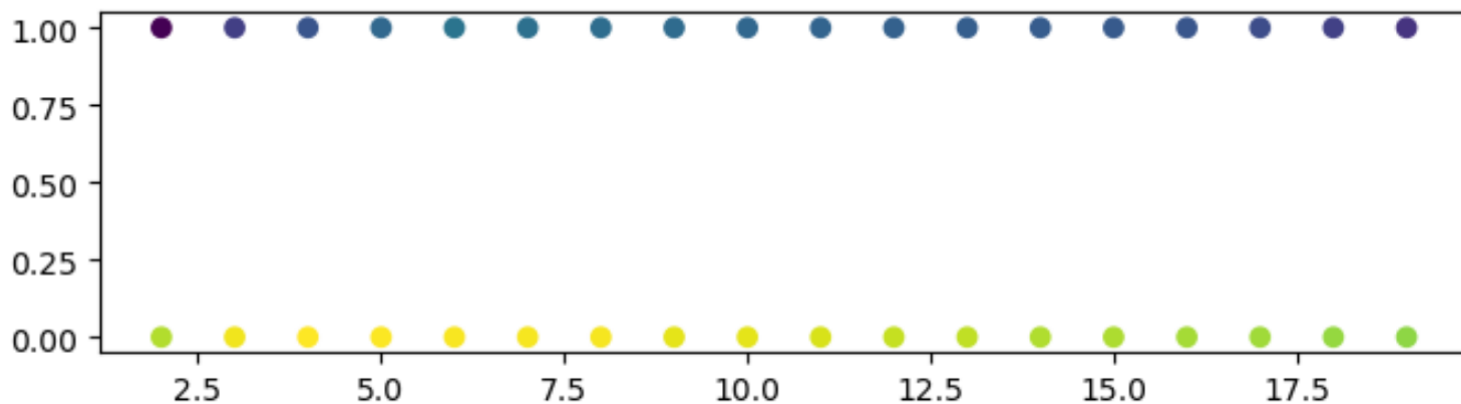
```
GSCV.best_params_
```

```
{'KNN__n_neighbors': 4, 'VT__threshold': 0}
```

```
pred = GSCV.predict(X) # GSCV is equal to the best estimator
```

# Пример – Grid Search (Python)

```
plt.scatter(*pd.DataFrame(GSCV.cv_results_["params"]).T.values, c=GSCV.cv_results_["mean_test_score"])  
plt.gcf().set_size_inches(8, 2)
```



# Пример – Случайный поиск (Python)

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform
```

```
model = Pipeline([("scaler", scaler),
                  ("VT", VarianceThreshold()),
                  ("KNN", KNeighborsRegressor())])
```

```
distributions = {"KNN__n_neighbors": randint(2, 20),
                "VT__threshold": uniform(0, 1)}
```

```
# 5-fold cross-validation
```

```
RSCV = RandomizedSearchCV(model, distributions, n_iter=20,
                          cv=5, random_state=123)
```

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

```
pass
```

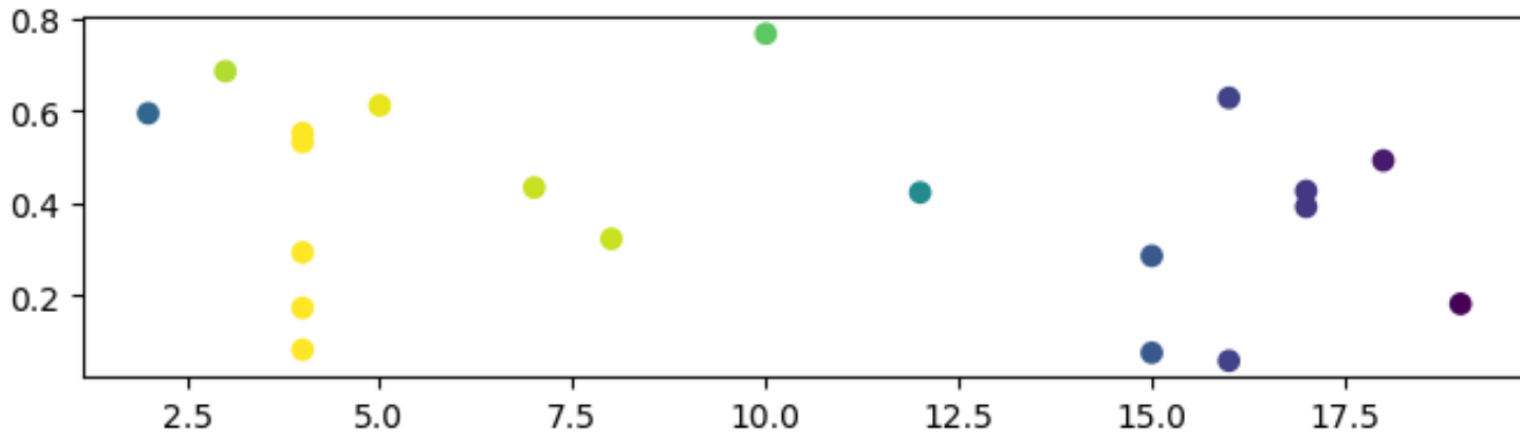
# Пример – Случайный поиск (Python)

```
RSCV.best_params_
```

```
{'KNN__n_neighbors': 4, 'VT__threshold': 0.5513147690828912}
```

```
pred = RSCV.predict(X) # RSCV is equal to the best estimator
```

```
plt.scatter(*pd.DataFrame(RSCV.cv_results_["params"]).T.values, c=RSCV.cv_results_["mean_test_score"])  
plt.gcf().set_size_inches(8, 2)
```





# Пример – Отбор (Python)

```
from sklearn.experimental import enable_halving_search_cv # Required import
from sklearn.model_selection import HalvingGridSearchCV, HalvingRandomSearchCV
```

```
model = Pipeline([("scaler", scaler),
                  ("VT", VarianceThreshold()),
                  ("KNN", KNeighborsRegressor())])
```

```
distributions = {"KNN__n_neighbors": randint(2, 20),
                "VT__threshold": uniform(0, 1)}
```

```
HRSV = HalvingRandomSearchCV(model, distributions, cv=5,
                             factor=2, # Candidate selection cut-off
                             # Resource increasing during selection:
                             resource="n_samples",
                             min_resources=100)
```

```
HRSV.fit(X, y)
pass
```

# Пример – Отбор (Python)

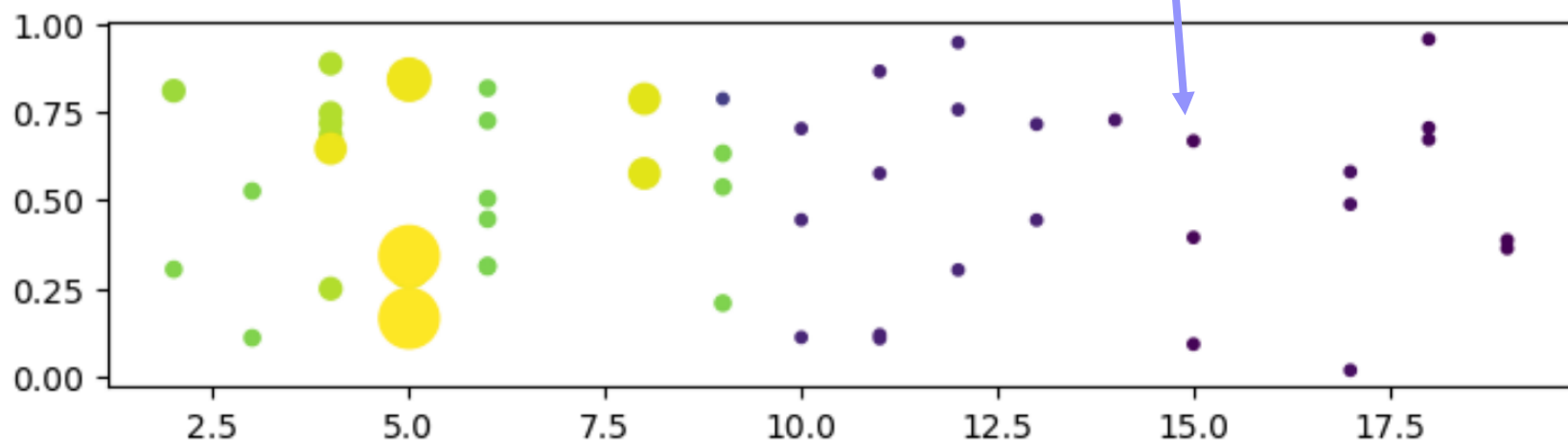
```
HRSV.best_params_
```

```
{'KNN__n_neighbors': 5, 'VT__threshold': 0.3417047325655804}
```

```
pred = HRSV.predict(X) # RSCV is equal to the best estimator
```

```
plt.scatter(*pd.DataFrame(HRSV.cv_results_["params"]).T.values,  
            c=HRSV.cv_results_["mean_test_score"],  
            s=HRSV.cv_results_["n_resources"]/10)  
plt.gcf().set_size_inches(8, 2)
```

Размер отвечает за число семплов



# Бутстреппинг

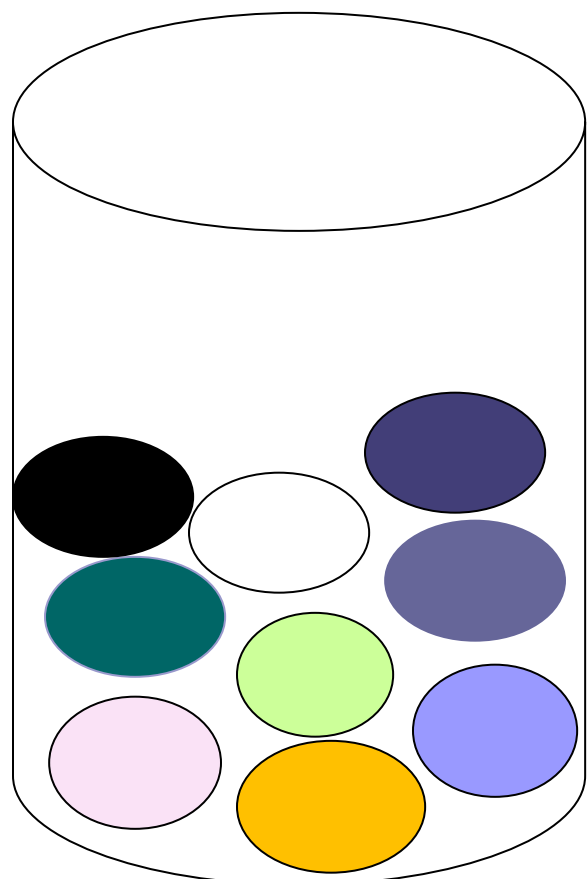
- *Бутстреппинг* представляет собой мощный статистический инструмент, который может быть использован для количественной оценки неопределенности, связанной с данным методом статистического обучения.
- Например, он может позволить произвести оценку стандартной ошибки коэффициента или доверительного интервала для этого коэффициента.
- Использование термина бутстреппинг происходит от фразы, чтобы *to pull oneself up by one's bootstraps*, - цитата из книги «Удивительные приключения барона Мюнхгаузена»

*Барон упал на дно глубокого озера. Когда казалось, что все было потеряно, он решил вытащить себя своими собственными силами.*

# Бутстреппинг

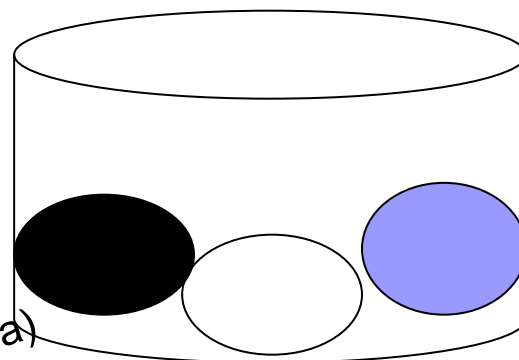
- Подход бутстреппинга позволяет имитировать процесс получения новых случайных наборов данных, так что мы можем оценить дисперсию нашей оценки, не создавая дополнительных образцов.
- Вместо того, чтобы постоянно получать независимые наборы данных, мы получаем различные наборы путем многократной выборки наблюдений из исходного набора *с замещением* (или *с возвращением*).
- Каждый из этих "наборов данных" создается путем выборки *с замещением* и имеет *такой же размер* как наш исходный набор данных. В результате некоторые наблюдения могут появляться более одного раза в наборе данных бутстреппинга, а некоторые нет вообще.

# Случайная выборка с возвратом и без

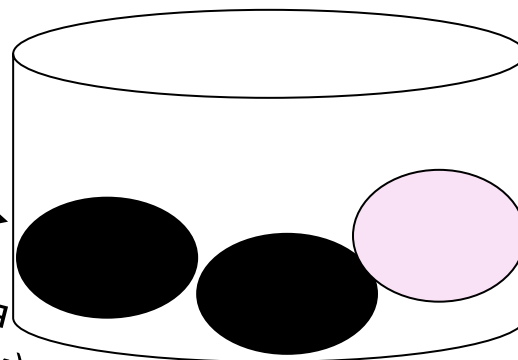


«Сырые» данные

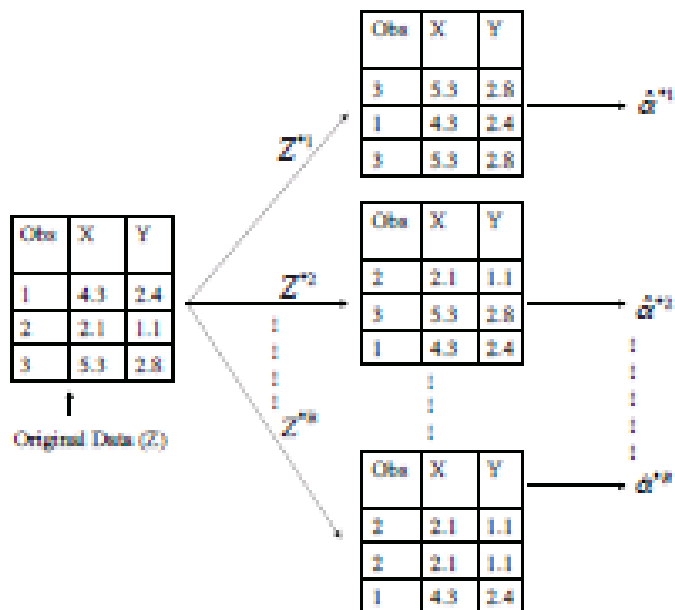
SRSWOR  
(простая случайная  
выборка без возврата)



SRSWR  
(простая случайная  
выборка с возвратом)



# Демонстрационный пример с тремя наблюдениями



- Графическая иллюстрация бутстреппингового подхода на маленькой выборке, содержащей из  $N = 3$  наблюдений.
- Каждый бутстреппинговый набор данных содержит  $n$  наблюдений, отобранных с заменой из исходного набора.
- Каждый такой набор данных начальной используется для получения оценки  $\alpha$

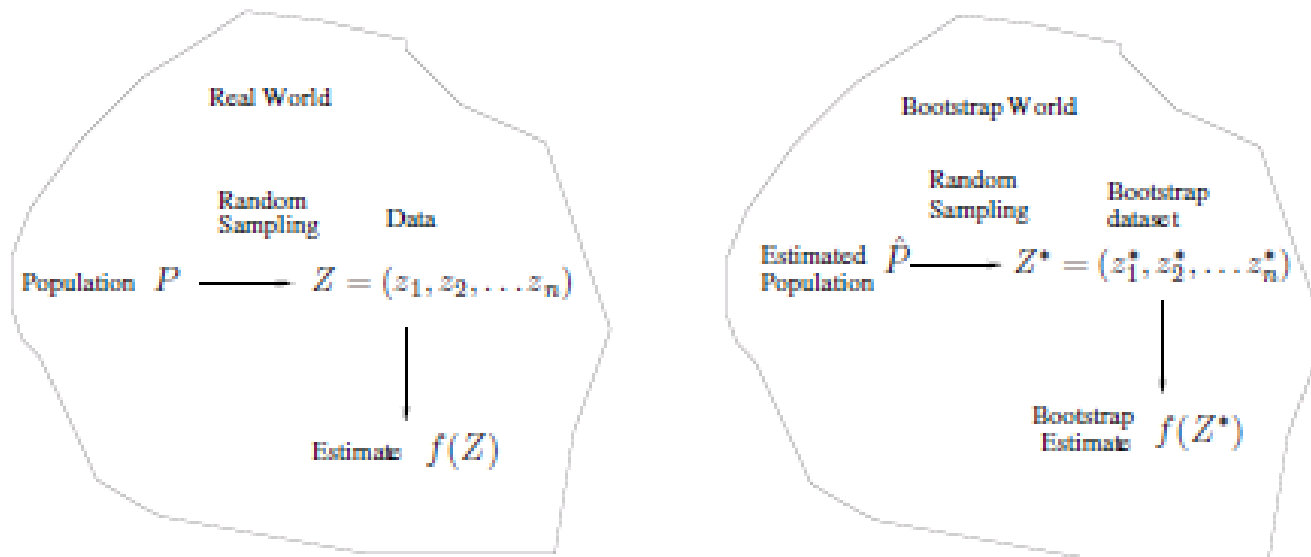
# Бутстрепинг

- Обозначая первый набор данных бутстреппинга как  $Z^{*1}$ , мы используем  $Z^{*1}$ , чтобы выполнить новую оценку для  $\alpha$ , которую обозначим  $\hat{\alpha}^{*1}$
- Эта процедура повторяется  $B$  раз для некоторого большого значения  $B$  (например, 100 или 1000), чтобы получить  $B$  различных наборов данных бутстреппинга  $Z^{*1}, Z^{*2}, \dots, Z^{*B}$ , и  $B$  соответствующих оценок  $\alpha$ :  $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \dots, \hat{\alpha}^{*B}$ .
- Оценим стандартную ошибку этих оценок бутстреппинга, используя формулу:

$$SE_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1} \sum_{r=1}^B (\hat{\alpha}^{*r} - \bar{\hat{\alpha}}^*)^2}.$$

- Она служит в качестве оценки стандартной ошибки, полученной на исходном наборе данных.

# Общая схема бутстрепинга



- В более сложных ситуациях, определение подходящего способа для получения выборок бутстрепинга может потребовать значительных усилий.
- Например, если данные представляют собой временные ряды, мы не можем просто выбирать наблюдения с замещением



# Другие применения бутстрепинга

- В основном используется для получения оценки стандартных ошибок.
- Также обеспечивает приближенные доверительные интервалы для параметра генеральной совокупности.
- Вышеуказанный интервал называется доверительный интервал *бутстреппинга*. Это самый простой способ (среди многих подходов) для получения доверительного интервала бутстрепинга.
- Используется в *багинг ансамблях* моделей (BAG = Bootstrap Aggregation)

# Как бутстрепинг оценивает ошибку прогнозирования

- При кросс-валидации каждая из  $K$  папок валидации отличается от других  $K - 1$  папок, используемых для обучения: *перекрытия нет*. Это очень важно для получения результатов.
- Для оценки ошибки прогнозирования с помощью бутстреппинга мы могли бы подумать об использовании каждого набора данных бутстреппинга в нашей обучающей выборке и исходного набора данных как валидационного набора (или наоборот).
  - Но каждая выборка бутстреппинга имеет значительное перекрытие с исходными данными. Около двух третей исходных точек данных появляются в каждой выборке бутстреппинга.
  - Это приведет бутстреппинг к существенному недооцениванию истинной ошибки прогнозирования
- Удаление перекрытия (*out of bag*)- можно частично решить эту проблему, используя для оценки только те наблюдения, которые не появились (случайно) в текущей выборке бутстреппинга.

# Пример (Python)

```
from sklearn.utils import resample
```

```
X, y = fetch_california_housing(return_X_y=True, as_frame=True)  
X
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25
...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24

20640 rows × 8 columns

# Пример (Python)

```
ITER = 100
SAMPLES = 100
frame = []
for i in range(ITER):
    sample = resample(X, replace=True, n_samples=SAMPLES, stratify=None)
    stat = sample["HouseAge"].mean()
    frame.append(stat)
frame = np.array(frame).flatten()
frame = pd.Series(frame).sort_values()
```

Доверительные интервалы 90% для среднего возраста жилища:

```
frame.quantile(0.05), frame.quantile(0.95)
```

(26.248, 30.6515)

# Бутстреп-регрессия (Python)

```
from sklearn.ensemble import BaggingRegressor
```

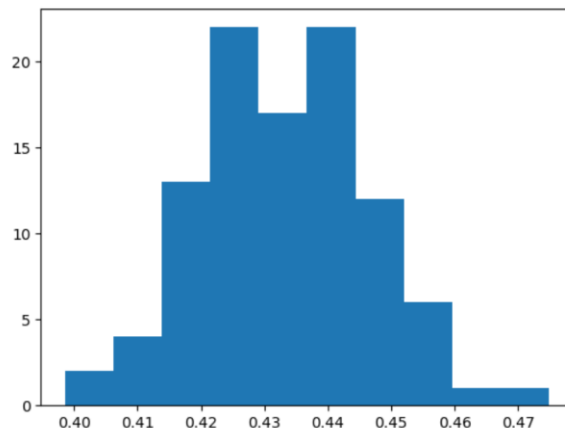
```
estimator = BaggingRegressor(LinearRegression(), n_estimators=100,  
                             bootstrap=True, max_samples=0.1,  
                             random_state=42)
```

```
features = X[["MedInc", "HouseAge"]]
```

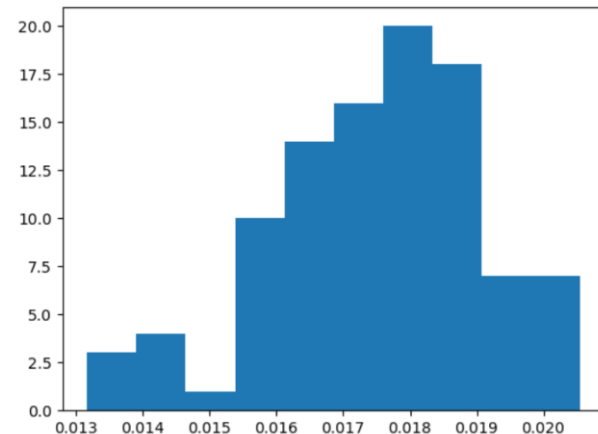
```
estimator.fit(features, y)  
pass
```

```
coefs = np.array([x.coef_ for x in estimator.estimators_])
```

```
plt.hist(coefs[:, 0])  
pass
```



```
plt.hist(coefs[:, 1])  
pass
```



# Бутстреп-регрессия (Python)

```
pred = np.array([x.predict(features.values) for x in estimator.estimators_]).T  
pred.shape
```

```
(20640, 100)
```

```
plt.plot(np.percentile(pred, q=5, axis=1)[:25], c="blue")  
plt.plot(np.percentile(pred, q=95, axis=1)[:25], c="red")  
plt.scatter(range(25), estimator.predict(features)[:25], c="green")  
pass
```

