

Лекция 3: метрический подход к прогнозированию, проклятие размерности, переобучение

Что прогнозирует модель?

- Если есть несколько наблюдений с одинаковым вектором признаков x_* , но разными откликами $y_{*1}, y_{*2}, \dots, y_{*k}$, то какой прогноз минимизирует эмпирический риск (ошибку)? Ответ:

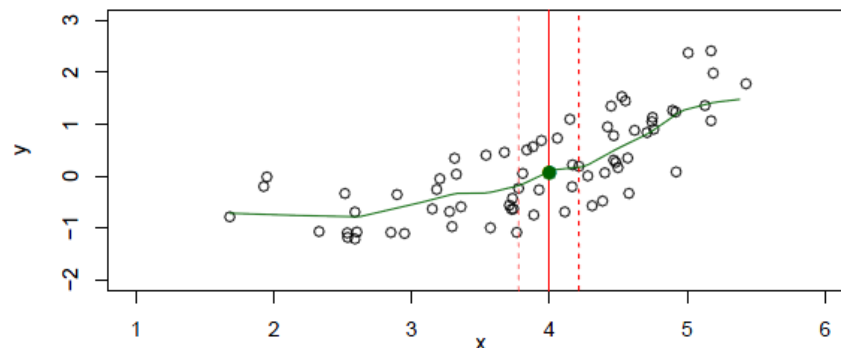
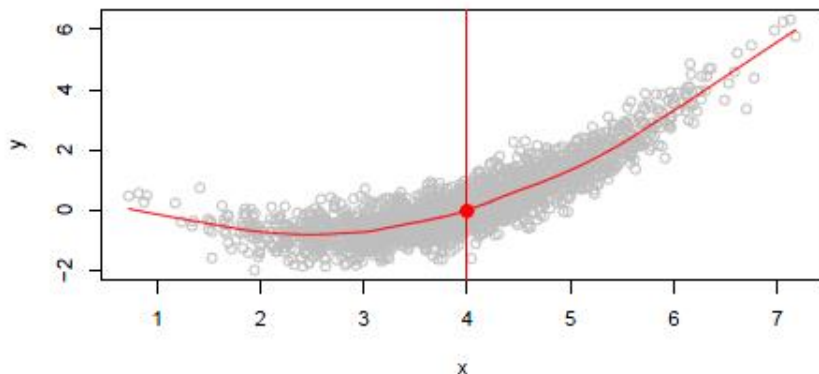
$$y_* = \arg \min_y Q(y, \{y_{*1}, y_{*2}, \dots, y_{*k}\}) = \arg \min_y \frac{1}{k} \sum_{*i} L(y_{*i}, y),$$

например, для квадратичной функции потерь: $y_* = E(y|x = x_*)$

- Если повторяющихся наблюдений нет, то можно «приблизить»:

$$y_* = E(y|x = x_*) \approx \text{mean}[y|x \in N(x_*)], \text{ где } N(x_*) - \text{«окрестность» } x_*$$

- Но должна выполняться гипотеза о компактности (или о непрерывности) и может быть проблема проклятия размерности



Обобщенный метрический классификатор

- Для выбранного x_* , ранжируем выборку $\{x^{(1)}, x^{(2)}, \dots, x^{(l-1)}\}$, так чтобы:

$$d(x_*, x^{(1)}) \leq d(x_*, x^{(2)}) \leq \dots \leq d(x_*, x^{(l-1)})$$

$x^{(i)}$ - i -й сосед x_* , а $y^{(i)}$ - отклик i -го соседа

- Метрический алгоритм классификации:

$$a(x, Z) = \arg \max_{y \in Y} \Gamma_y(x)$$

$\Gamma_y(x) = \sum_{i=1}^l [y = y^{(i)}] w(i, x)$ - оценка близости объекта x к классу y ,
 $w(i, x)$ – важность i -го соседа, не возрастает по i , неотрицательна

- Методы:

- «ближайшего соседа» $w(i, x) = [i \leq 1]$
- «к ближайших соседей» $w(i, x) = [i \leq k]$

Простые методы К ближайших соседей

■ Общая схема работы:

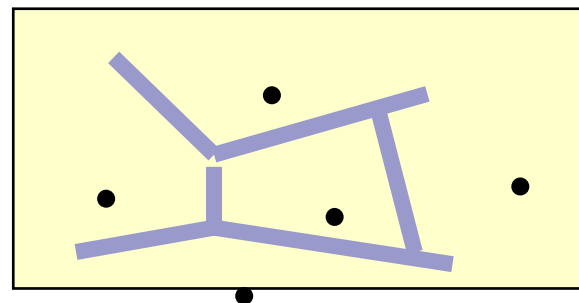
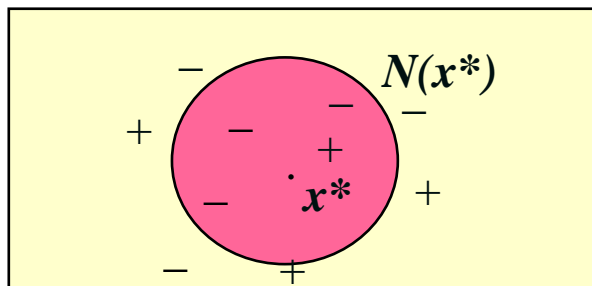
- Каждый пример – точка в пространстве, все примеры хранятся
- Вводится метрика расстояния с учетом нормирования координат
- Ищется К (от 1 до ...) ближайших соседей
- Прогноз вычисляется как агрегирующая функция от откликов найденных соседей

■ 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

- На базе KNN, но:
 - помимо распределения «отклика» учитываются и расстояния до соседей $w(i, x) = [i \leq k]w_i$, где w_i зависит от близости до x
- Примеры весов:
 - линейный вес $w_i = \frac{k+1-i}{k}$, экспоненциальный $w_i = q^i, 0 < q < 1$
 - ядра, например, $w_i = \exp(-d_i^2/h)$
- Усреднение отклика за счет:
 - «взвешенного» голосования для классификации и «взвешенного» среднего для регрессии (далее покажем откуда это следует)

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

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

Метод К ближайших соседей с дискриминантным адаптивным расстоянием

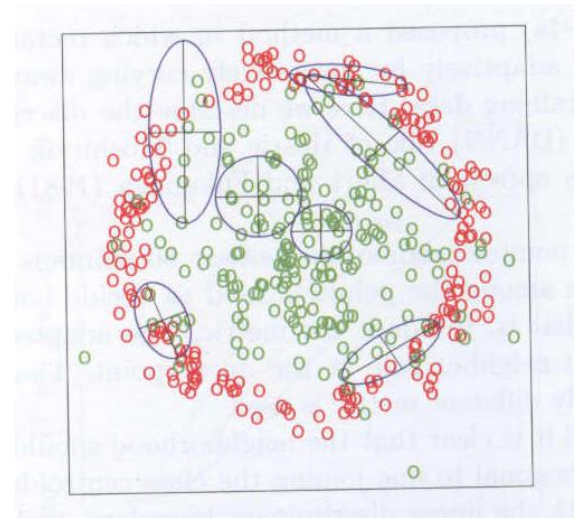
■ Метод DANN:

- На базе KNN, но используется итеративный (m-номер итерации) расчет локального расстояния Махаланобиса с учетом структуры распределения соседей в окрестности:

$$d^{(m)}(x^*, x_i) = (x^* - x_i)^T \Sigma_{(m)}^{-1} (x^* - x_i)$$

■ Параметры алгоритма:

- K_M – число соседей для оценки метрики (нужно побольше)
- K – число соседей для прогноза (лучше поменьше)



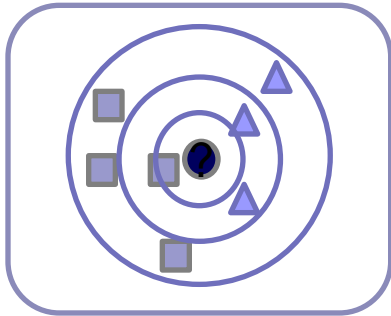
Выбор параметра K

Важность K:

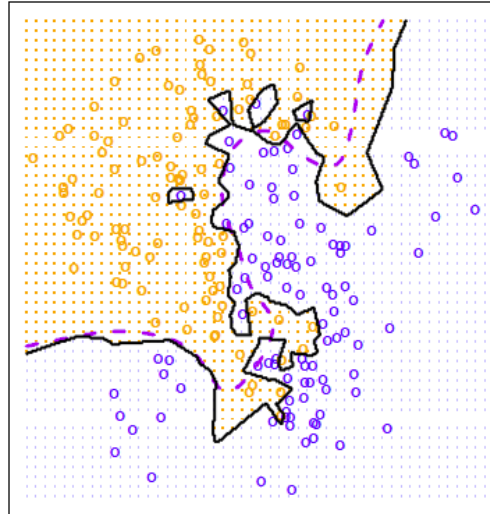
$k = 1$: Результат = квадрат

$k = 5$: Результат = треугольник

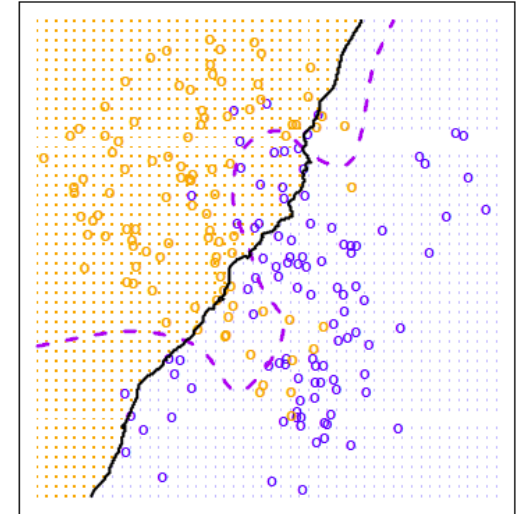
$k = 7$: Снова квадрат



KNN: K=1



KNN: K=100



■ Выбор k :

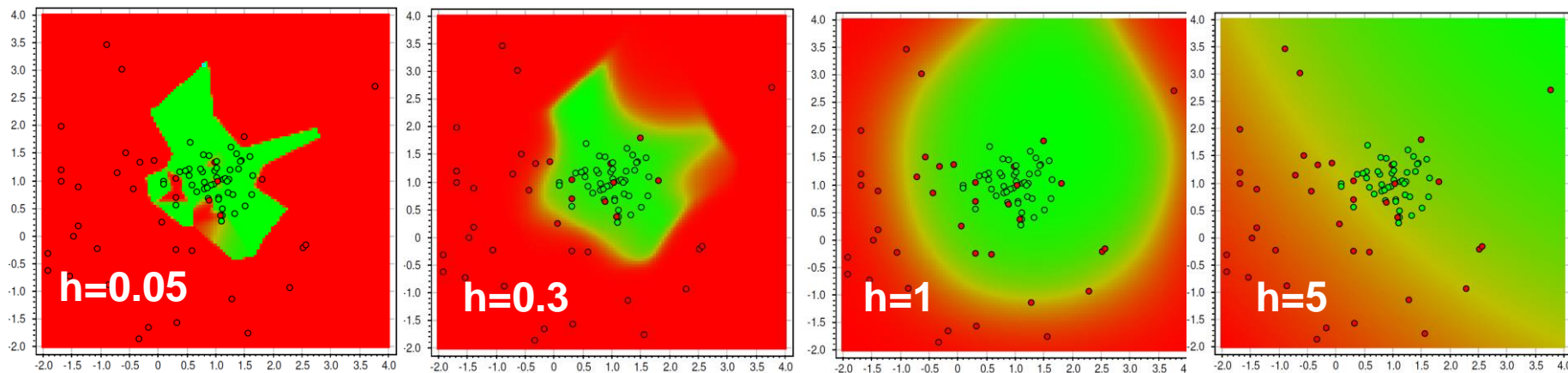
- Если k мал, то чувствительность к шуму, и негладкие границы классов (или линии уровня для регрессий)
- Если k велико, то окрестность может сильно «задеть» соседний класс, зато гладкие границы
- При классификации следует нечетный k , чтобы не было «ничьей»
- Выбирается кросс-валидацией или на валидационном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

Метод окна Парзена (для классификации)

- Вес задается радиально-базисным ядром (h – «ширина ядра»):

$$w(i, x) = K \left(\frac{d(x, x^{(i)})}{h} \right)$$

- h – фиксировано $a(x, Z, h) = \arg \max_{y \in Y} \sum_{i=1}^l [y = y^{(i)}] K \left(\frac{d(x, x^{(i)})}{h} \right)$
- k – фиксировано $a(x, Z, k) = \arg \max_{y \in Y} \sum_{i=1}^l [y = y^{(i)}] K \left(\frac{d(x, x^{(i)})}{d(x, x^{(k)})} \right)$



Параметрические модели

- Параметрическое семейство функций

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

Θ – множество допустимых параметров

- Линейная модель:

- Классификация $g(x, \theta) = \text{sign}[\sum_{i=1}^q \theta_i f_i(x)], Y = \{-1, 1\}$

- Регрессия $g(x, \theta) = \sum_{i=1}^q \theta_i f_i(x), Y = \mathbb{R}$

- θ -вектор параметров

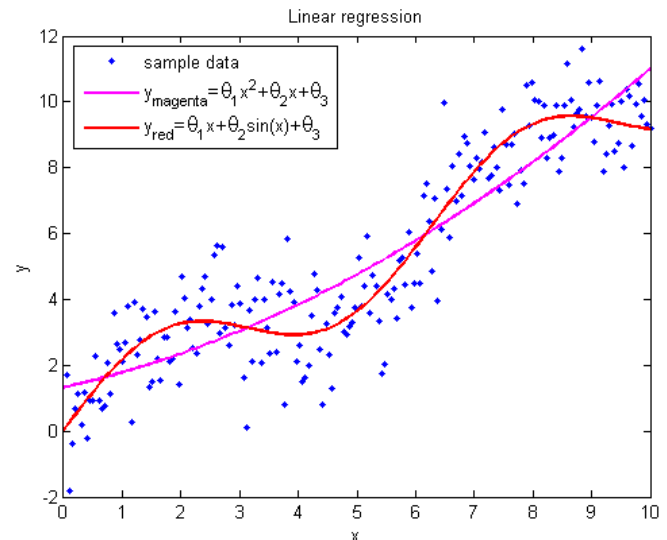
- f_i -не обязательно линейная

- если f_i -зависит от одного признака,

- то модель аддитивная

- Пример:

- Признаки $\{1, x, x^2\}$ vs $\{1, x, \sin(x)\}$



Метод наименьших квадратов для линейной регрессии

- Оценка ошибки - квадратичная функция потерь:

$$Q(\theta, Z) = \frac{1}{l} \sum_{i=1}^l (y_i - a(\bar{x}_i, \theta))^2 = \frac{1}{l} \sum_{i=1}^l \left(y_i - \theta_0 - \sum_{j=1}^p x_{ij} \theta_j \right)^2$$

- В матричной форме:

$$Q(\theta, Z) = (\bar{y} - X\theta)^T (\bar{y} - X\theta)$$

- Единственное оптимальное решение (если матрица данных не сингулярная)

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

- Но не все так просто:

- Если сингулярная матрица данных из-за коррелированных факторов
- Большое число регрессоров – плохая точность и интерпретируемость

Непараметрическая (ядерная) регрессия Надарая-Ватсона

- Суть метода:

- МНК (квадратичная функция потерь)
- Локальный прогноз константой в точке
- Ядерные веса для определения локальной окрестности

- Постановка задачи:

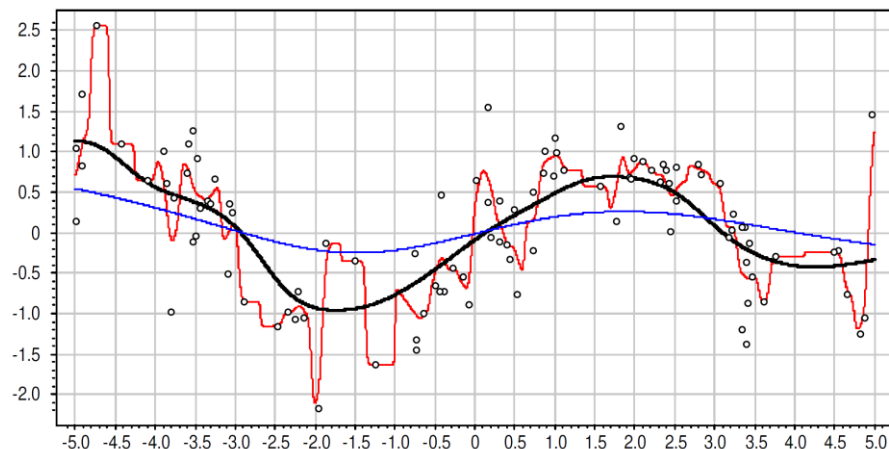
$$Q(Z, \theta) = \frac{1}{l} \sum_{i=1}^l K\left(\frac{d(x, x^{(i)})}{h}\right) (y_i - \theta)^2 \rightarrow \min_{\theta \in \mathbb{R}}$$

- Решающая функция:

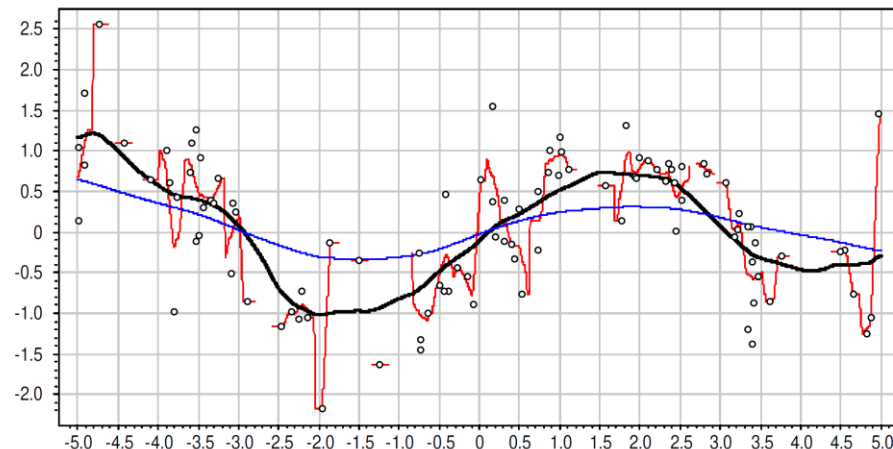
$$a(x, Z, h) = \theta = \frac{\sum_{i=1}^l y_i K\left(\frac{d(x, x^{(i)})}{h}\right)}{\sum_{i=1}^l K\left(\frac{d(x, x^{(i)})}{h}\right)}$$

Непараметрическая (ядерная) регрессия Надарая-Ватсона

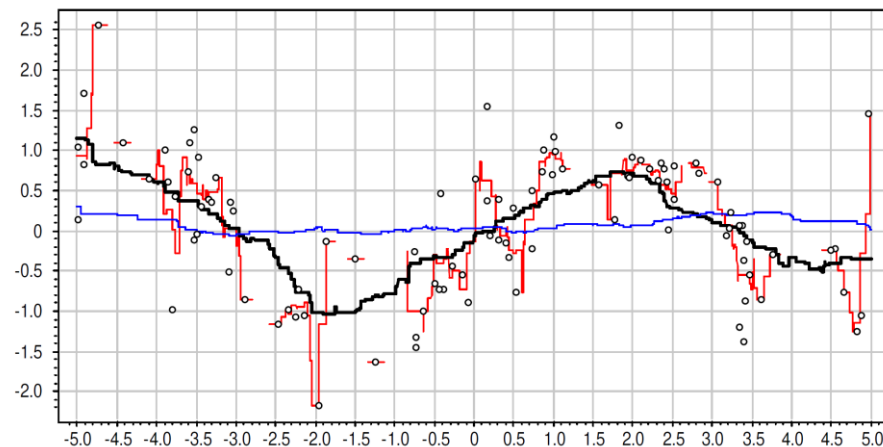
$h \in \{0.1, 1.0, 3.0\}$, гауссовское ядро $K(r) = \exp(-2r^2)$



$h \in \{0.1, 1.0, 3.0\}$, треугольное ядро $K(r) = (1 - |r|)[|r| \leq 1]$



$h \in \{0.1, 1.0, 3.0\}$, прямоугольное ядро $K(r) = [|r| \leq 1]$

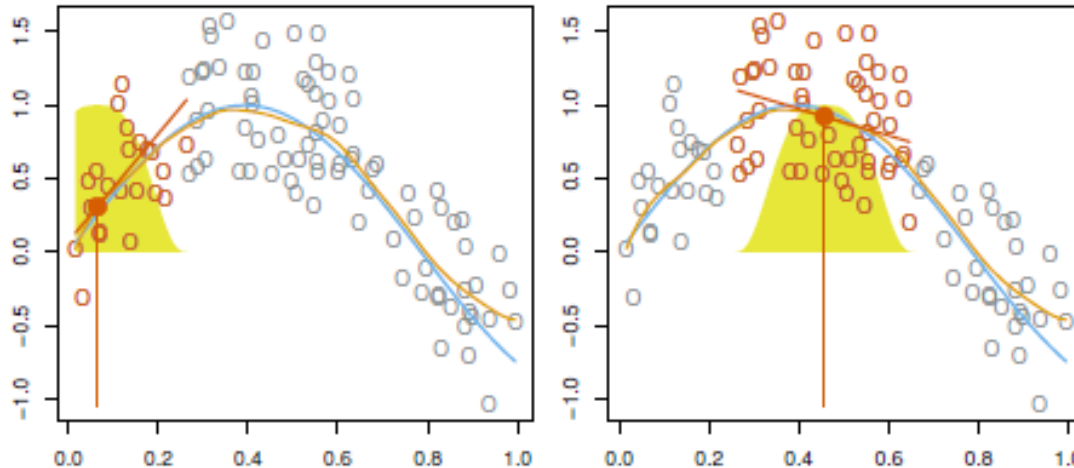


- Чем больше h тем «проще» функция
- Гладкость определяется ядром (непрерывная/кусочно линейная, кусочно постоянная аппроксимация)
- Разрыв, если нет точек в окрестности

Локальная взвешенная регрессия

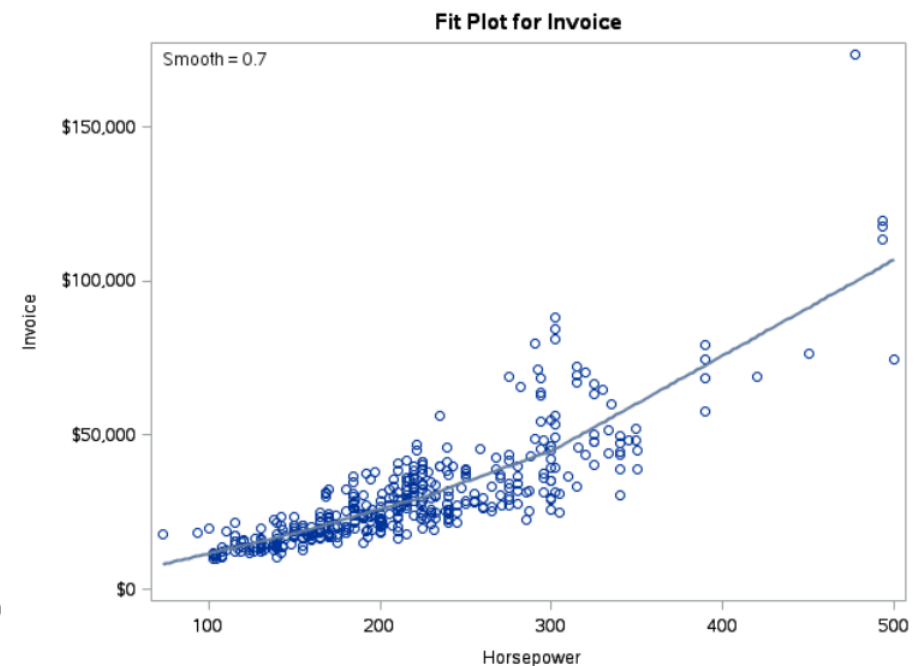
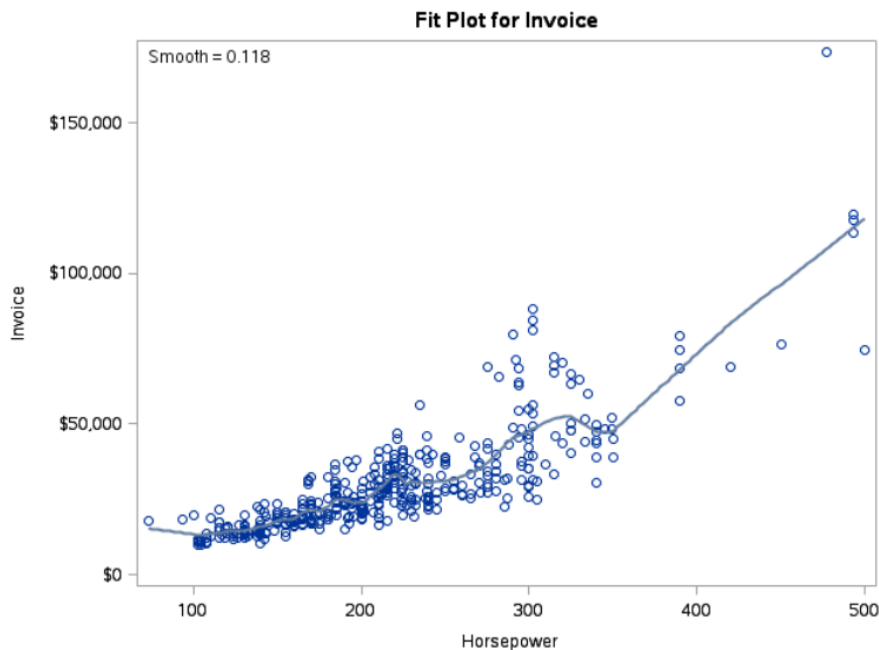
- Вместо константы (как в kernel regression) простая локальная параметрическая модель, например, линейная:

$$Q(Z, \theta) = \frac{1}{l} \sum_{i=1}^l K\left(\frac{d(x, x^{(i)})}{h}\right) (y_i - \mathbf{x}^T \boldsymbol{\theta})^2 \rightarrow \min_{\boldsymbol{\theta} \in \mathbb{R}^p}$$



Локальная взвешенная регрессия

- Нужно задавать параметр сглаживания (фактически – штраф за сложность), который определяет число точек окрестности, чтобы не усложнять модель:



Пример (Python)

Data Set Characteristics:

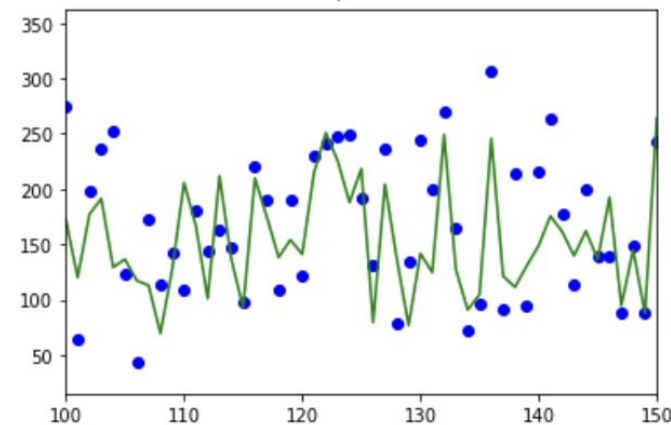
Number of Instances:	442
Number of Attributes:	First 10 columns are numeric predictive values
Target:	Column 11 is a quantitative measure of disease progression one year after baseline
Attribute Information:	<ul style="list-style-type: none">• age age in years• sex• bmi body mass index• bp average blood pressure• s1 tc, total serum cholesterol• s2 ldl, low-density lipoproteins• s3 hdl, high-density lipoproteins• s4 tch, total cholesterol / HDL• s5 ltg, possibly log of serum triglycerides level• s6 glu, blood sugar level

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

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

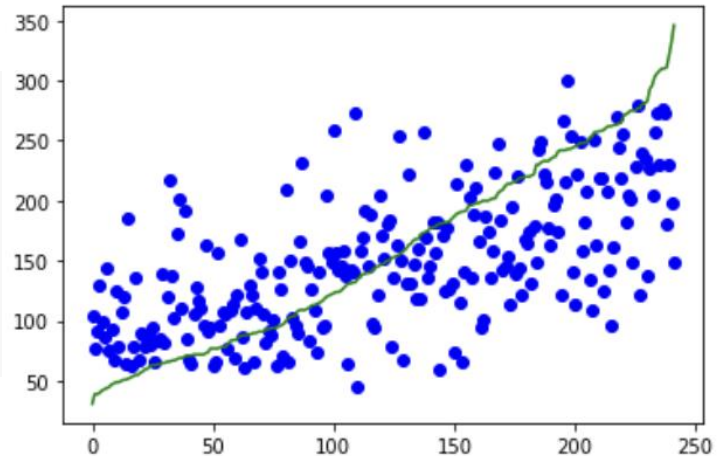
```
N = 200  
data = load_diabetes()  
X, X_test = data.data[:N], data.data[N:]  
y, y_test = data.target[:N], data.target[N:]
```

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

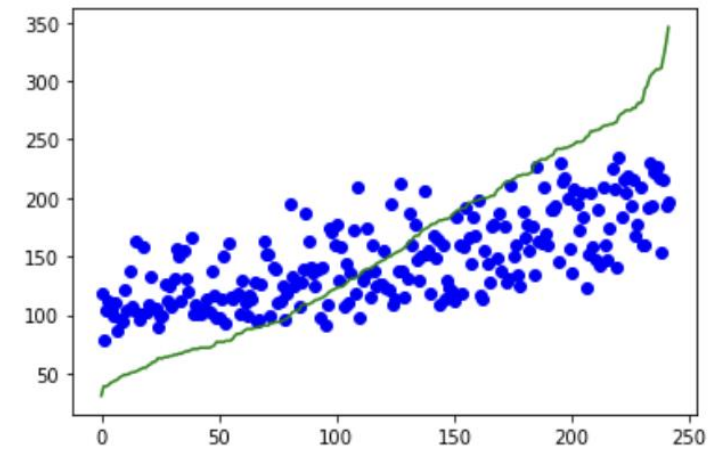


Пример (Python)

```
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")
```



Свойства метрических методов

■ Основные свойства:

- ☐ «Ленивая модель» - не надо ничего обучать
- ☐ Обязательно нужна хорошая метрика и значимые признаки
- ☐ Есть критические **метапараметры**, определяющие сложность модели (гладкость границы или изолиний)

■ Достоинства:

- ☐ Простота реализации
- ☐ Один из самых точных методов (при корректной настройке)
- ☐ Легко адаптируется под сложные типы «откликов», включая ранжирование, многотемность и т.д.
- ☐ Можно интегрировать экспертные знания, задавая веса у примеров, или параметры у метрики

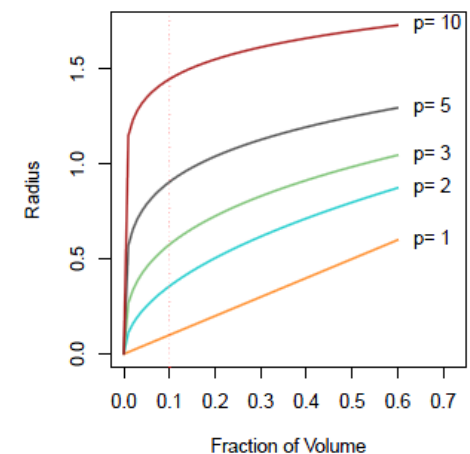
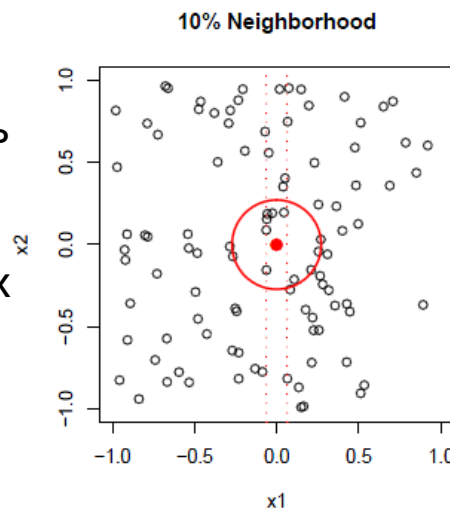
■ Недостатки:

- ☐ «черный ящик» - результат не интерпретируемый совсем
- ☐ Достаточно вычислительно трудоемкие
- ☐ **«Проклятие размерности»**

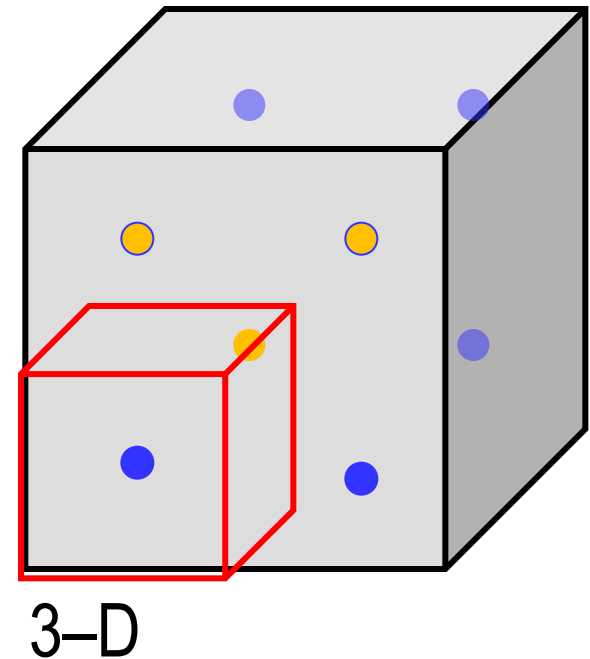
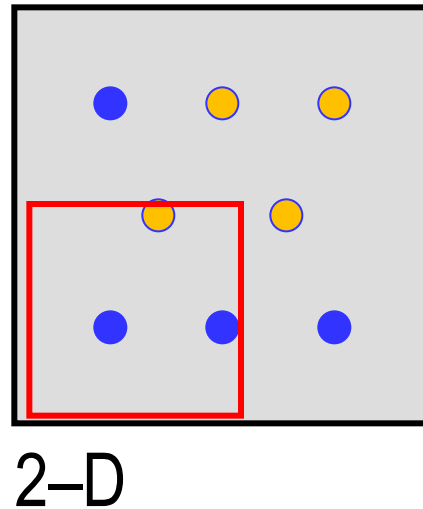
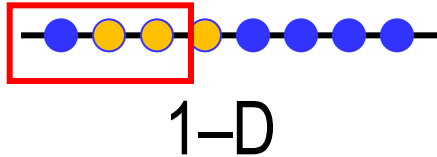
Проклятие размерности

- Суть проблемы: экспоненциальный рост числа необходимых наблюдений при линейном росте размерности пространства
- Пример: ближайшие соседи как правило расположены далеко при больших размерностях пространства признаков.

Например, нам нужно получить значительную часть выборки, чтобы сгладить границу и снизить случайность в усредненном прогнозе, пусть 10%. 10% соседей для случая больших размерностей не может быть локализована, так что мы уже не можем оценить отклик на основе локального усреднения.



Модельный пример, демонстрирующий проклятие размерности

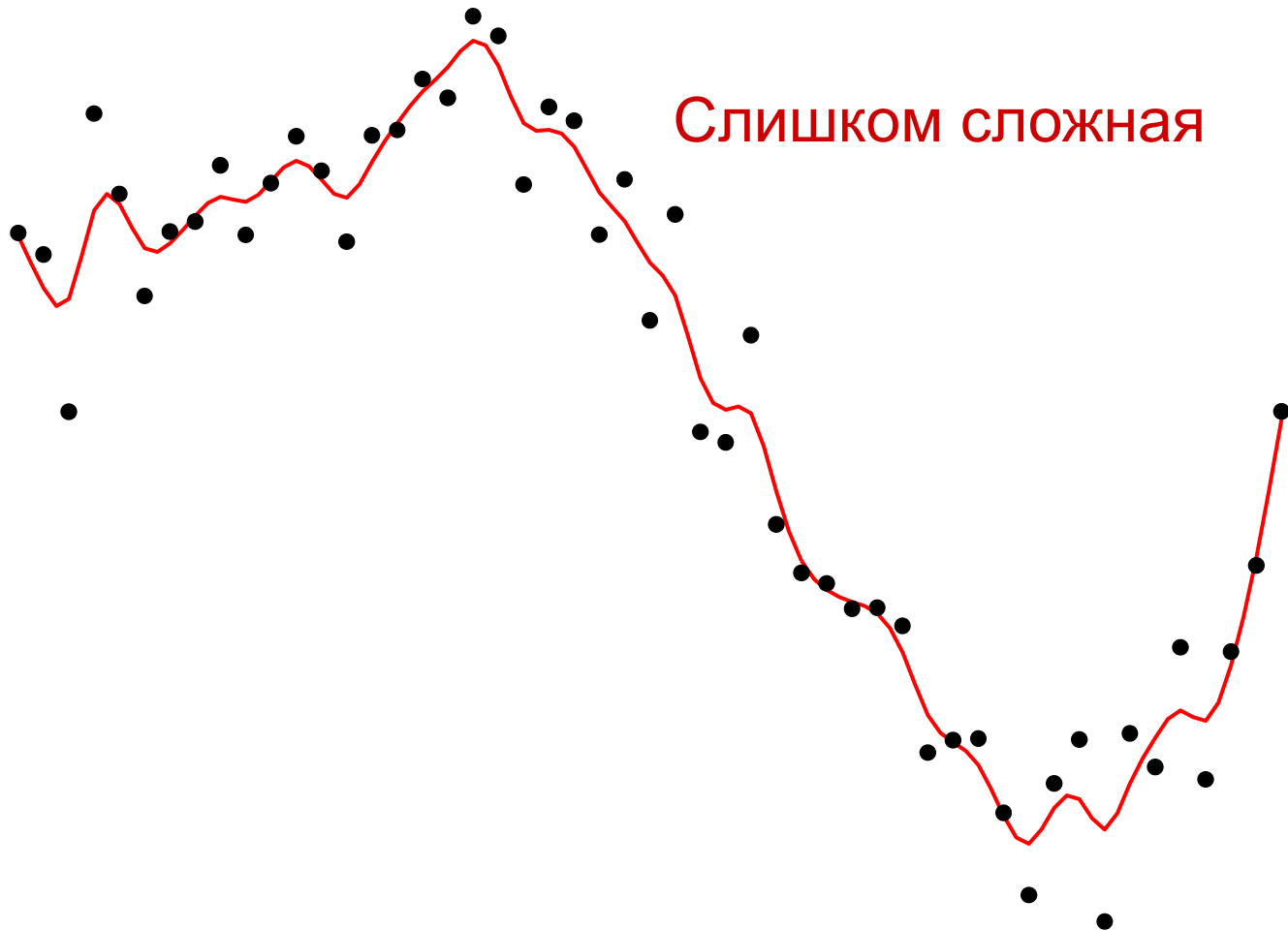


- $r=K/N$
- $E_p(r)=r^{1/p}$
- $E_{10}(0.01)=0.63$
- $E_{10}(0.1)=0.8$

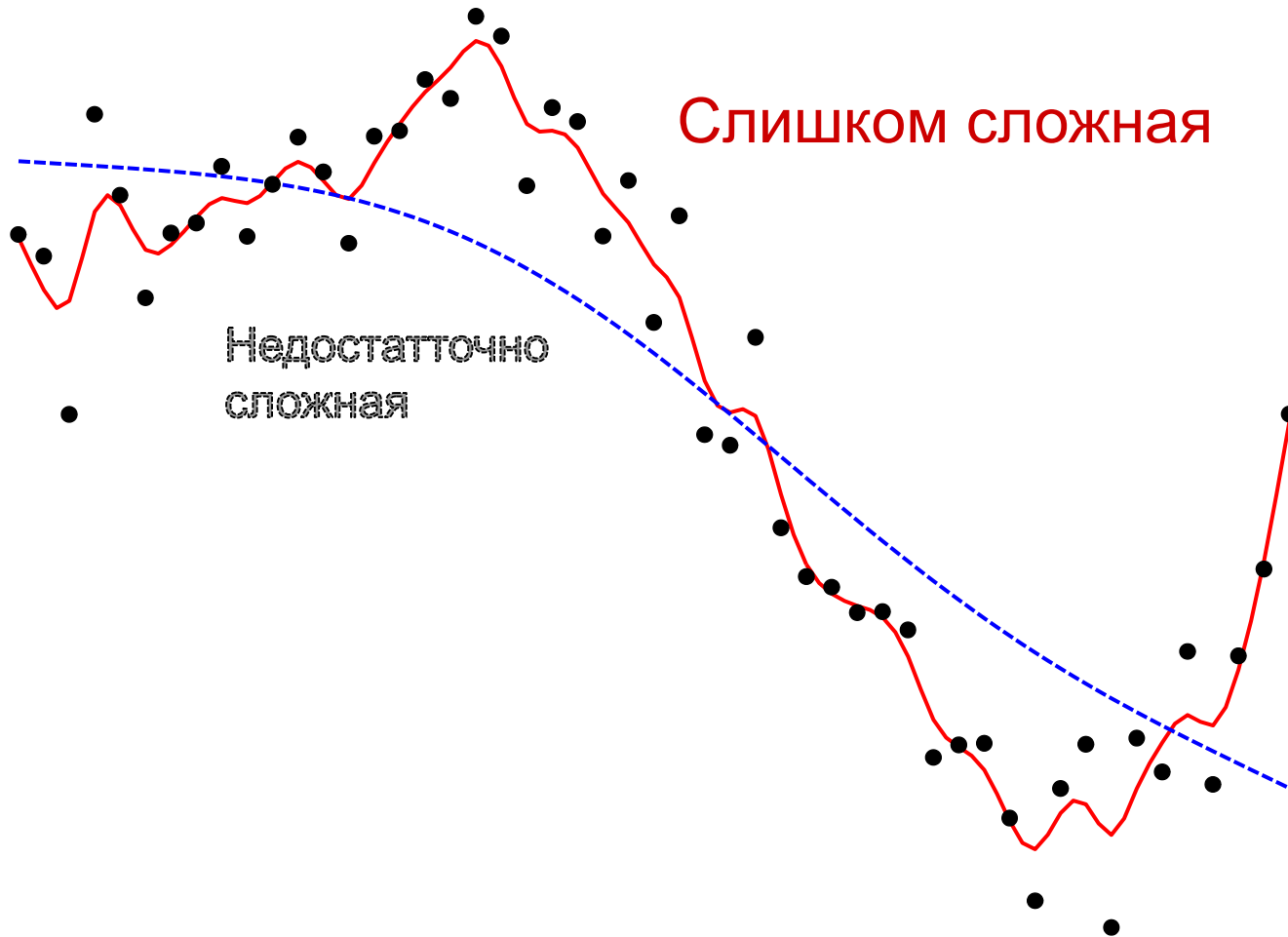
Сложность модели



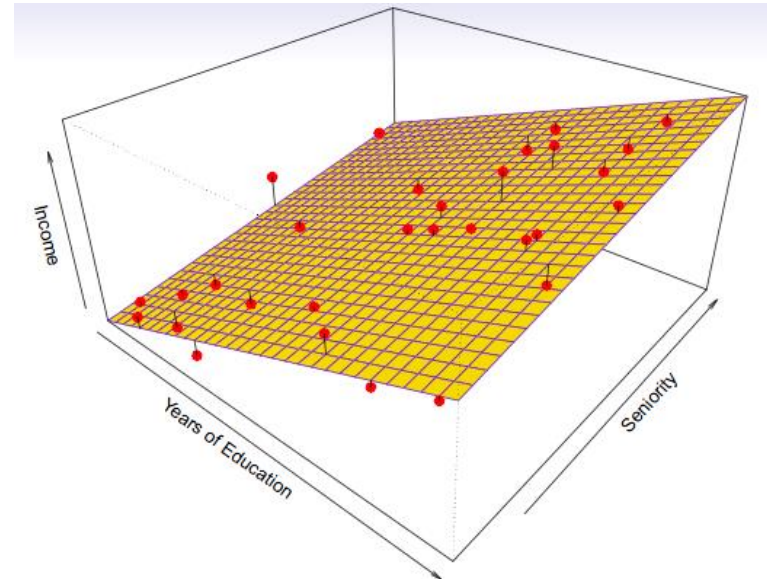
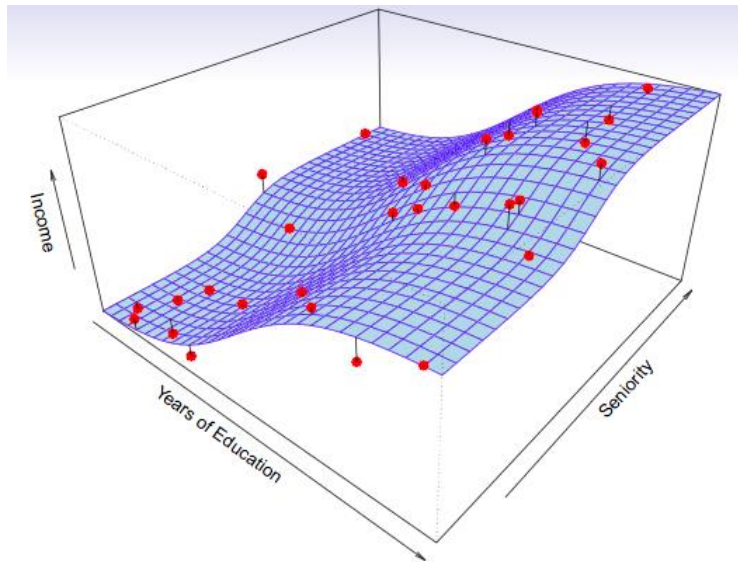
Сложность модели



Сложность модели



Проблема недообучения и переобучения



Модельный пример.

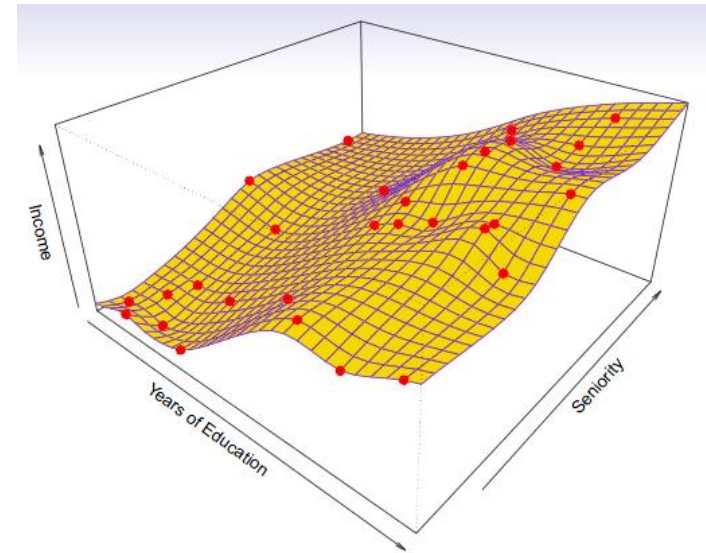
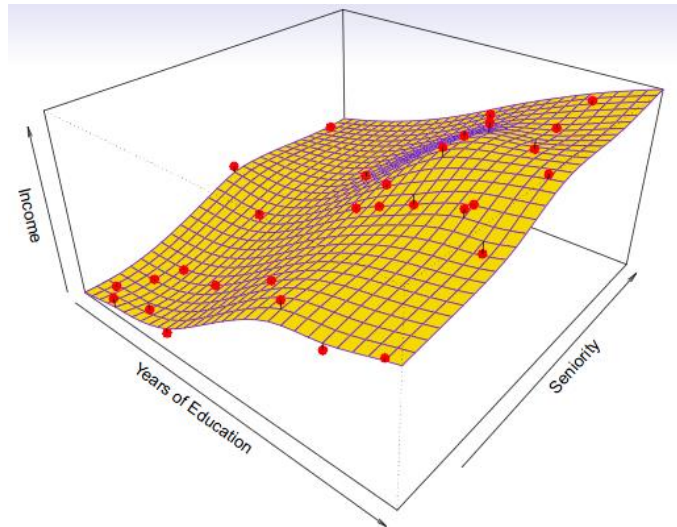
- Красные точки - наблюдения, синяя поверхность – истинная зависимость $\text{income} = f(\text{education}, \text{seniority}) + \epsilon$

- 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!

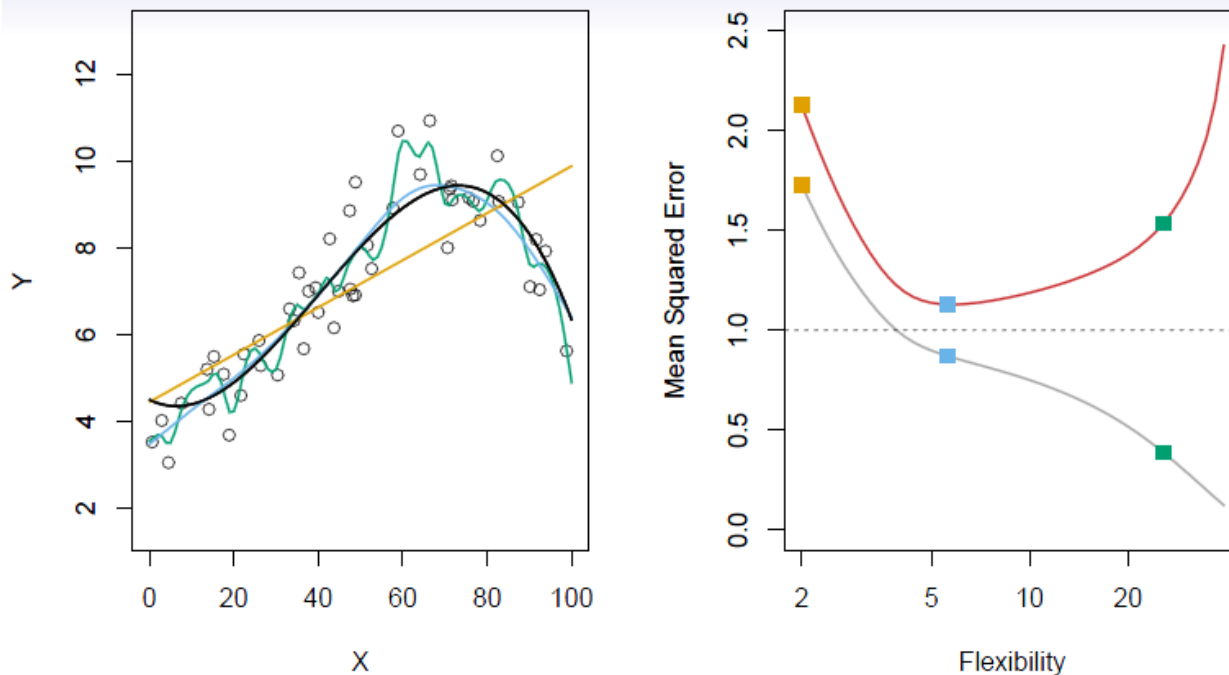
Недообучение vs переобучение

- Основная проблема машинного обучения!!!
 - Недообучение - низкое качество (большой эмпирический риск) на тренировочном наборе и на этапе скоринга
 - Переобучение - высокое качество (маленький эмпирический риск) на тренировочном наборе и плохое качество на этапе скоринга (большой эмпирический риск)
- Причины:
 - Сложность модели: например, для параметрических моделей много степеней свободы (параметров модели) или слишком сложное уравнение или большая норма вектора параметров
 - Плохое качество данных: шум и выбросы, малый объем или несбалансированность тренировочной выборки
 - Зависимости в пространстве признаков
- Обобщающая способность:
 - способность алгоритма «качественно» прогнозировать отклик для объектов одной природы (из одной генеральной совокупности), которых не было в тренировочном наборе
 - Как оценить?

Борьба с переобучением

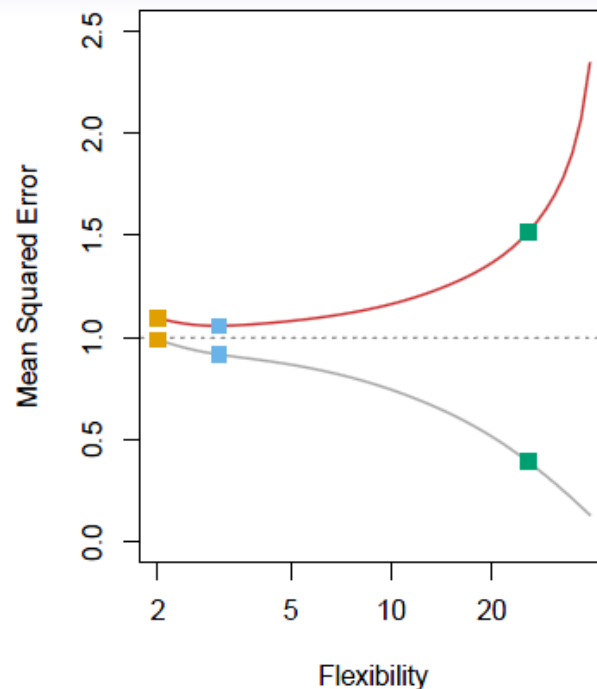
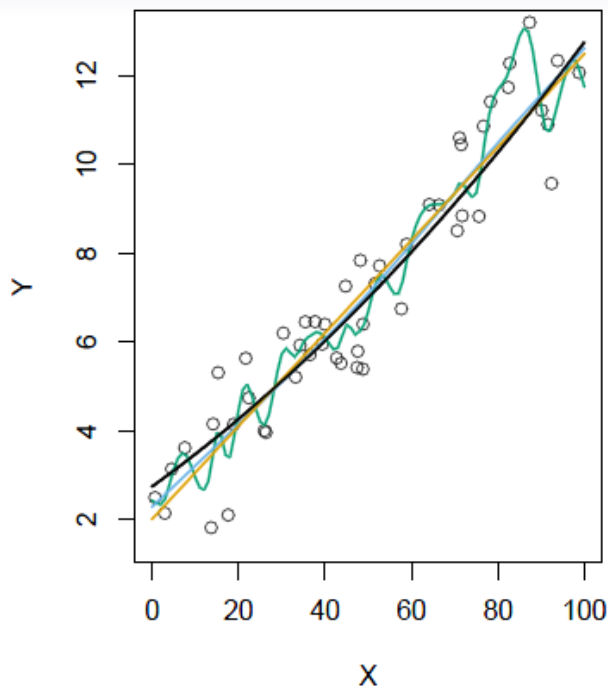
- Ограничить сложность модели (например, регуляризация)
- Преобразовать данные (удалять шум, уменьшать размерность и тд.)
- Использовать теоретические оценки обобщающей способности для некоторых методов обучения (обычно бесполезно, т.к. это оценки сверху)
- Эмпирически оценивать обобщающую способность с помощью тестовой выборки (или процедуры, имитируя проверку на тестовой выборке):
 - Строим модель на обучающем наборе данных и хотим, чтобы она была наилучшей.
 - Можем взять, например, квадратичную функцию потерь и оценить ее через среднеквадратичную ошибку MSE_{Tr}
 - Оценка может быть смещена в сторону более сложных моделей.
 - Поэтому мы вычисляем оценку MSE_{Te} , используя тестовый набор данных, который не участвовал в обучении модели

Оценка качества модели (сложная зависимость, много шума)



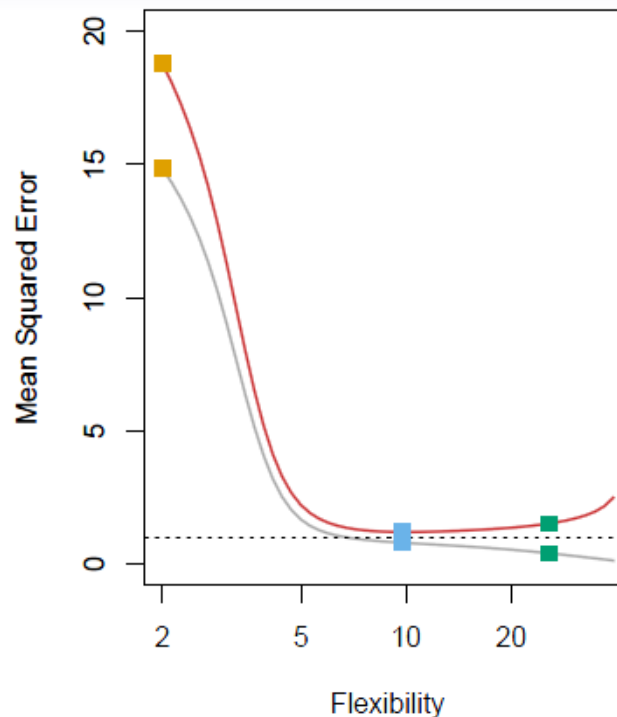
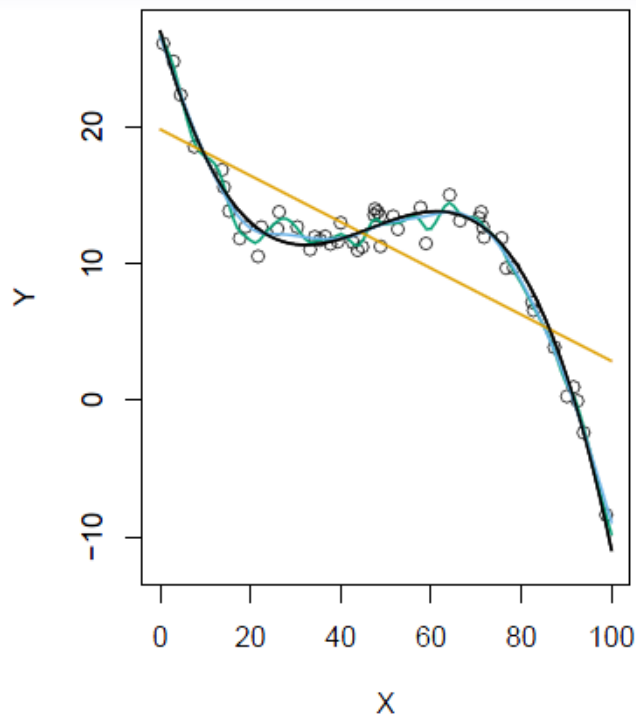
- Кривая, обозначенная черным цветом, - истинные значения.
- Красная кривая на правом рисунке – MSE_{Te} , серая кривая – MSE_{Tr} .
- Оранжевая, голубая и зеленая кривые соответствуют подгонке моделей различной сложности.
- Простые модели недообучены, сложные модели переобучены

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

MSE декомпозиция

$$\begin{aligned} MSE &= E[(a(x) - y(x))^2] = E[a(x)^2] + E[y(x)^2] - E[2a(x)y(x)] = \\ &= \text{Var}(a(x)) + \text{Var}(y(x)) + (E[a(x)] - E[y(x)])^2 \end{aligned}$$

Дисперсия прогноза

Дисперсия шума

Квадрат смещения

(не зависит от модели)

Компромисс: Дисперсией vs Смещение!!!!

Сложнее модель => точнее приближение => меньше смещение +++

Сложнее модель => больше параметров => больше дисперсия ---

... и наоборот ...

Поиск баланса между точностью и сложностью = поиск компромисса
между смещением и дисперсией

MSE декомпозиция (примеры)

$$y = y(x) + \varepsilon$$

y – наблюдения отклика, $y(\cdot)$ – истинная зависимость, ε – шум $N(0, \sigma)$

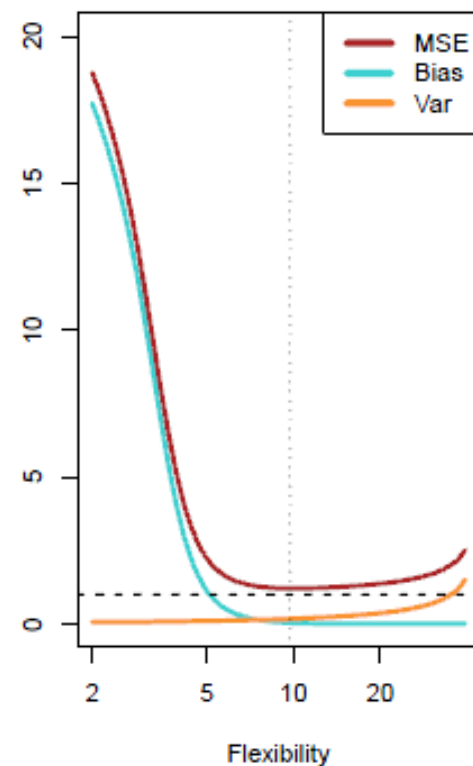
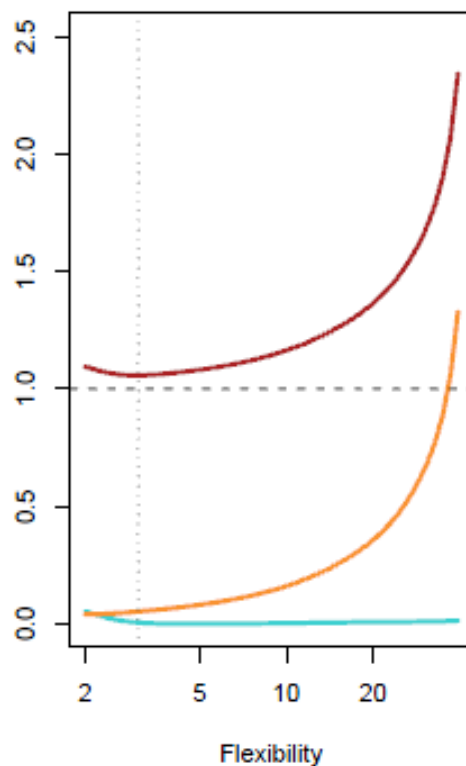
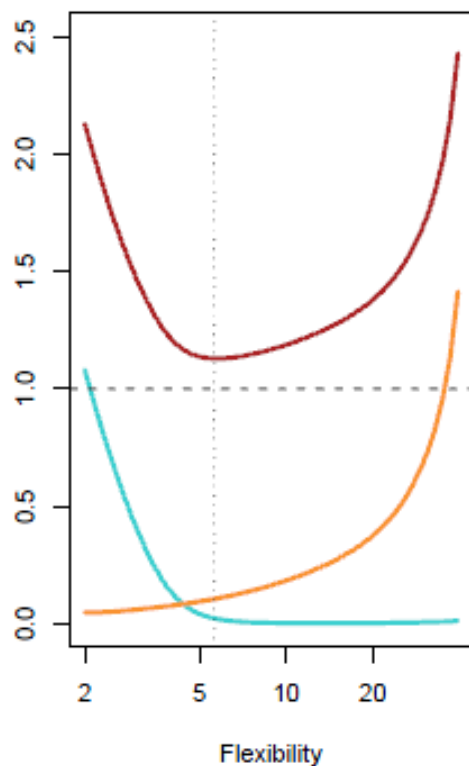
- KNN (к-число соседей):

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

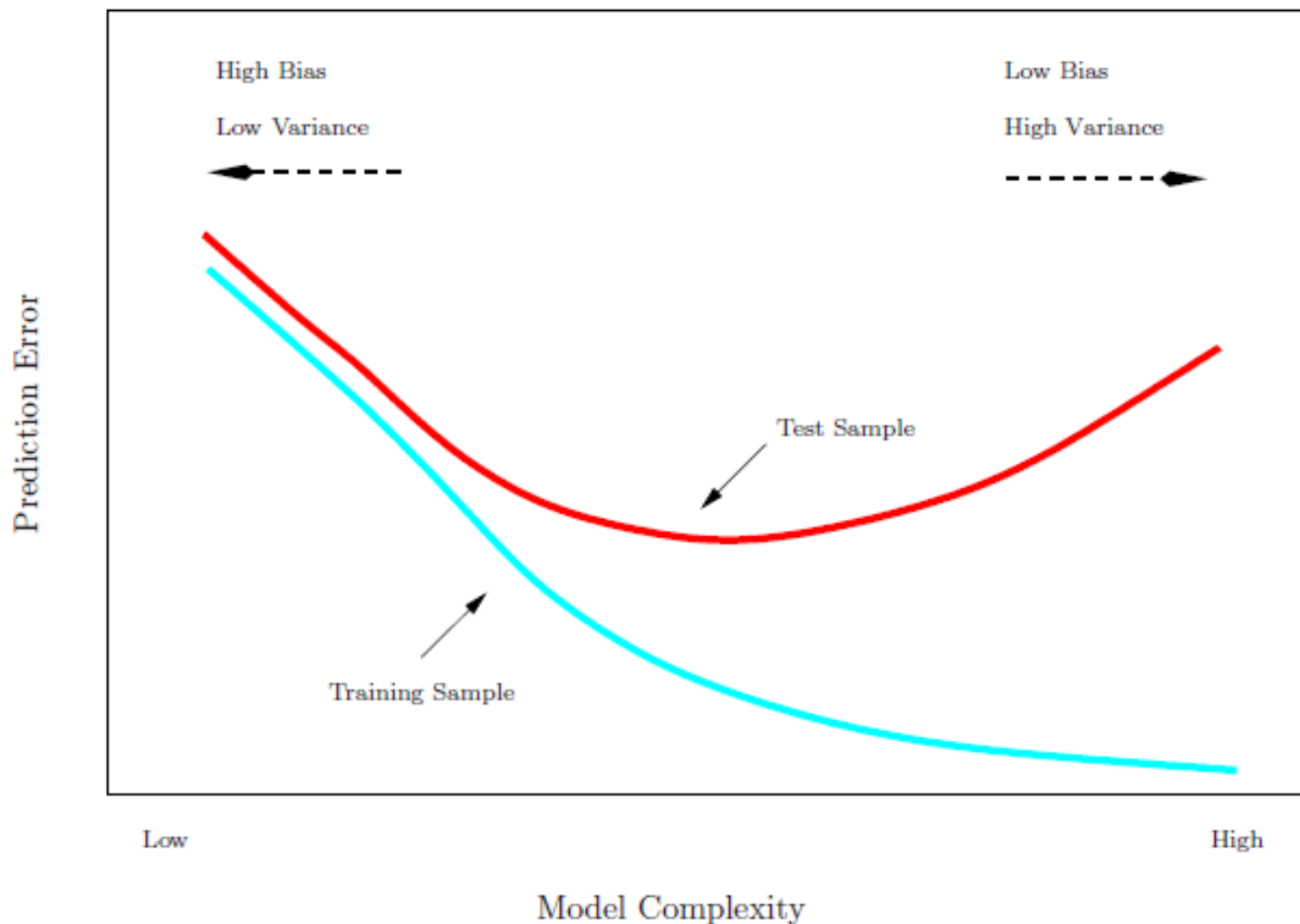
- Линейная регрессия (p -размерность пространства признаков, l – размер выборки):

$$MSE = \sigma^2 + \frac{p}{l} \sigma^2 + \frac{1}{l} \sum_x [E[a(x)] - y(x)]^2$$

Компромисс отклонения и смещения для трех примеров



Качество на обучающем и тестовом наборе



Валидация, кросс-валидация и бутстреппинг

- Эти методы позволяют:
 - оценить ошибки прогнозирования тестового набора
 - найти оценки параметров модели
 - выбрать лучшую модель
- Различия между *ошибкой тестирования* и *ошибкой обучения*:
 - Ошибка тестирования - это усредненная ошибка, которая возникает в результате применения модели машинного обучения для прогнозирования отклика на новом наблюдении, которое не было задействовано в процессе обучения.
 - Ошибка обучения вычисляется после применения алгоритма машинного 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.

$$HO(\mu, Z, Z_{val}) = Q(\mu(Z \setminus Z_{val}), Z_{val})$$

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:

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

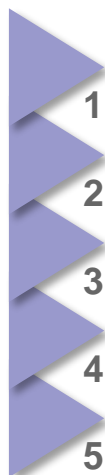
Оценка моделей

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

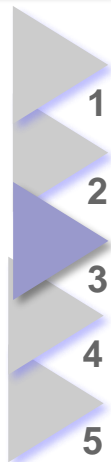
Выбор модели

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

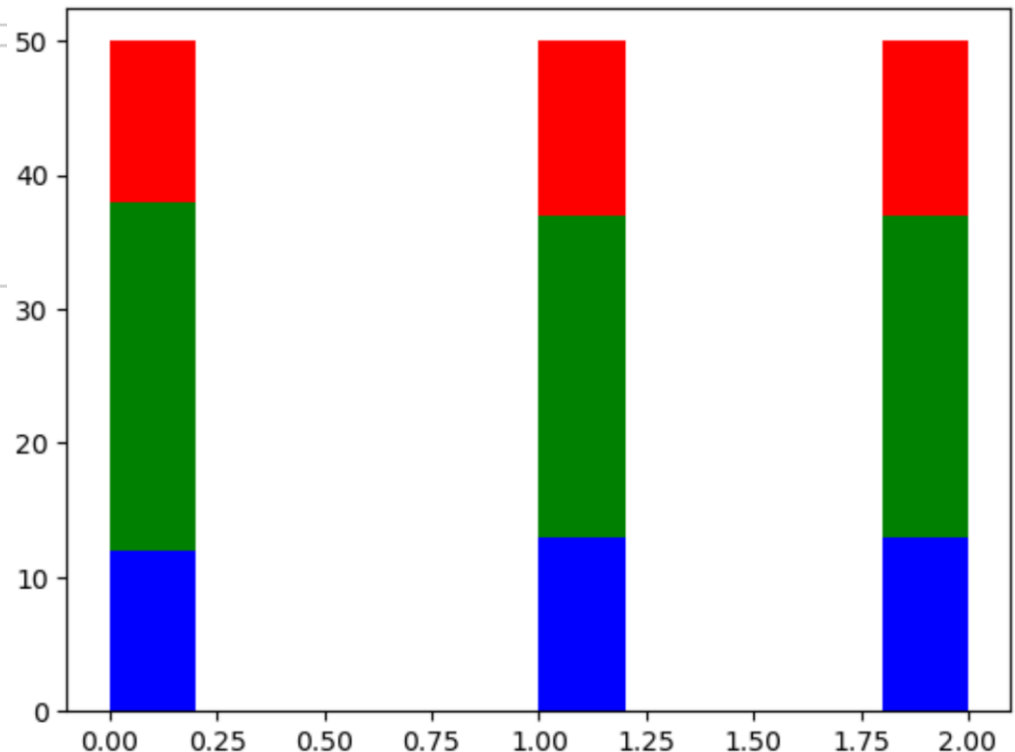
Пример (Python)

```
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)
```

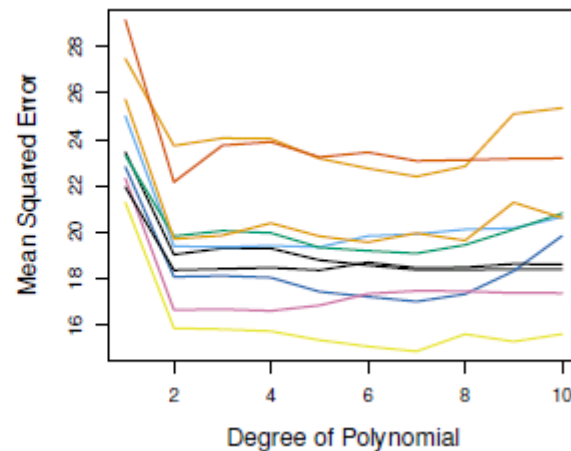
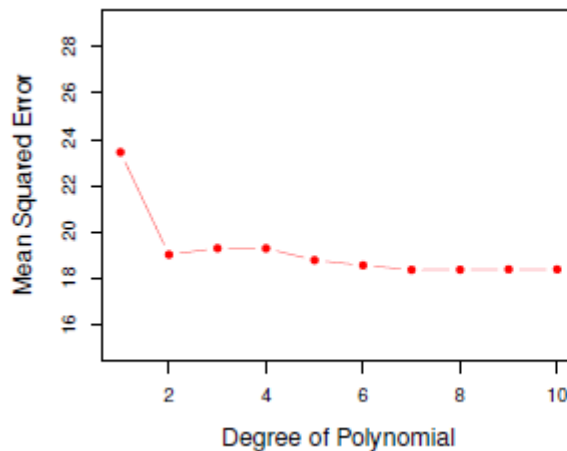
```
plt.hist(iris.target, color="red")
plt.hist(y_train, color="green")
plt.hist(y_test, color="blue")
pass
```

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.



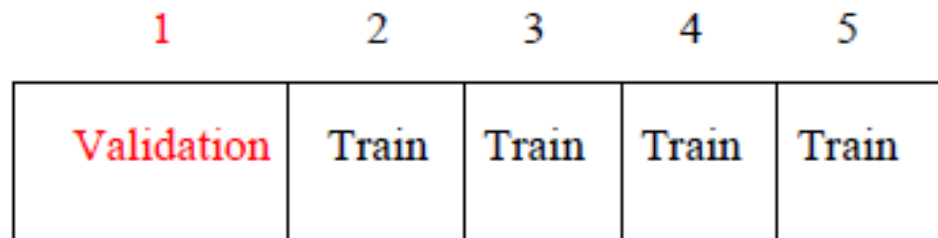
Слева показано одиночное разбиение, справа - множественное

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

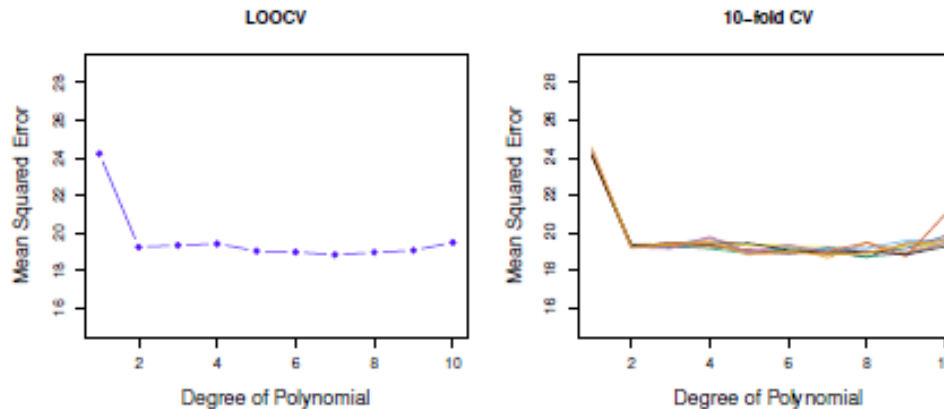
- Широко используемый подход для оценки ошибки тестирования.
- Оценки могут быть использованы для:
 - выбора оптимальной модели,
 - оценки тестовой ошибки результирующей выбранной модели.
- Идея - разделить данные на K частей равного размера. Мы удаляем часть k , строим модель на оставшихся частях, а 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 Z_1, \dots, Z_K , где Z_k - наблюдения в части k . l_k - наблюдений в части k , удобно брать $l_k = l/K$
- Вычислим: $CV_Z(\mu) = \sum_{k=1}^K \frac{l_k}{l} Q(\mu(Z \setminus Z_k), Z_k)$



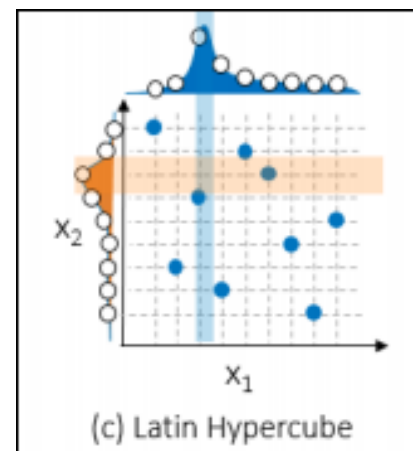
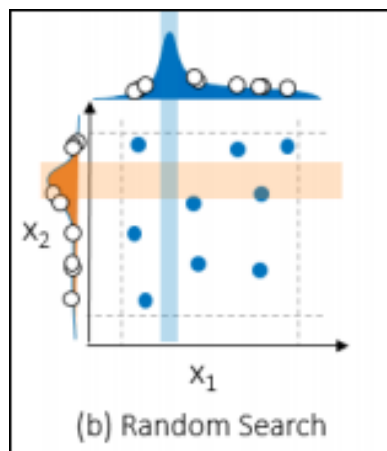
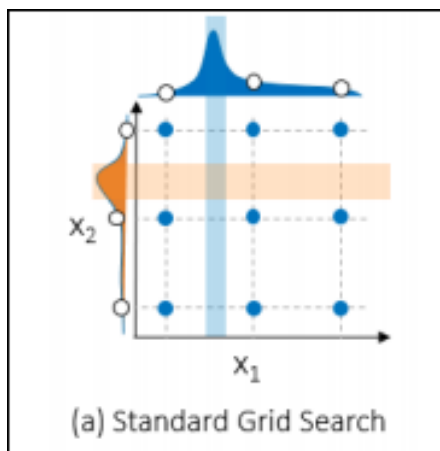
- При $K = l$ имеем l папок или кросс-валидацию с попеременным исключением одного наблюдения – скользящий контроль (*leave-one out cross-validation*, LOOCV).



Кросс-валидация для оценки метапараметров (мета-обучение) и выбора модели

- Задают стратегию перебора вариантов метапараметров
- Запускают кросс-валидацию для разных значений метапараметров
- Рассчитывают кросс-валидационные ошибки для каждого варианта
- Выбирают лучшее значение метапараметра по кросс-валидационной ошибке
- Перестраивают модель на всей выборке с этим значением метапараметра

Кросс-валидация и валидация для выбора метопараметров (мета-обучение)



■ AutoML:

- ☐ Поиск по решетке
- ☐ Случайный поиск
- ☐ Латинский гиперкуб
- ☐ Эволюционные и генетические алгоритмы поиска
- ☐ «Мета» оптимизация
- ☐ Байесовская оптимизация

■ Оценка качества:

- ☐ Не обязательно (и даже как правило) оценка качества для выбора модели совпадает с функцией потерь для обучения модели

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 – 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]}
```

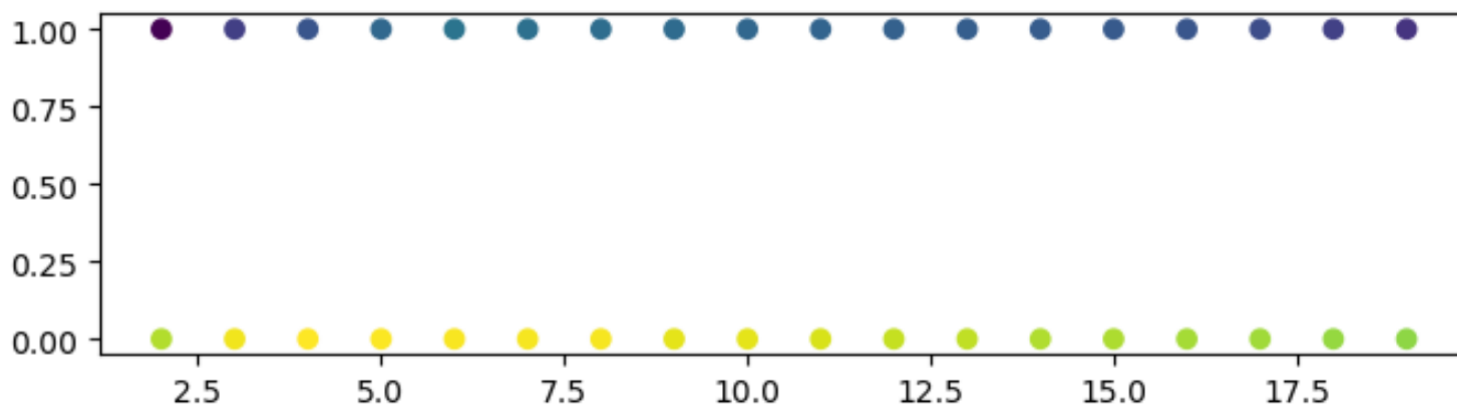
```
# 5-fold cross-validation  
GSCV = GridSearchCV(model, parameters, cv=5)  
GSCV.fit(X, y)  
pass
```

```
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
```

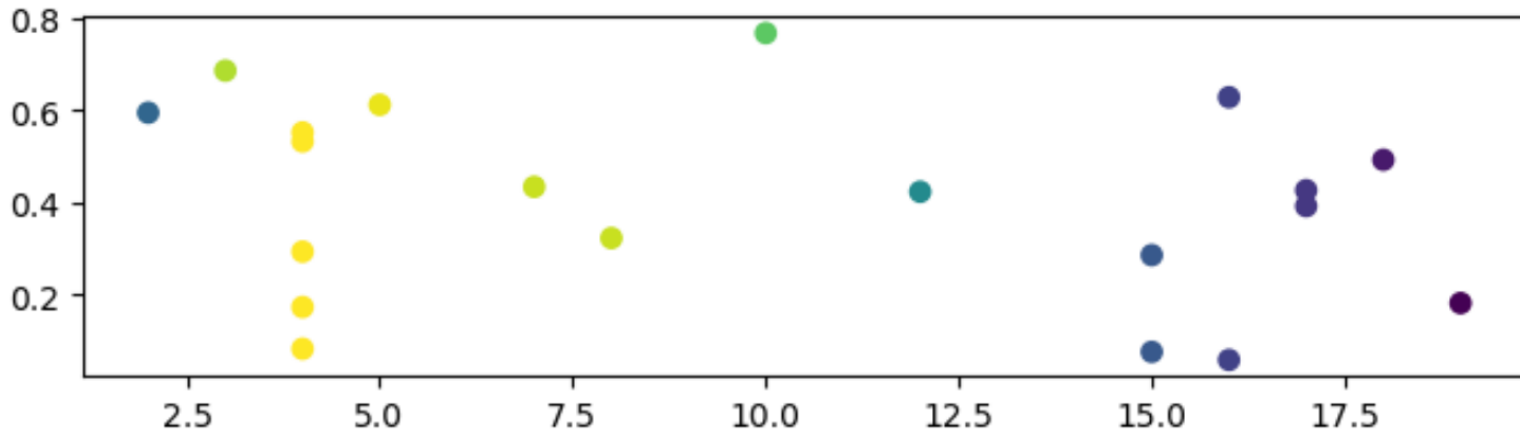
Пример – Случайный поиск (Python)

```
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)

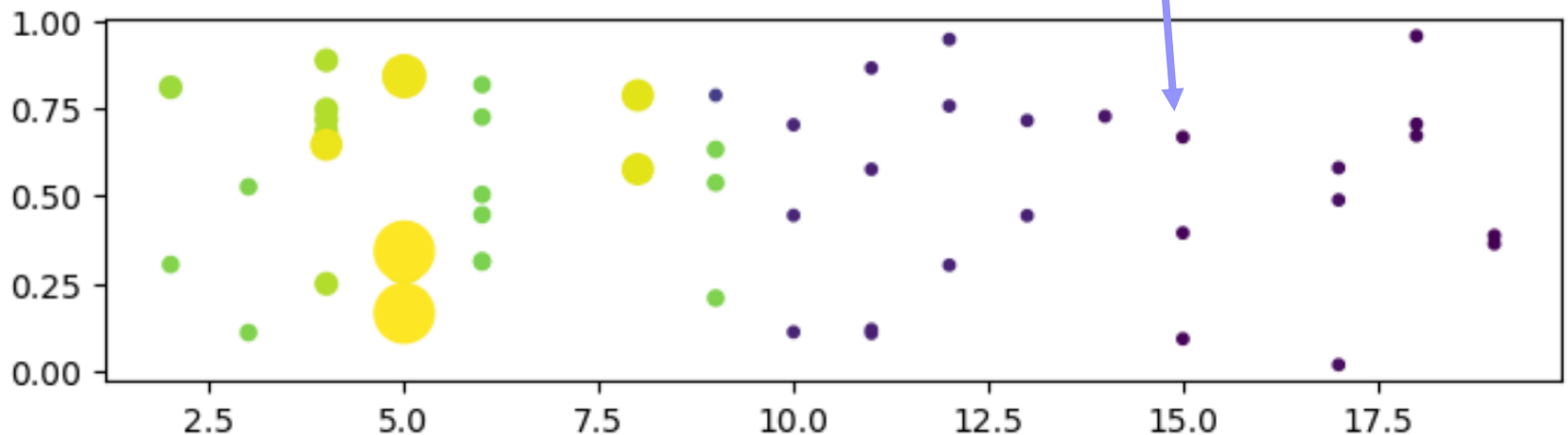
```
HRSV.best_params_
```

```
{'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



Бутстрэппинг

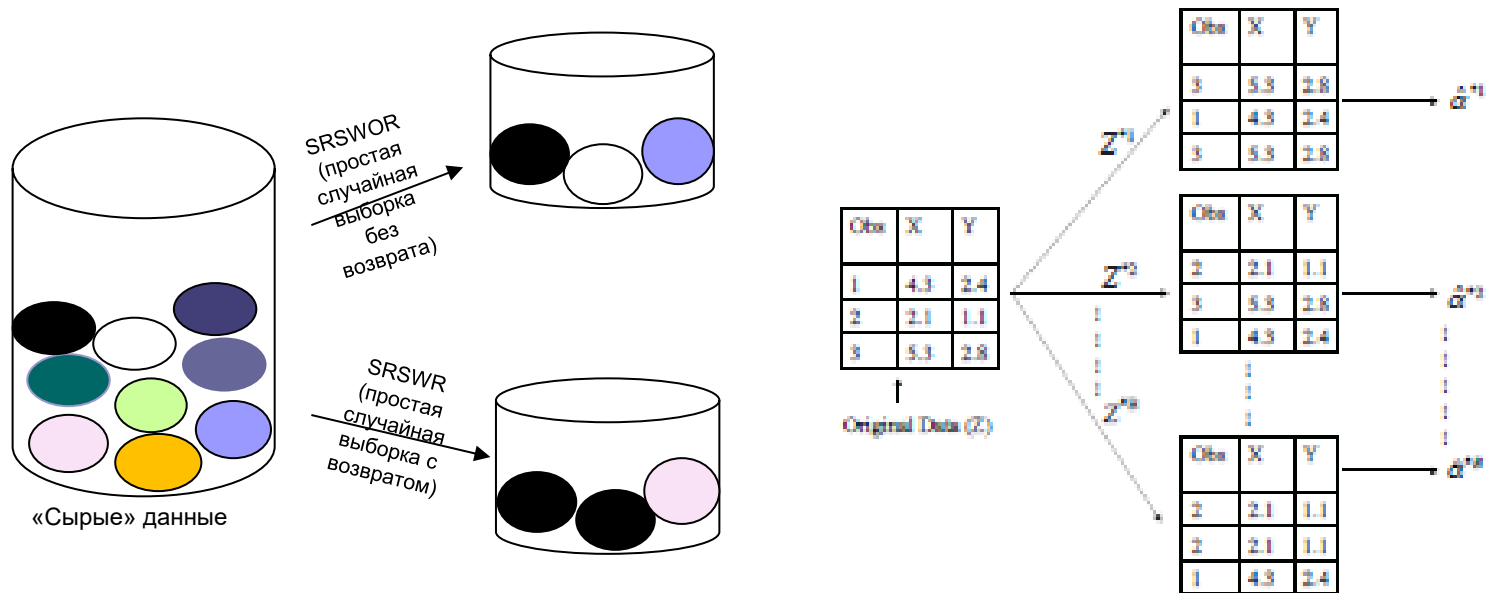
- *Бутстрэппинг* представляет собой мощный статистический инструмент, который может быть использован для количественной оценки неопределенности.
- Например, он может позволить произвести оценку стандартной ошибки коэффициента или доверительного интервала для этого коэффициента.
- Использование термина бутстреппинг происходит от фразы, чтобы *to pull oneself up by one's bootstraps*, - «пример» 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.

Бутстрэппинг

- Подход бутстрэппинга 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 бутстрэппинга, а некоторые нет вообще.

Демонстрационный пример



- Графическая иллюстрация бутстреппингового подхода на маленькой выборке
- Каждый бутстреппинговый набор данных содержит наблюдения, отобранные с заменой из исходного набора.
- Каждый такой набор данных начальной используется для получения оценки

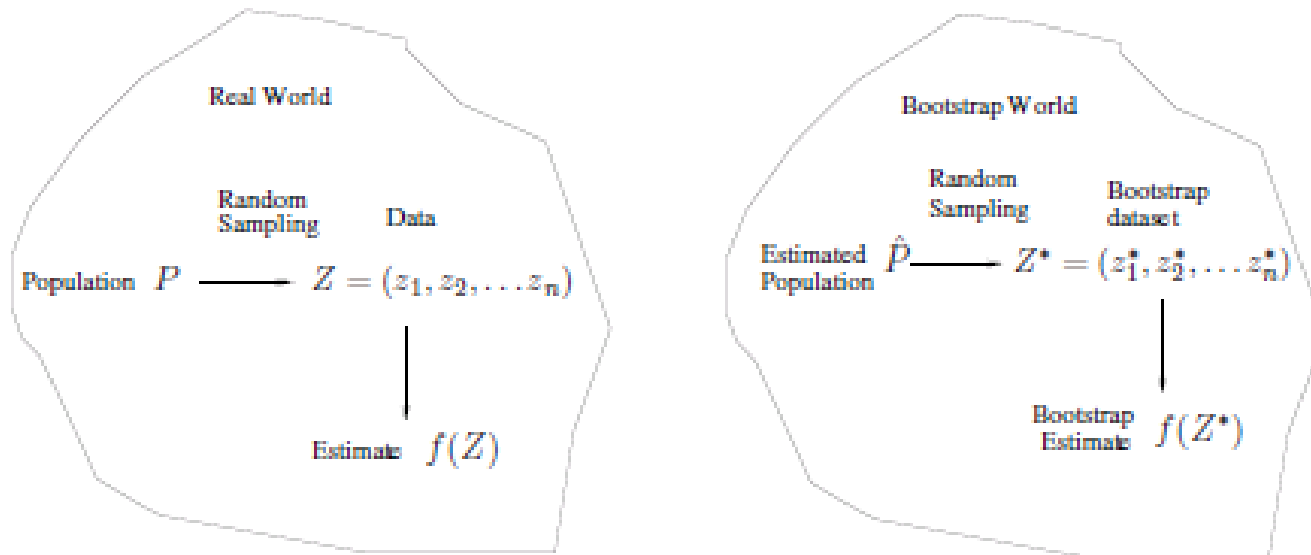
Бутстрэппинг

- Обозначая k -й набор данных бутстрэппинга как Z^{*k} , мы используем его, чтобы выполнить новую оценку для a^{*k}
- Эта процедура повторяется B раз для некоторого большого значения B (например, 100 или 1000), чтобы получить B различных наборов данных бутстрэппинга $Z^{*1}, Z^{*2}, \dots, Z^{*B}$, и B соответствующих оценок $a^{*1}, a^{*2}, \dots, a^{*B}$
- Оценим среднее и стандартную ошибку этих оценок бутстрэппинга:

$$\tilde{a} = \frac{1}{B} \sum_{r=1}^B (a^{*r}), SE_B(a) = \sqrt{\frac{1}{B-1} \sum_{r=1}^B (\tilde{a} - a^{*r})^2}$$

- Они служат в качестве оценки, полученной на тестовом наборе данных.

Общая схема бутстреппинга



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

Как бутстрепинг оценивает ошибку прогнозирования

- При кросс-валидации каждый из K блоков валидации отличается от других $K - 1$, используемых для обучения: *перекрытия нет*.
- Для оценки ошибки прогнозирования с помощью бутстреппинга мы могли бы использовать каждый набор данных бутстреппинга как валидационный набор для остальных (или наоборот), но:
 - каждая выборка бутстреппинга имеет значительное перекрытие с исходным набором (около двух третей).
 - 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 $OOB(\mu) = \sum_{k=1}^K \frac{l_k}{l} Q(\mu(Z^{*k}), Z \setminus Z^{*k})$

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)
```

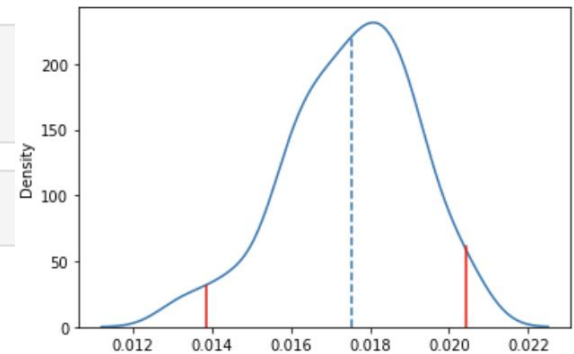
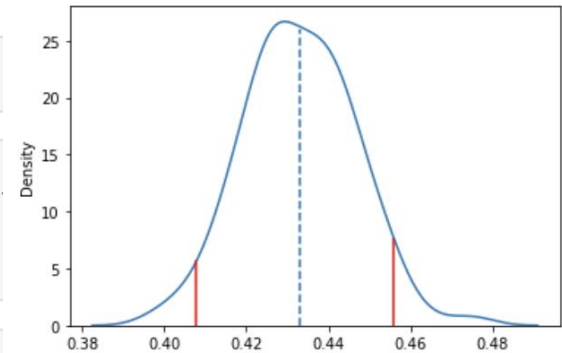
```
features = X[["MedInc", "HouseAge"]]
```

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

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

```
sns.kdeplot(data=coefs[:, 0])  
plt.axvline(coefs[:, 0].mean(), 0, 0.93, ls="--")  
plt.axvline(np.quantile(coefs[:, 0], 0.025), 0, 0.20, c="red")  
plt.axvline(np.quantile(coefs[:, 0], 0.975), 0, 0.27, c="red")
```

```
sns.kdeplot(data=coefs[:, 1])  
plt.axvline(coefs[:, 1].mean(), 0, 0.91, ls="--")  
plt.axvline(np.quantile(coefs[:, 1], 0.025), 0, 0.13, c="red")  
plt.axvline(np.quantile(coefs[:, 1], 0.975), 0, 0.25, c="red")  
####
```

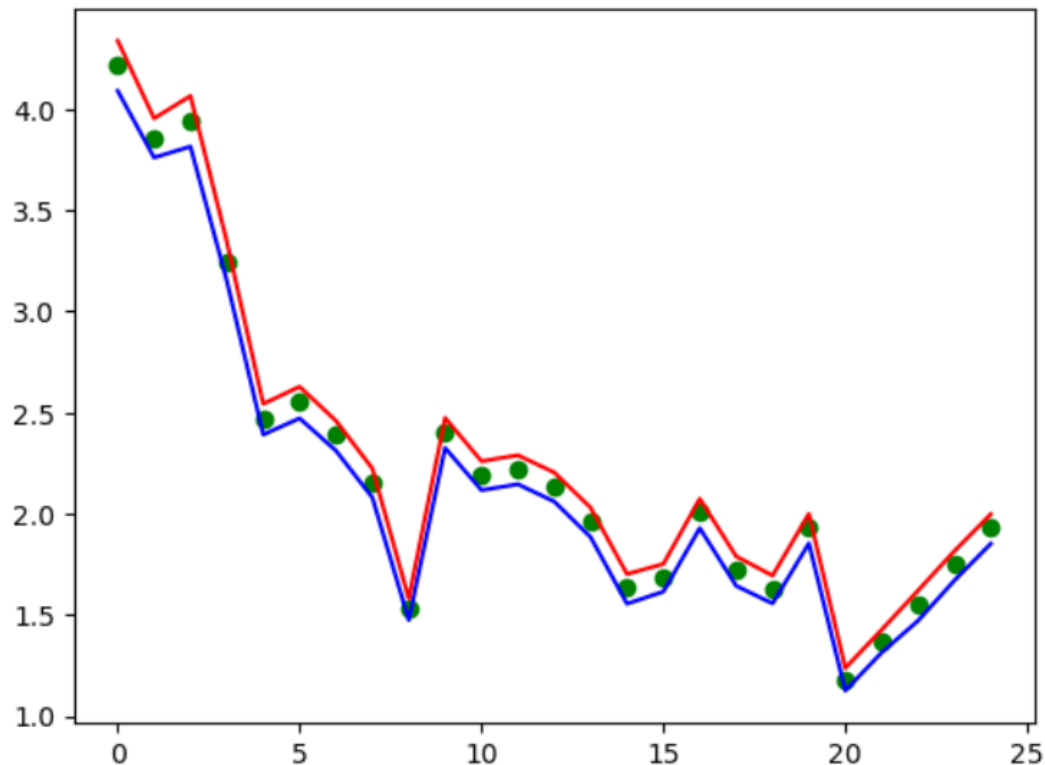


Бутстреп-регрессия (Python)

```
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
```



Контрольный опрос

