«Машинное обучение и анализ данных»

Метрические методы

Александр Дьяконов

21 сентября 2020 года

План

Методы, которые используют расстояния

Геометрия

Ближайший центроид

Метод k ближайших соседей

Теоретическое обоснование

Обобщения (весовые, на регрессию и т.п.)

Метрики

Приложения

Эффективные методы поиска соседей

Метрические алгоритмы

«distance-based» – анализируются расстояния

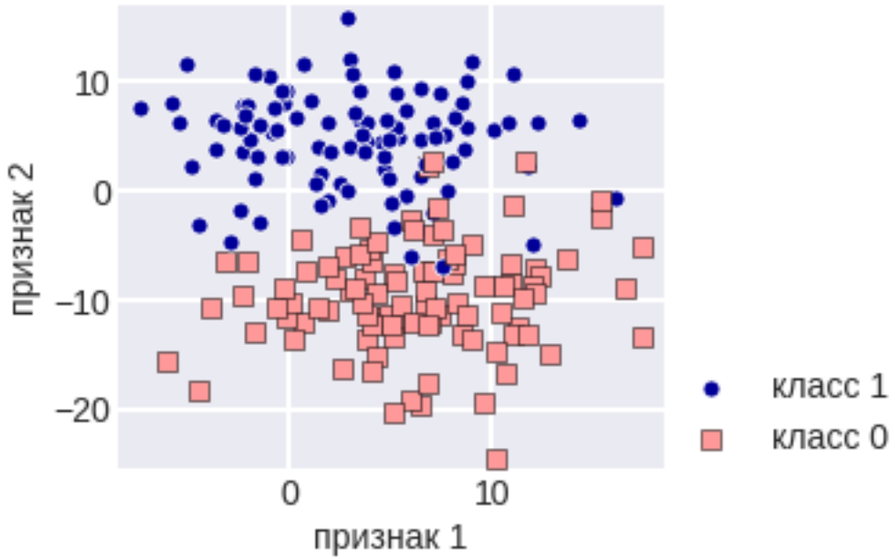
$$\rho(x, x_1), \ldots, \rho(x, x_m)$$

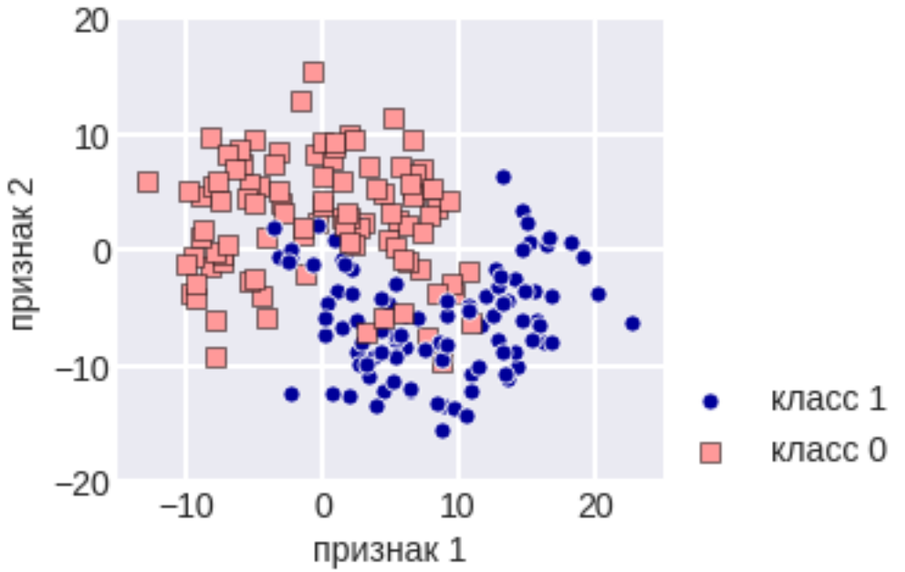
- **Nearest centroids algorithm / Distance from Means**
- **kNN (Nearest Neighbor)**

ещё называют:

- **«memory-based»**
- **«instance-based»**
- **«non-parametric»**

Модельные задача классификации





Ближайший центроид (Nearest centroid algorithm)

Задача классификации на непересекающиеся классы с вещественными признаками:

$$Y = \{1, 2, \ldots, l\}$$

$$x_i \in \mathbb{R}^n$$

центроиды:

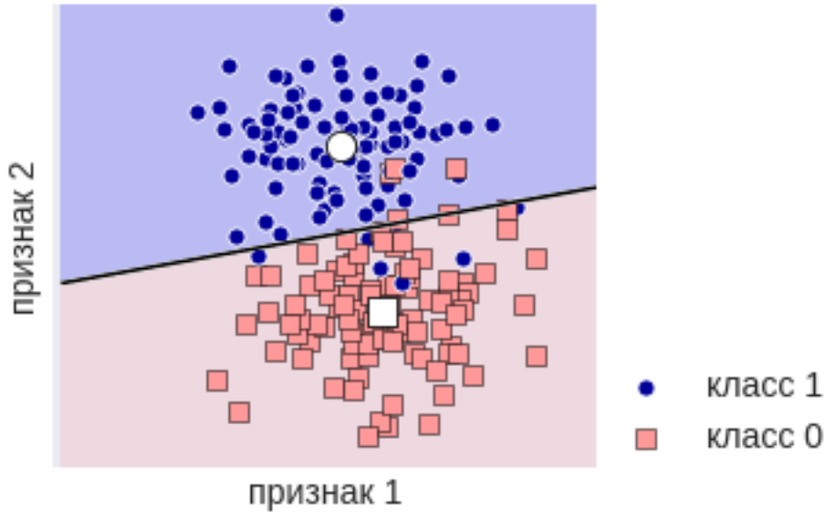
$$c_j = \frac{1}{|\{i : y_i = j\}|} \sum_{i: y_i = j} x_i$$

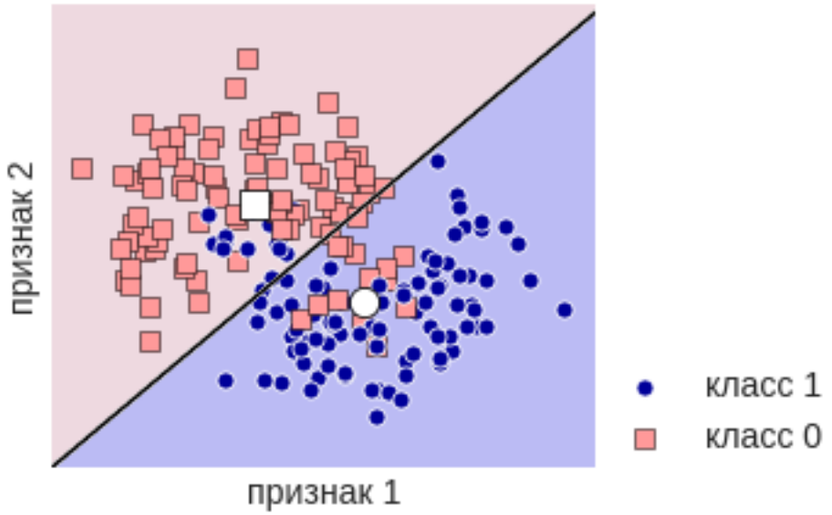
классификация:

$$a(x) = \arg\min_j \rho(x, c_j)$$

обобщается на случаи, когда можно вычислить «средний объект»

Ближайший центроид (Nearest centroids algorithm)





+ хранить только центроиды (их можно адаптивно менять)

+ понятие центроида можно менять («средний объект»)

+ простая реализация

+ размер модели = число классов $\times$ описание центроида

– очень простой алгоритм

интуитивно подходит в задачах,

где объекты разных классов распределены «колоколообразно»

Минутка кода: ближайший центроид (Nearest centroids algorithm)

```
from sklearn.neighbors.nearest_centroid import NearestCentroid
model = NearestCentroid()
model.fit(X, y)
a = model.predict(X2)
```

есть параметр

```
metric='euclidean'
```

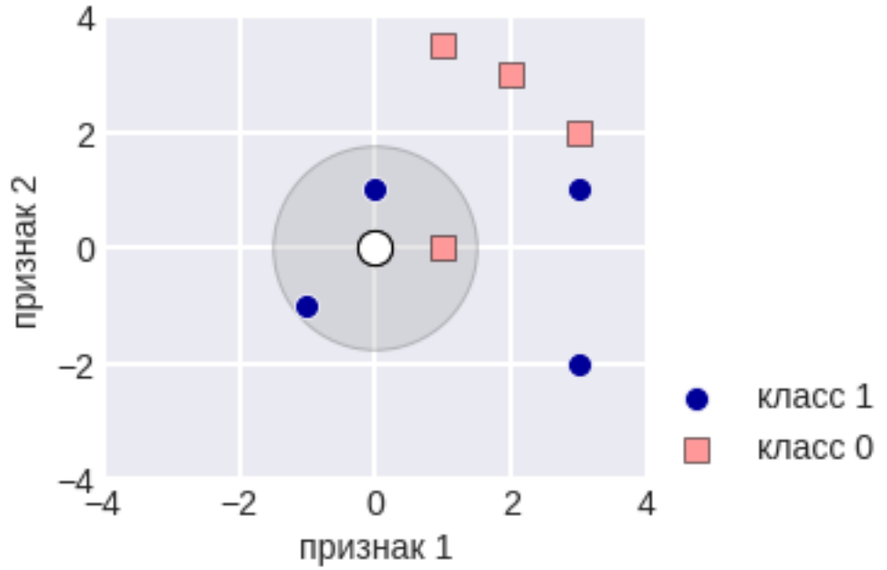
Подход, основанный на близости

Задача классификации

$$a(x) = \text{mode}(y_i \mid x_i \in N(x))$$

Задача регрессии

$$a(x) = \text{mean}(y_i \mid x_i \in N(x))$$



$N(x)$ – **окрестность (neighborhood) объекта** x

(похожие на него объекты)

Окрестность

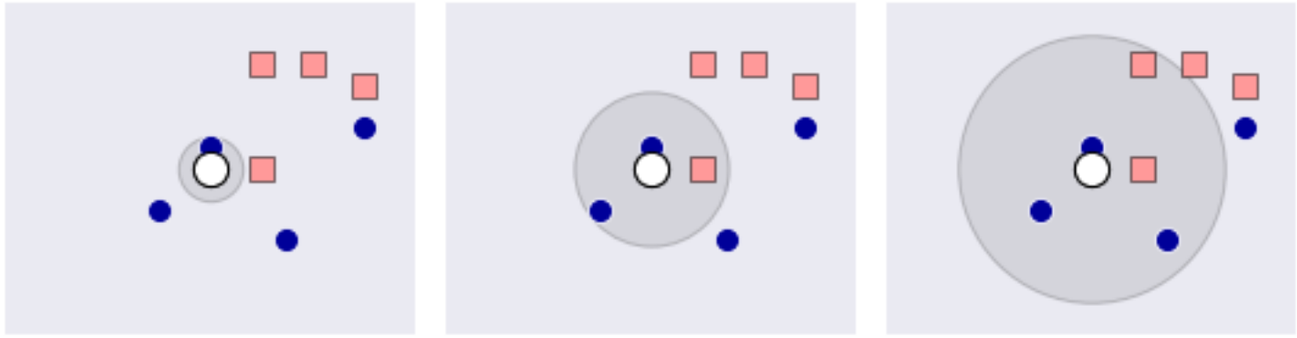
Если X – метрическое пространство с метрикой ρ,

пусть нумерация объектов такая, что

$$\rho(x, x_1) \le \ldots \le \rho(x, x_m)$$

k ближайших соседей: $N(x) = \{x_1, \ldots, x_k\}$

– метод k ближайших соседей (kNN = k nearest neighbours)



Есть также «Fixed-Radius Near Neighbor»

$$N(x) = \{x_t \mid \rho(x_t, x) \le R\}$$

Метод k ближайших соседей (kNN)

k = 1 – алгоритм ближайшего соседа (nearest neighbour algorithm)

формально нет обучения – храним всю выборку

работа алгоритма – просматриваем всю выборку

(+ вычисляем расстояние до каждого объекта обучения)

– ленивый алгоритм (lazy learning)

Гиперпараметр k можно выбрать на скользящем контроле дальше

Ещё гиперпараметры (потом):

- **метрика (+ параметры метрики)**
- **ядро (+ параметры ядра)**

Обоснование 1NN

Теорема. В достаточно однородном метрическом пространстве объектов бинарной задачи классификации ошибка 1NN не выше удвоенной ошибки оптимального алгоритма.

Пусть в некоторой области вероятность встретить объект класса 0 (без огр-я общ-ти) – $p \le 0.5$ – (это же и вероятность ошибки оптимального алгоритма), тогда

	объект	
сосед	**класс 0 –** p	**класс 1 –** $1-p$
класс 0 – p	pp	$p(1-p)$
класс 1 – $(1-p)$	$p(1-p)$	$(1-p)(1-p)$

вероятность ошибочной классификации

$$2p(1-p) = 2p - 2p^2 \le 2p$$

Термины

Нетерпеливый алгоритм (Eager learner)

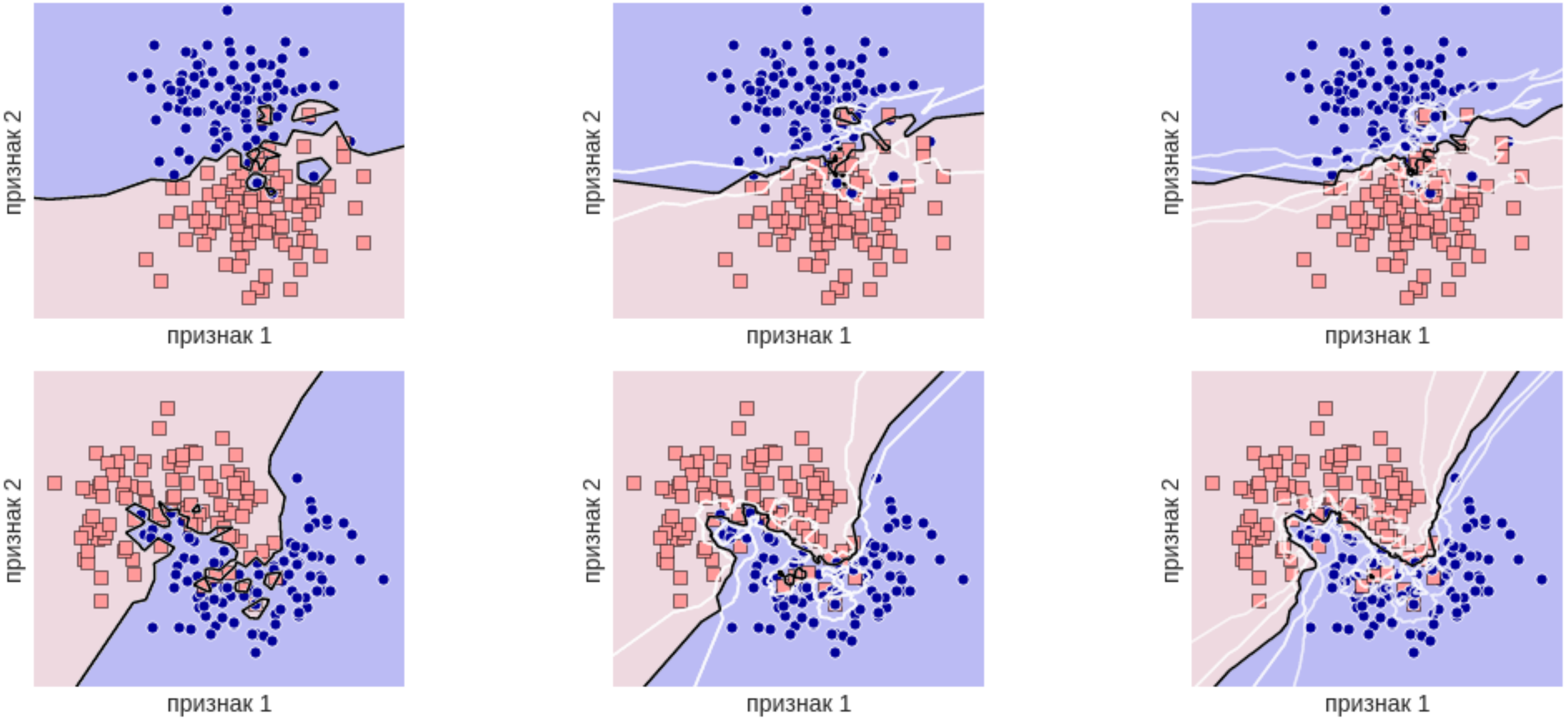
как только есть обучение – получает значения параметров (учит модель)

Ленивый алгоритм (Lazy learner)

не использует обучающую выборку до классификации

Решение модельной задачи при разном числе соседей

k=1 k=3 k=5



Решение модельной задачи при разном числе соседей

Как увидим дальше, k отвечает за «сложность модели»

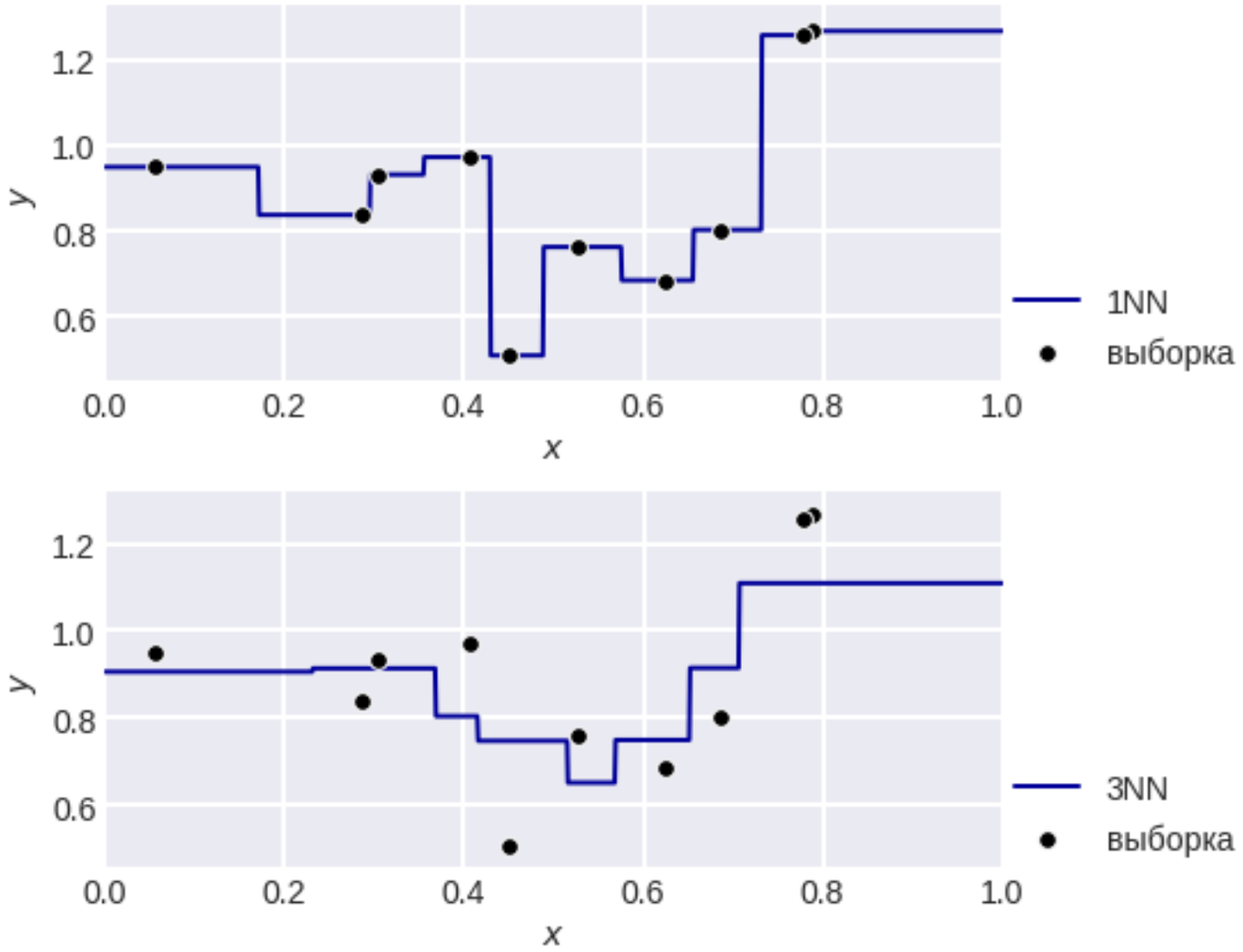
Минутка кода

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X, y)
a = model.predict(X2)
p = model.predict_proba(X2)[:, 1]
```

параметры разберём ниже

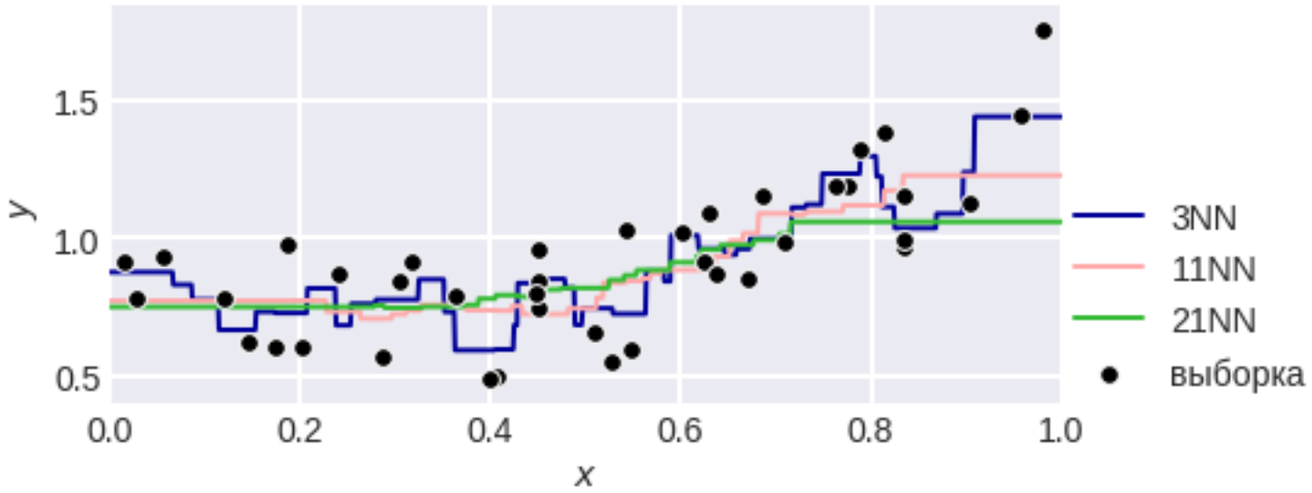
Метод ближайшего соседа

обобщается на регрессию



Метод ближайшего соседа

обобщается на регрессию



Минутка кода

```
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=3) # kNN-регрессия
model.fit(x_train, y_train) # обучение
a = model.predict(x_test) # ответ
```

Метод ближайшего соседа в регрессии

есть теоретическое обоснование

$$y = f(x) + \varepsilon \sim \{x_i, y_i\}_{i=1}^m,$$

$\mathbf{E}f^2 < +\infty$ **(больше нет ограничений)**

$a \sim k\text{NN}$ **«universally consistent»:**

Если $k \to +\infty$ **и** $k / n \to 0$**, то**

$$\mathbf{E}\,|a(x) - f(x)| \xrightarrow[n \to +\infty]{} 0$$

Кстати, это аппроксимация непрерывной функцией

Теоретическое обоснование для других метрических методов

[Fix, Hodges, 51]: classification + regularity, R^d

[Cover, Hart, 65, 67, 68]: classification + regularity, any metric

[Stone, 77]: classification, universal, R^d

[Devroye, Wagner, 77]: density estimation + regularity , R^d

[Devroye, Gyorfi, Kryzak, Lugosi, 94]: regression, universal, R^d

[Chaudhuri, Dasgupta, 14]: classification, nice metric/measure.

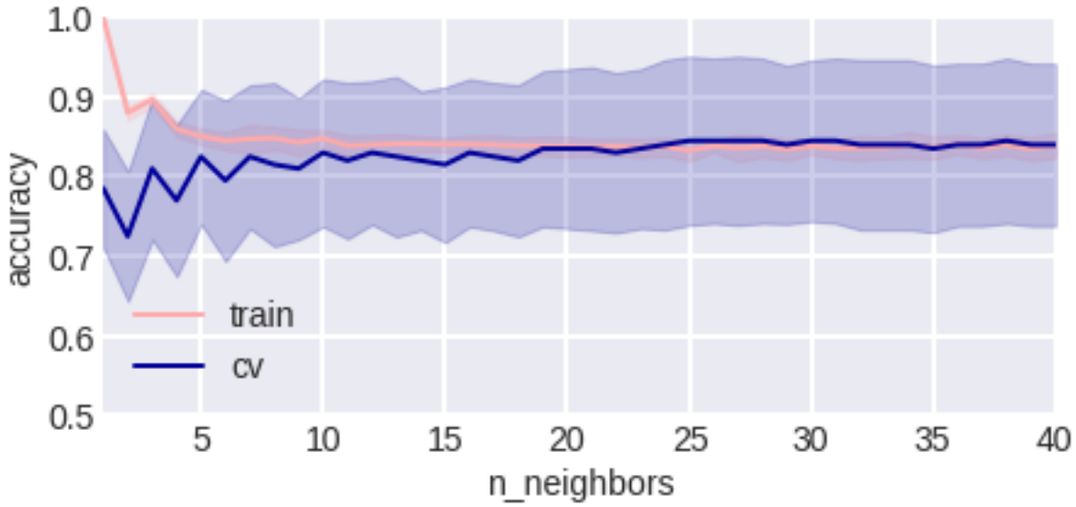
Везде k должен расти, но не очень быстро...

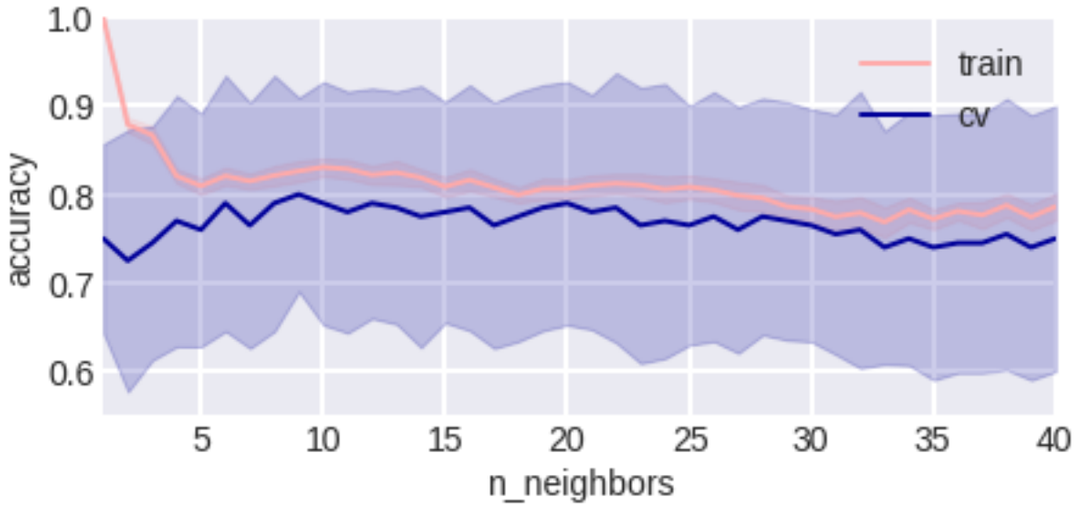
G. H. Chen, D. Shah E«xplaining the Success of Nearest Neighbor Methods in Prediction» // Publisher: Now Foundations and Trends https://devavrat.mit.edu/wp-content/uploads/2018/03/nn_survey.pdf

Подбор гиперпараметров специальными методами контроля

```
# cv-контроль
from  sklearn.model_selection import KFold
cv = KFold(n_splits=10, shuffle=True, random_state=2)
# модель
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5)
# параметр
param_name = "n_neighbors"
# его значения
pars = np.arange(1, 41)

# сделать тест
from sklearn.model_selection import validation_curve

train_errors, test_errors = validation_curve(model,
                            X, y,
                            param_name=param_name,
                            param_range=pars,
                            cv=cv.split(X),
                            scoring='accuracy',
                            n_jobs=-1)
```

будет дальше...



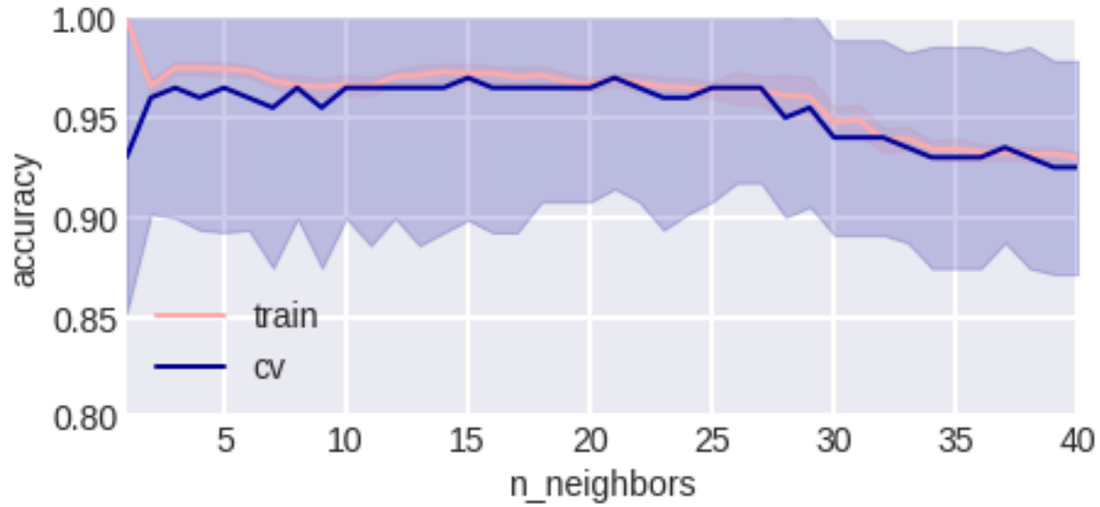


Как был получен второй график? с k_{opt}=9

Подбор гиперпараметров специальными методами контроля

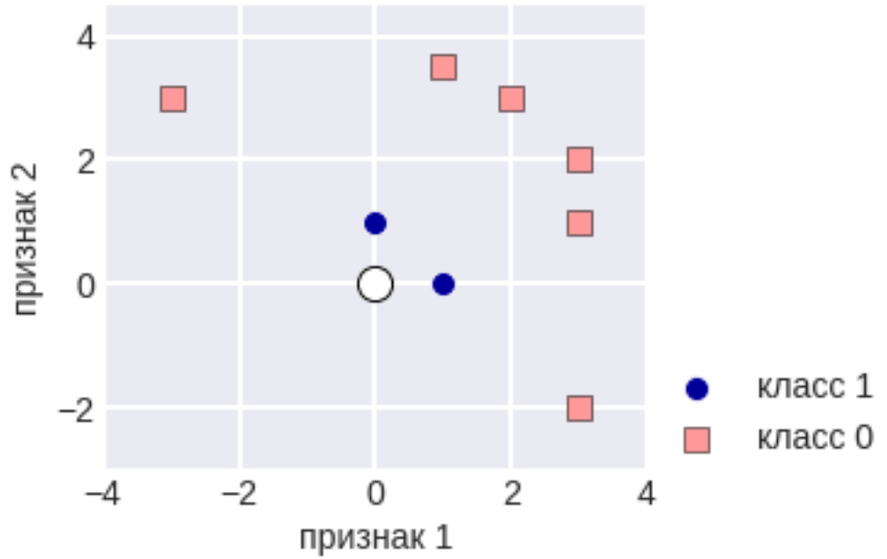
Как был получен второй график?

Сделали дисбаланс классов –
стало невыгодно делать большие k



а это если взять декартово произведение нескольких полумесяцев
(выборка сбалансированная)

Проблема



близкие соседи должны быть важнее

Весовые обобщения kNN

классика

$$\text{mode}(y_i \mid x_i \in N(x)) = \arg\max \sum_{t=1}^{k} I[y(x_t) = a]$$

обощение

$$\arg\max \sum_{t=1}^{k} w_t I[y(x_t) = a]$$

разные весовые схемы:

$$w_1 \geq w_2 \geq \ldots \geq w_k > 0$$

Весовые схемы

$$w_t = (k - t + 1)^{\delta}$$

$$k^{\delta} \geq (k-1)^{\delta} \geq \ldots \geq 1^{\delta} > 0$$

$$w_t = \frac{1}{t^{\delta}}$$

$$\frac{1}{1^{\delta}} \geq \frac{1}{2^{\delta}} \geq \ldots \geq \frac{1}{k^{\delta}} > 0$$

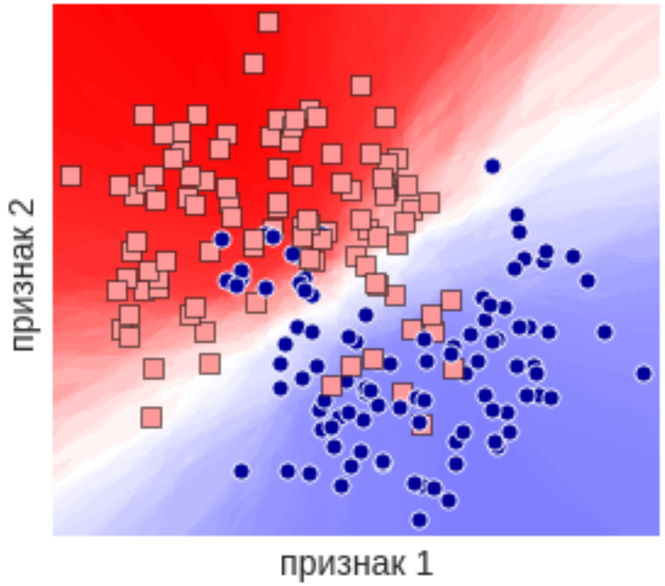
$$w_t = K\left(\frac{\rho(x, x_t)}{h(x)}\right)$$

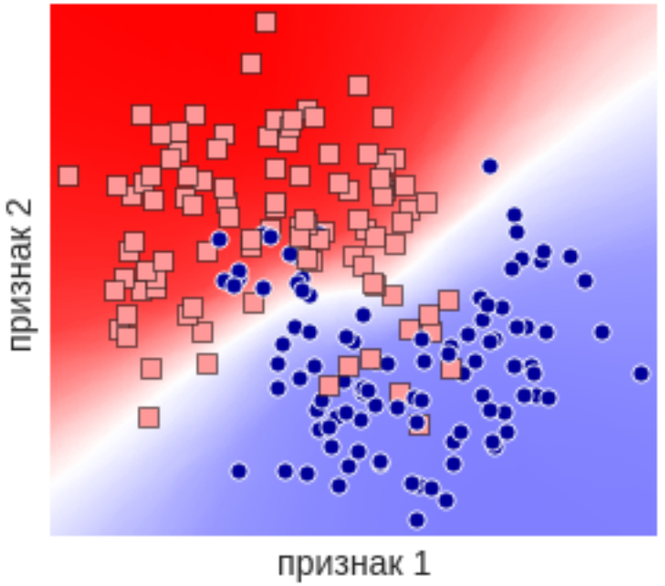
Последний способ хорош только на картинках…

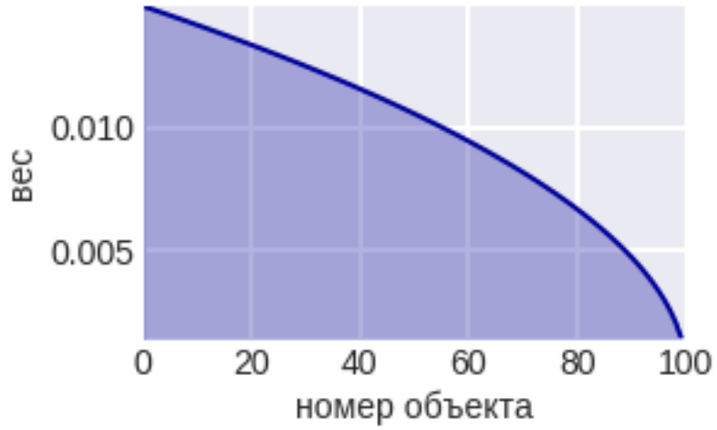
часто веса лучше отнормировать, чтобы сумма = 1

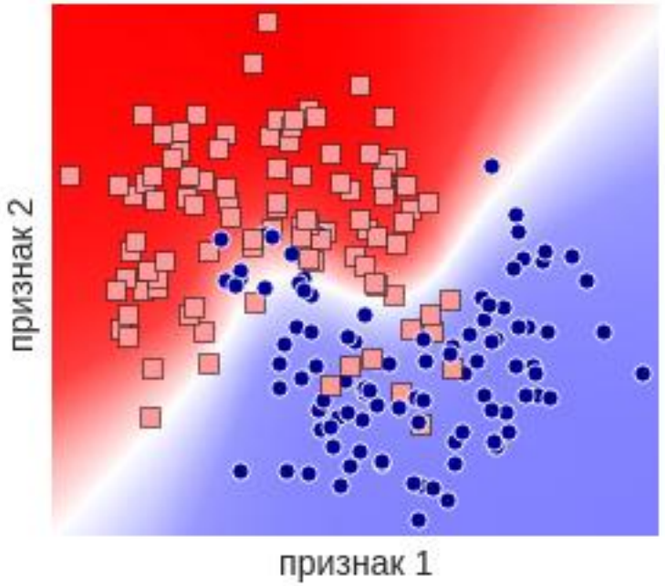
главное преимущество –
богатое пространство вероятности в задачах классификации!
можем различать степени принадлежности у разных объектов
потом будем подробно разбирать

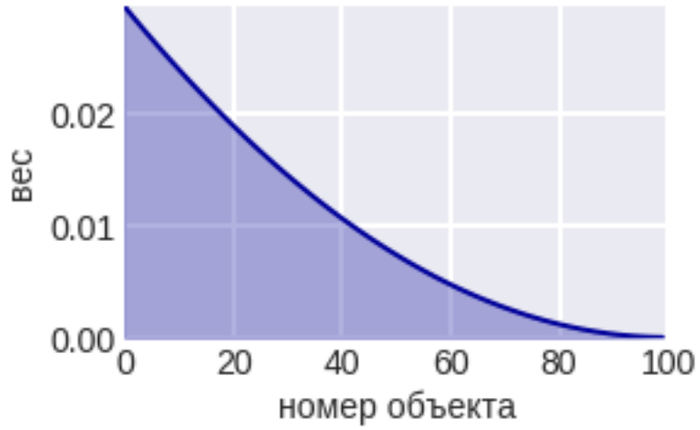
Весовые обобщения kNN



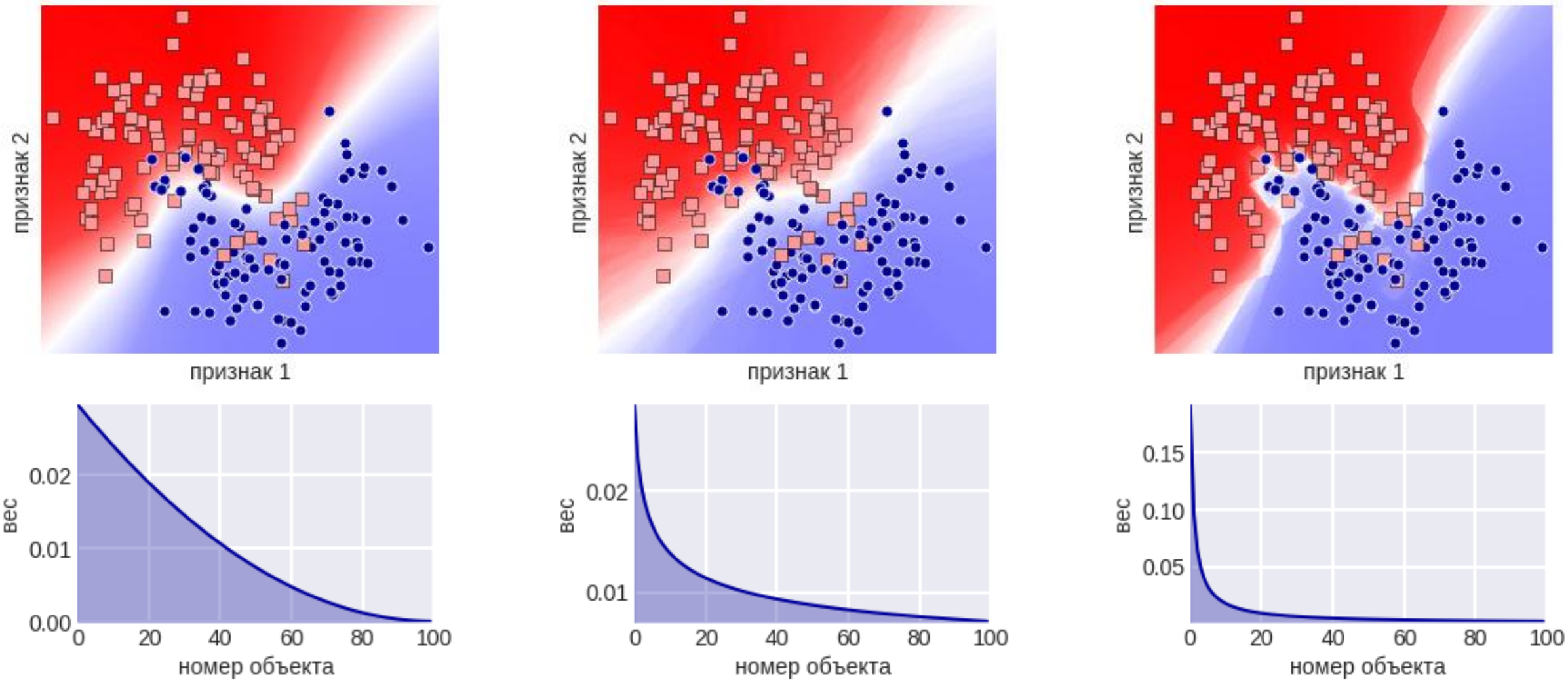








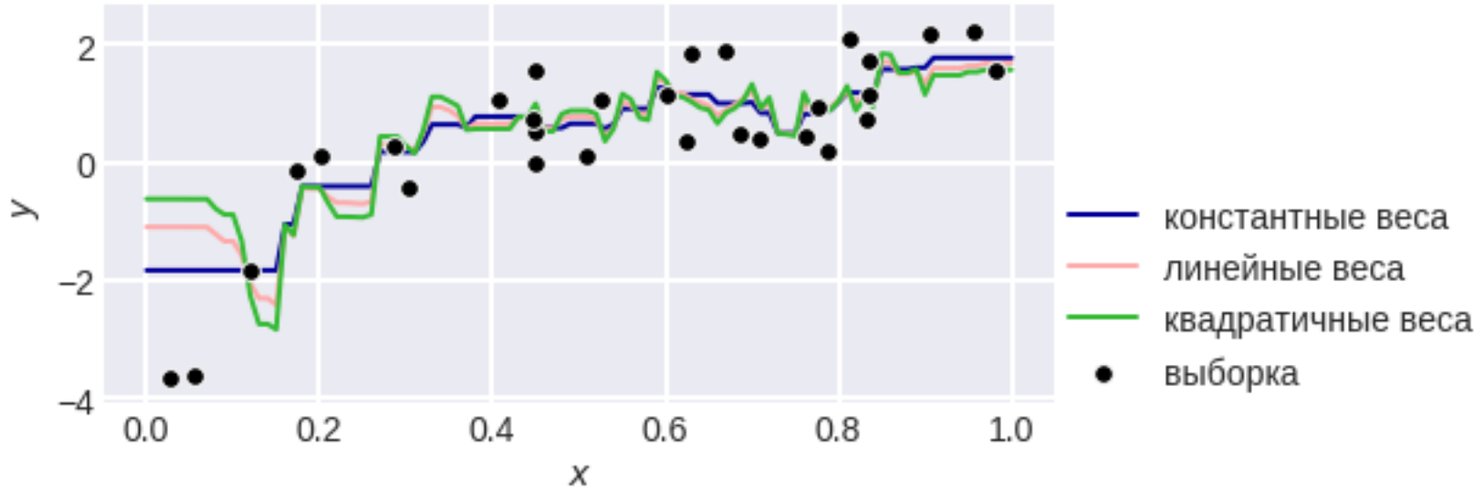
Весовые обобщения kNN



Весовые обобщения в регрессии

$$\frac{\sum_{t=1}^{k} w_t y(x_t)}{\sum_{t=1}^{k} w_t}$$

пример для 5NN



Эффект почти не заметен, дальше будет обобщение – регрессия Надарая-Ватсона

Метрики

Расстояние (метрика) на X – функция $\rho(x,z): X \times X \to \mathbb{R}$

1. $\rho(x,z) \geq 0$
2. $\rho(x,z) = 0 \Leftrightarrow x = z$ (без – полумерика/псевдометрика)
3. $\rho(x,z) = \rho(z,x)$
4. $\rho(x,z) + \rho(z,v) \geq \rho(x,v)$

- **Минковского L_p**
  - **Евклидова L_2**
  - **Манхэттенская L_1**
- **Махалонобиса**

- **Canberra distance**
- **Хэмминга**
- **косинусное**
- **расстояние Джаккарда**

- **DTW**
- **Левенштейна**

Различные метрики

Евклидова (L2)

$$\sqrt{\sum_{i=1}^{n}(x_i - z_i)^2}$$

Общий вариант – Минковского (L_p)

$$\left(\sum_{i=1}^{n}|x_i - z_i|^p\right)^{1/p}$$

Предельный случай – Чебышева (L_∞)

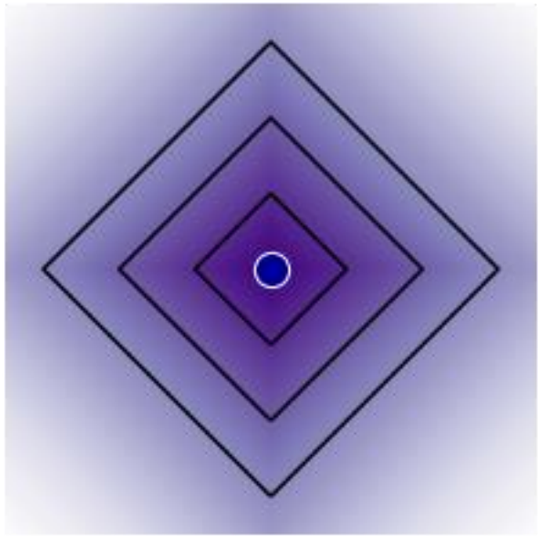
$$\left(\sum_{i=1}^{n}|x_i - z_i|^\infty\right)^{1/\infty} \sim \max_i |x_i - z_i|$$

Частный случай – Манхэттенская (L_1)

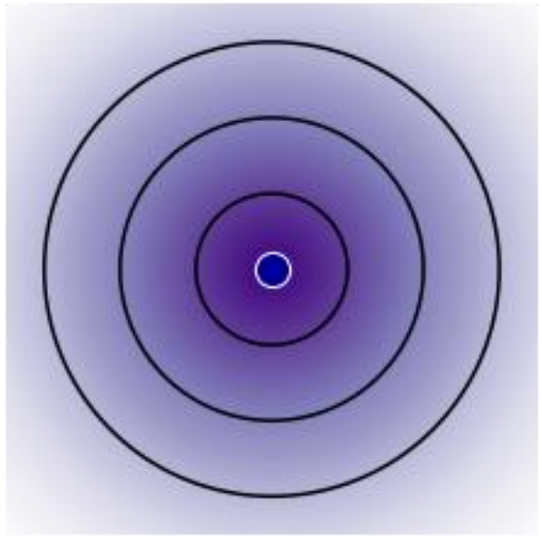
$$\sum_{i=1}^{n}|x_i - z_i|$$

Различные метрики

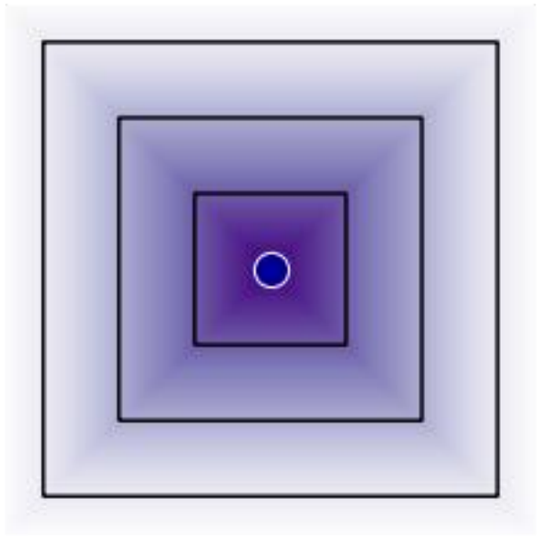
$$\left(| x_1 - z_1 |^p + | x_2 - z_2 |^p \right)^{1/p}$$



L_1



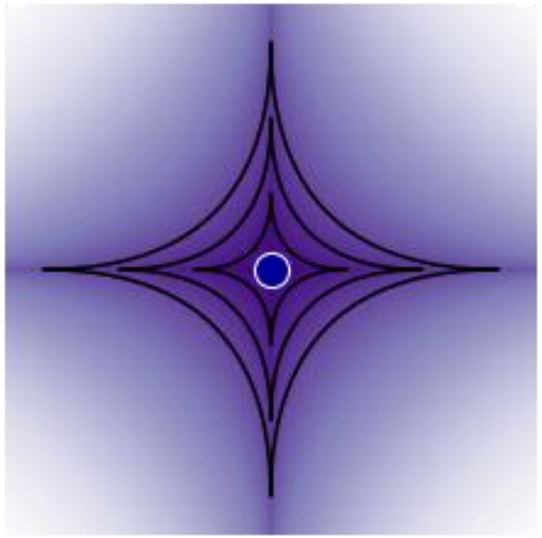
L_2



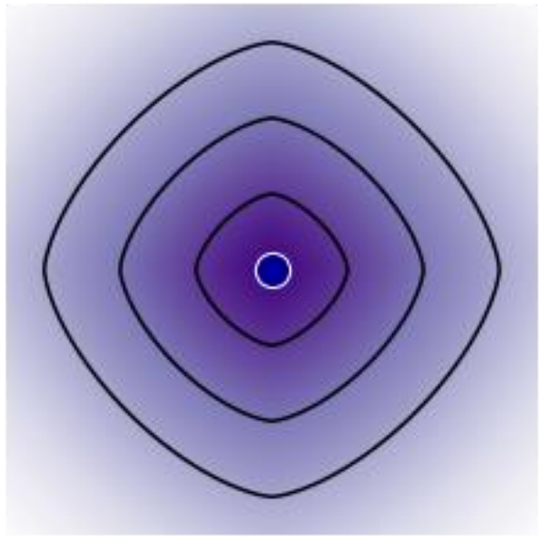
L_∞

Различные метрики

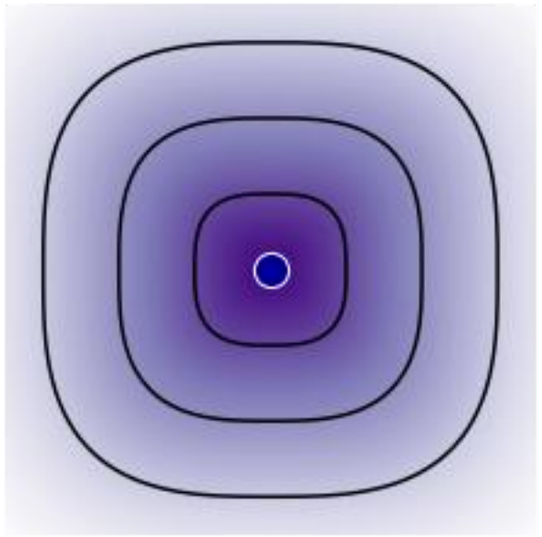
$$\left(| x_1 - z_1 |^p + | x_2 - z_2 |^p \right)^{1/p}$$



$L_{0.5}$



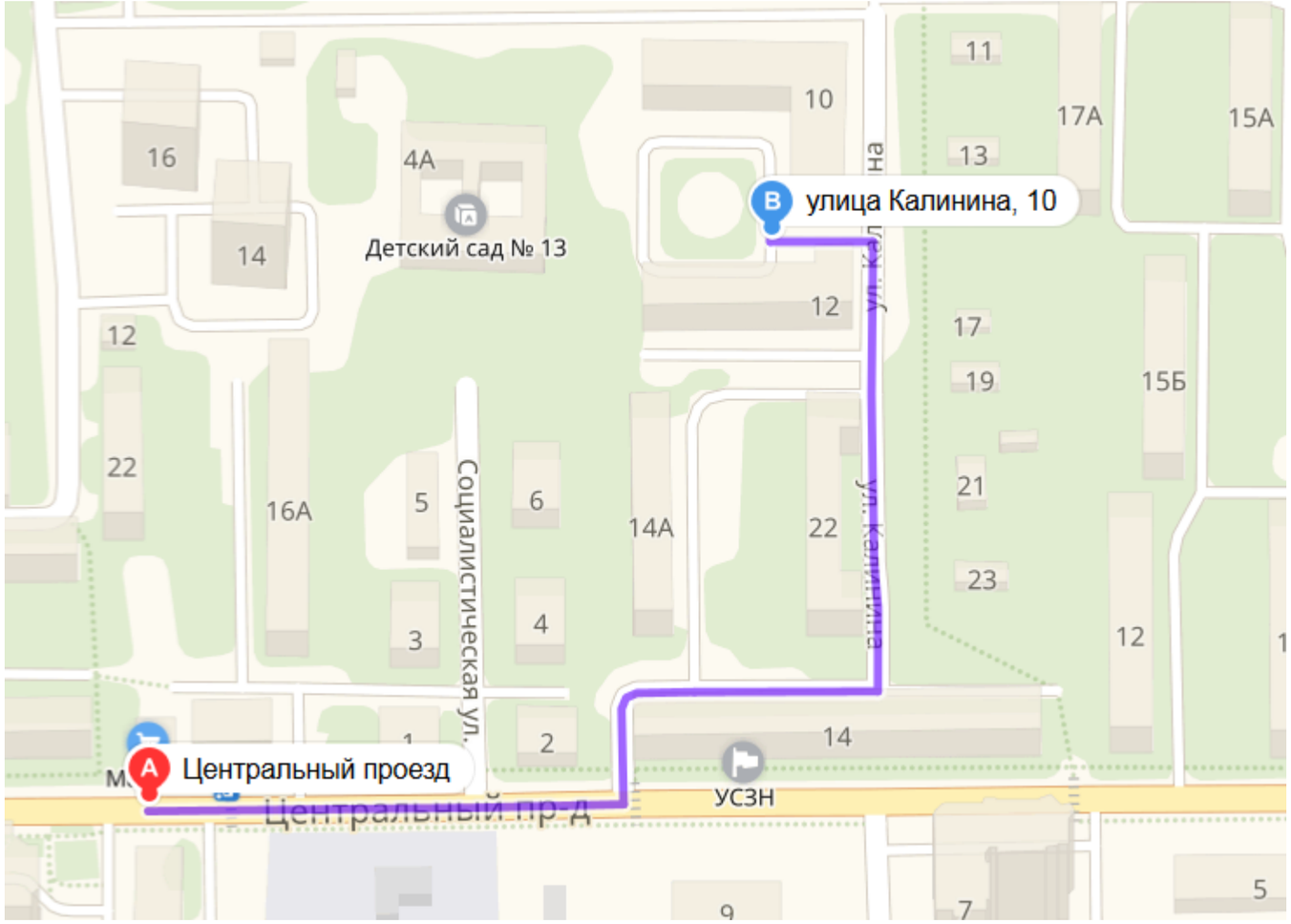
$L_{1.5}$



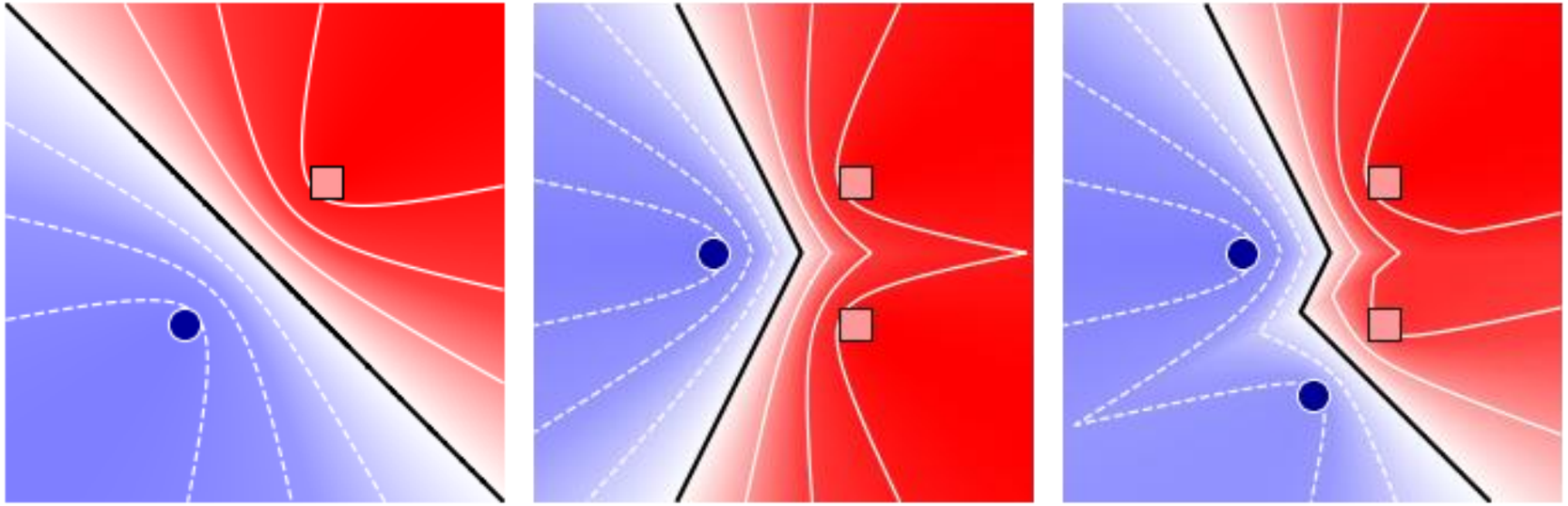
L_3

что такое L_0?

Различные метрики



Разделяющие поверхности L_2



Разделяющие поверхности L_1

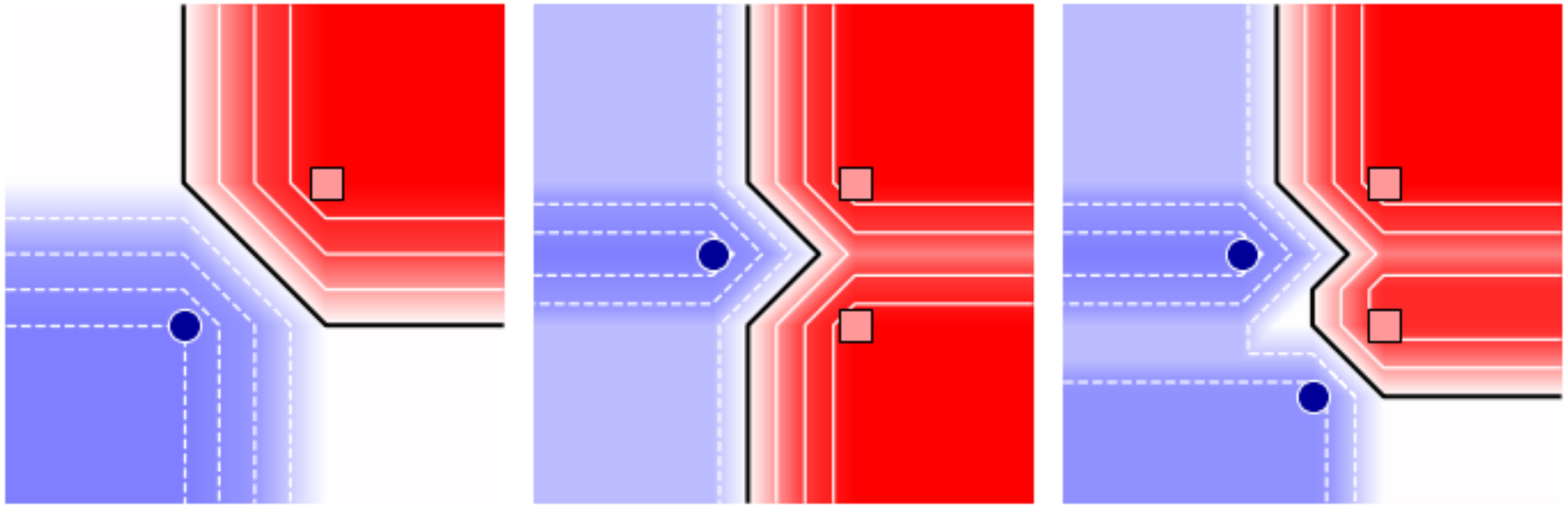
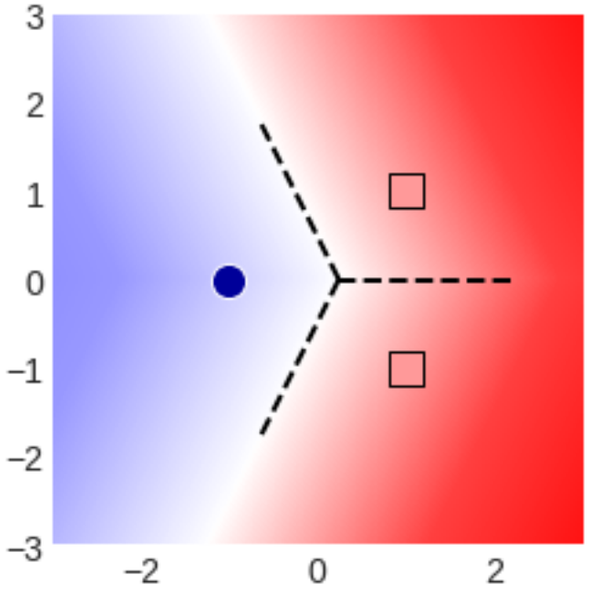
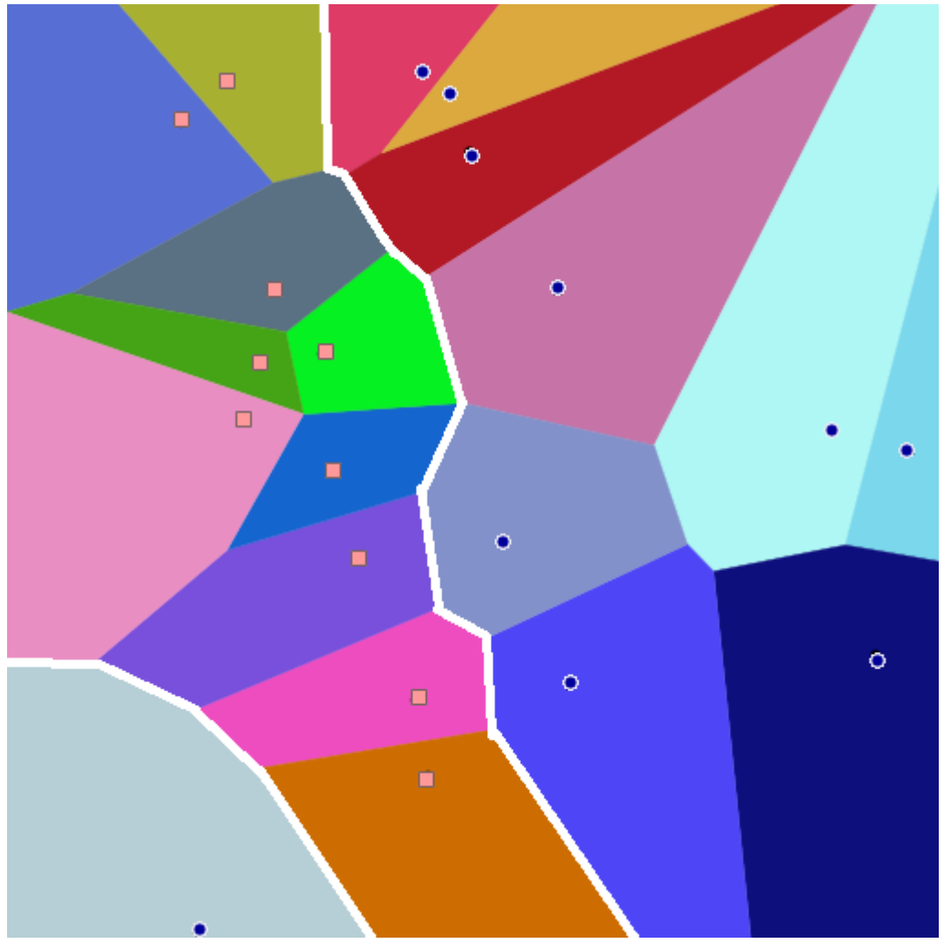


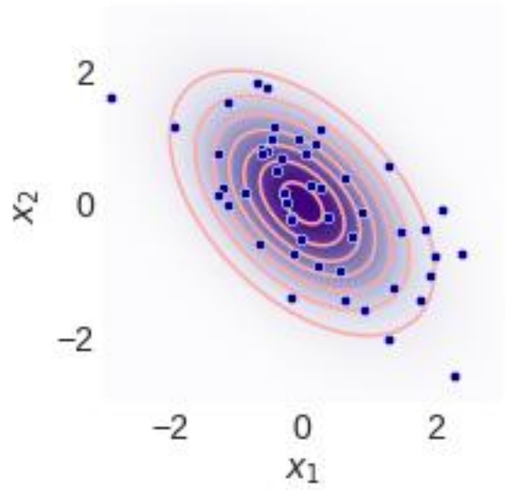
Диаграмма Вороного (Voronoi diagram)





https://en.wikipedia.org/wiki/Voronoi_diagram

Расстояние Махалонобиса (Mahalonobis distance)



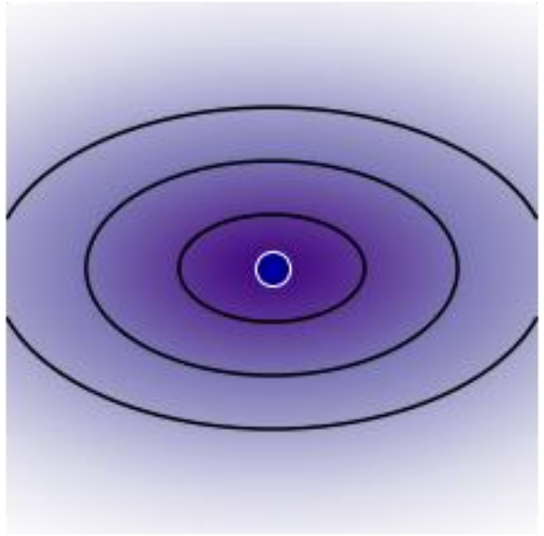
$$x \to \varphi(x) = \Sigma^{-1/2}(x - \mu)$$

стандартизует нормальные данные

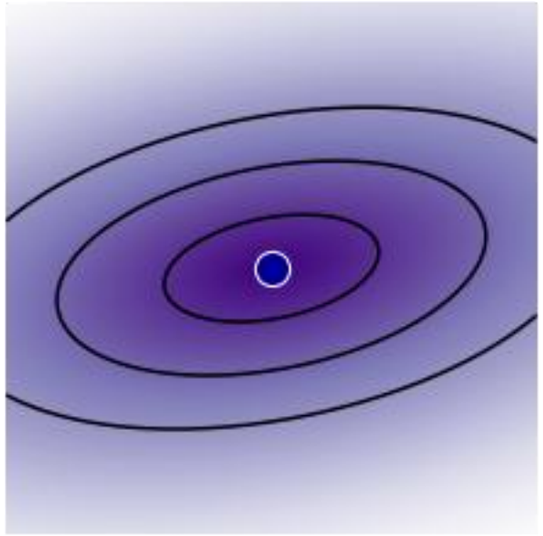
$$\text{norm}(\mu, \Sigma) \to \text{norm}(0, I)$$

$$\rho(x,z) = \rho_{L_2}(\varphi(x), \varphi(z)) = \sqrt{(\varphi(x) - \varphi(z))^{\text{T}}(\varphi(x) - \varphi(z))} = \sqrt{(x - z)^{\text{T}}\Sigma^{-1}(x - z)}$$

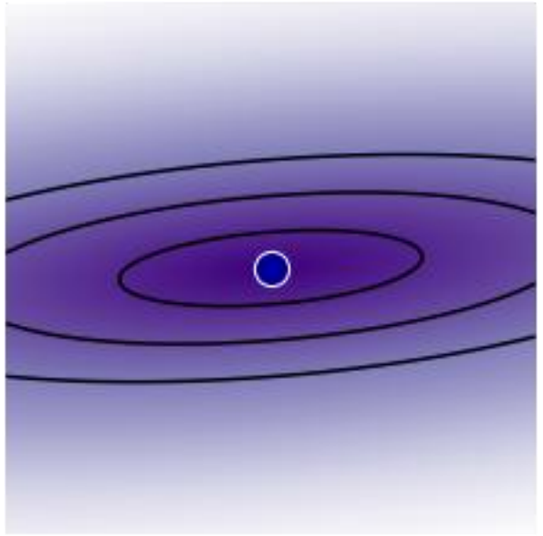
Расстояние Махалонобиса



$$\Sigma = \begin{bmatrix} 1.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$



$$\Sigma = \begin{bmatrix} 2 & 0.3 \\ 0.3 & 0.5 \end{bmatrix}$$



$$\Sigma = \begin{bmatrix} 4 & 0.3 \\ 0.3 & 0.25 \end{bmatrix}$$

Расстояния

Canberra distance

https://en.wikipedia.org/wiki/Canberra_distance

$$\frac{1}{n}\sum_{i=1}^{n}\frac{|x_i - z_i|}{|x_i| + |z_i|}$$

Хэмминга

$$\sum_{i=1}^{n} I[x_i \neq z_i]$$

Косинусная мера сходство

$$\cos(x, z) = \frac{x^{\mathrm{T}} z}{\|x\| \cdot \|z\|}$$

если работать с нормированными векторами,
достаточно рассматривать скалярное произведение

Расстояния

Расстояние Джаккарда (на множествах)

$$1-\frac{|X\cap Z|}{|X\cup Z|}$$

Расстояние на множествах индуцирует расстояние на бинарных векторах

$$1-\frac{x^{\mathrm{T}}z}{x^{\mathrm{T}}x+z^{\mathrm{T}}z-x^{\mathrm{T}}z}=\frac{x^{\mathrm{T}}x+z^{\mathrm{T}}z-2x^{\mathrm{T}}z}{x^{\mathrm{T}}x+z^{\mathrm{T}}z-x^{\mathrm{T}}z}$$

тут есть много разных вариантов…

Проблема выбора метрики

- **зависимость от масштаба**

нормировка признаков

однородные признаки

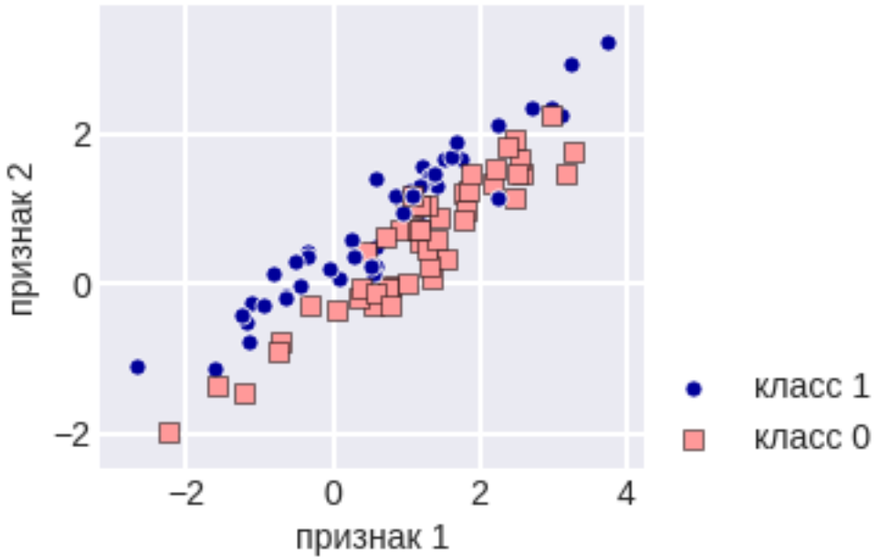
смесь метрик

- **можно выбирать не метрику, а близость**

пример: косинусная мера сходства

- **часто выбор функции расстояния, как ни странно, довольно прост…**

Обучение метрики: Metric Learning



Признаковое пространство явно нуждается в корректировке

$$x \to \varphi(x) = \Sigma^{-1/2}(x - \mu)$$

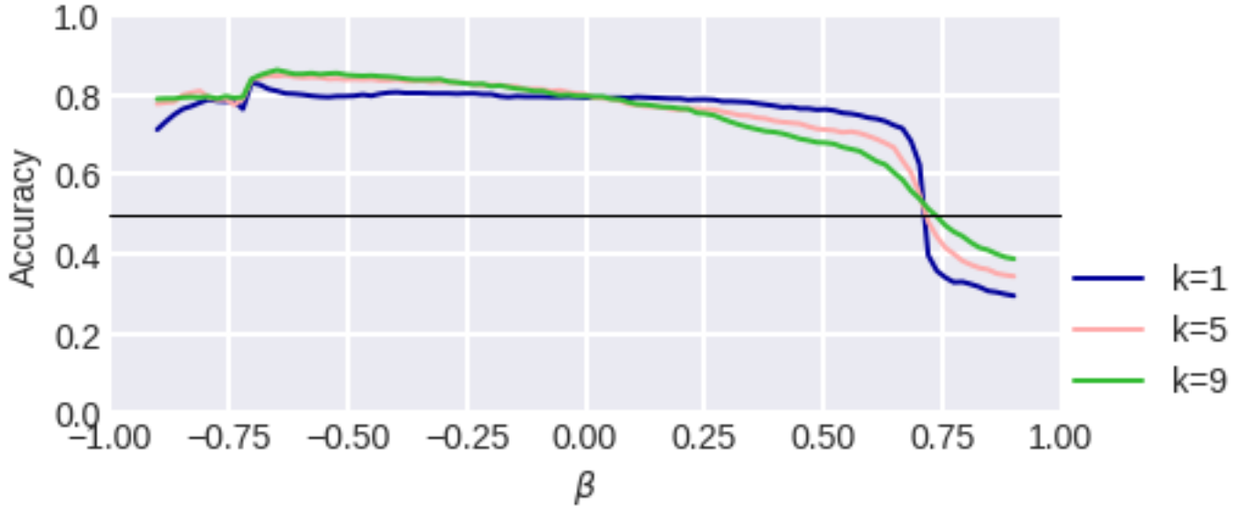
$$\rho(x, z) = \sqrt{(x - z)^{\mathrm{T}} \Sigma^{-1} (x - z)}$$

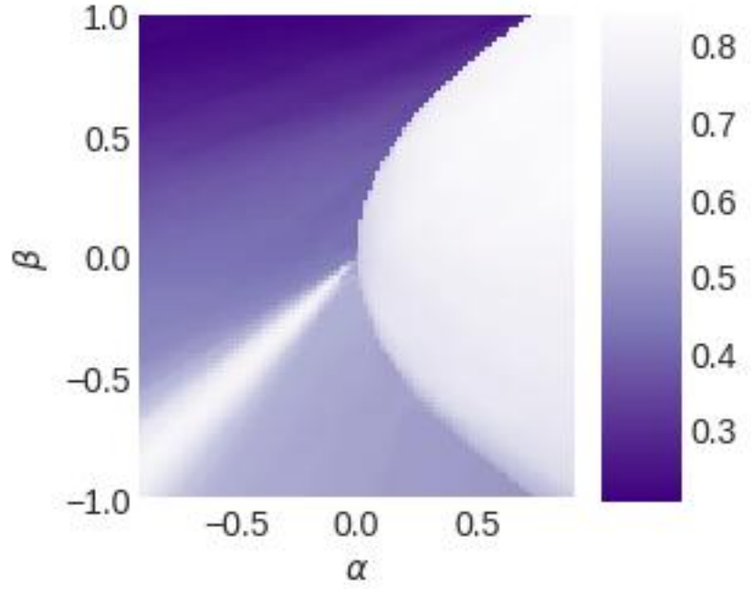
Можно считать, что Σ^{-1} – матрица параметров

Оптимизировать

- **Качество kNN**
- **Расстояния до своих / чужих**

Обучение метрики: Metric Learning





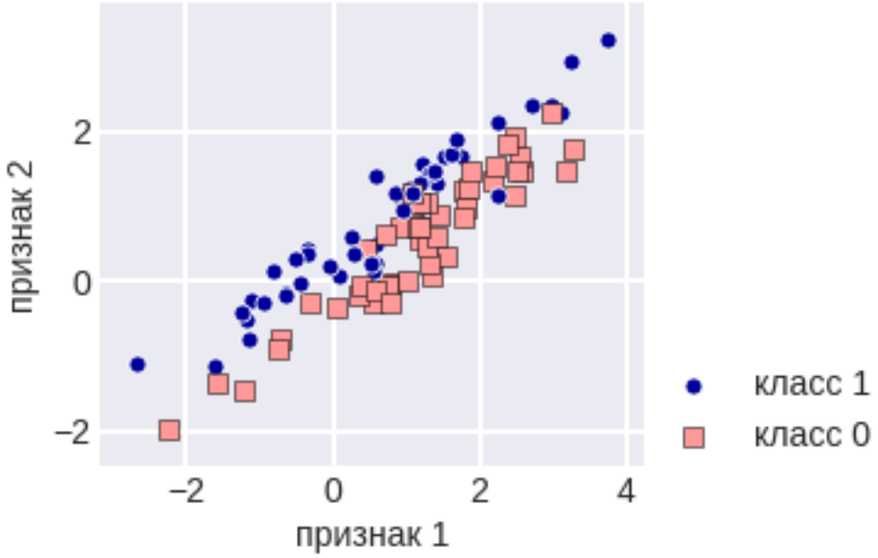

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

n_p, small_eps = 11, 0.000001
Q = np.zeros([n_p, n_p])
LS1 = np.linspace(-0.9, 0.9, n_p)
LS2 = np.linspace(-1, 1, n_p)

for i, b in enumerate(LS1):
    for j, a in enumerate(LS2):
        A = np.array([[a, b], [b, 1]])
        if np.abs(np.linalg.det(A)) < small_eps:
            A = A + small_eps * np.eye(2)
        model = KNeighborsClassifier(n_neighbors=5,  metric='mahalanobis',
                                     metric_params={'V': np.linalg.inv(A)})
        model.fit(X, y)
                    Q[i, j] = accuracy_score(y2, model.predict(X2))
```

$$\Sigma = \begin{bmatrix} \alpha & \beta \\ \beta & 1 \end{bmatrix}$$

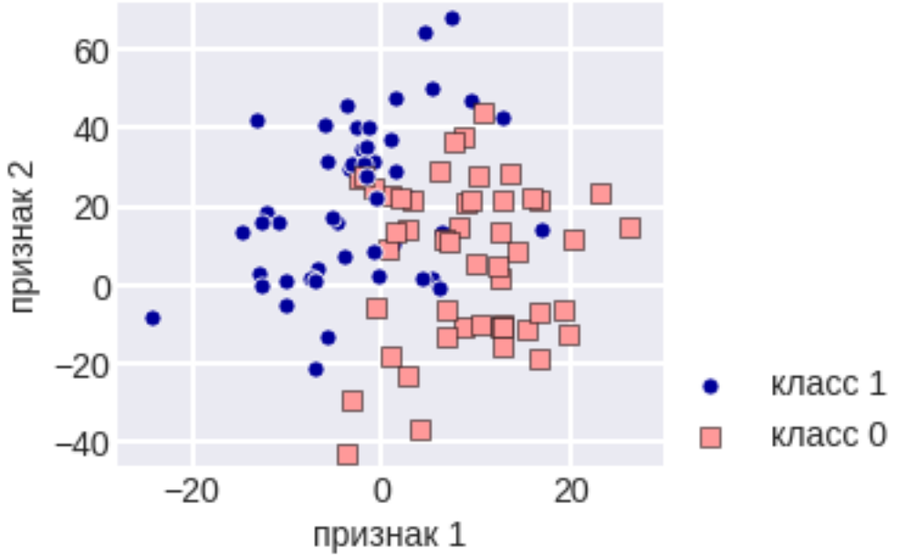
здесь можно перебрать параметры,
а вообще – методы оптимизации
почему такой рис?

Обучение метрики: Metric Learning

NeighborhoodComponentsAnalysis – улучшение качества 1NN с евклидовым расстоянием



до обучения



после обучения

```
from sklearn.neighbors import NeighborhoodComponentsAnalysis
nca = NeighborhoodComponentsAnalysis(max_iter=30, random_state=10)
nca = nca.fit(X, y)
X_embedded = nca.transform(X)
```

Обучение метрики: Metric Learning

NeighborhoodComponentsAnalysis – математика

ищем преобразование

$$x \to \varphi(x) = L(x - \mu)$$

$$p_{ij} = \frac{\exp(-\| Lx_i - Lx_j \|^2)}{\sum_{t \neq i} \exp(-\| Lx_i - Lx_t \|^2)}$$

если $x_i \in K_s$**, то**

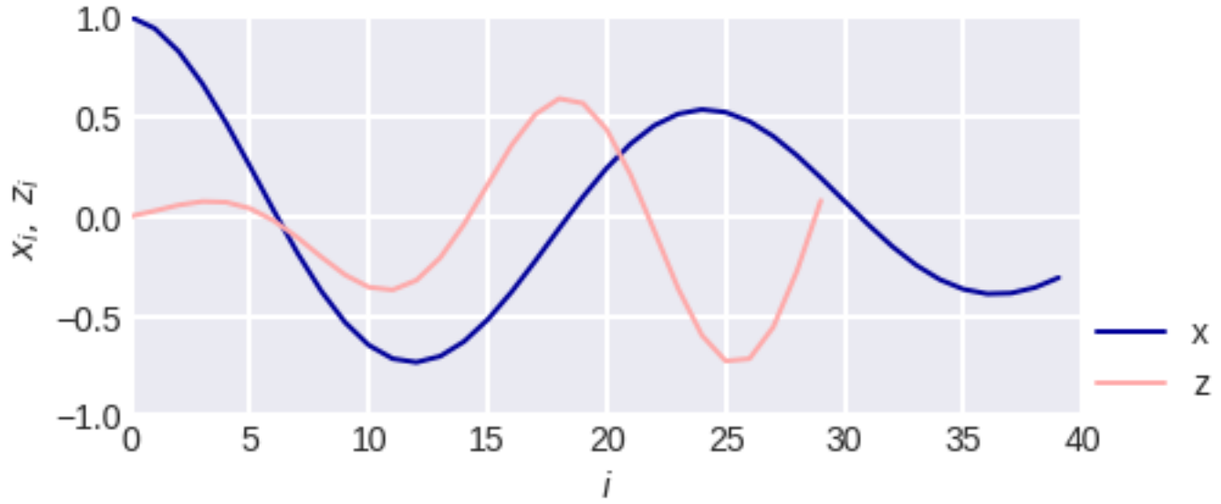
$$p_i = \sum_{x_j \in K_s} p_{ij}$$

(сумма вероятностей по объектам этого же класса)

задача оптимизации

$$\sum_{i=1}^{m} p_i \to \max$$

Метрики на временных рядах



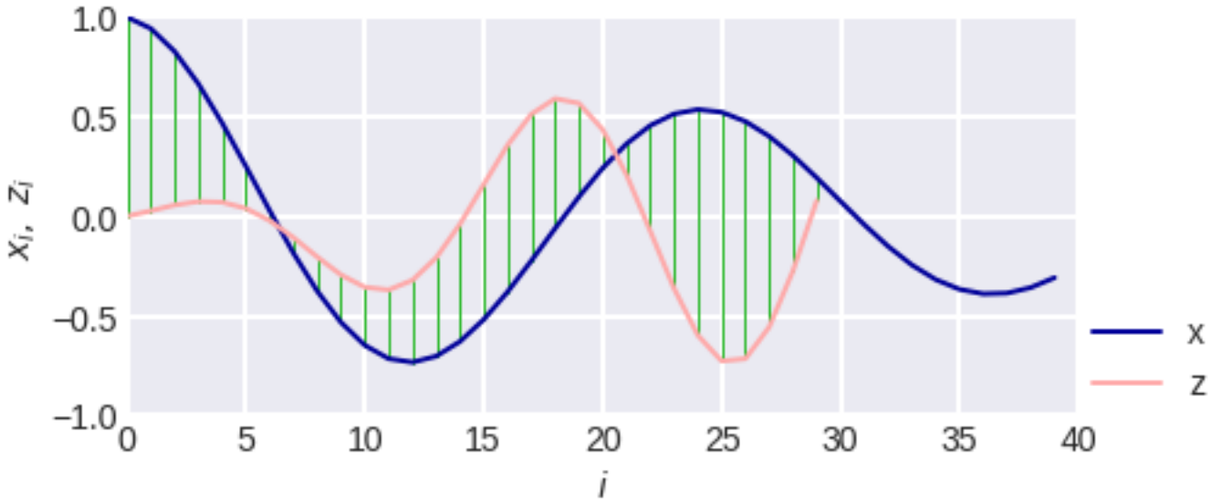
как ввести расстояние?

Ряды могут быть разной длины…

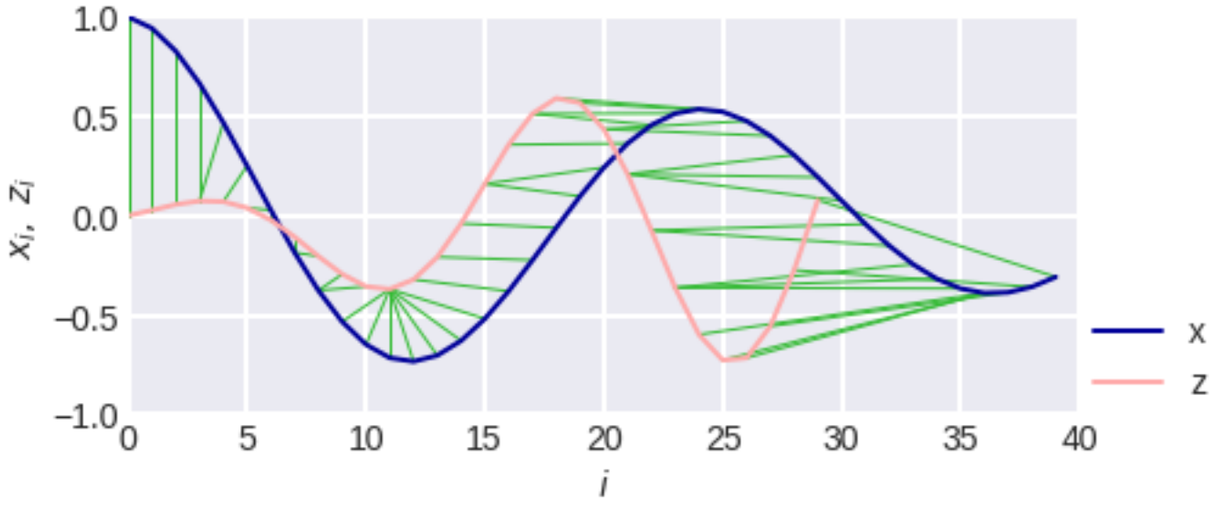
Как оценить похожесть формы?

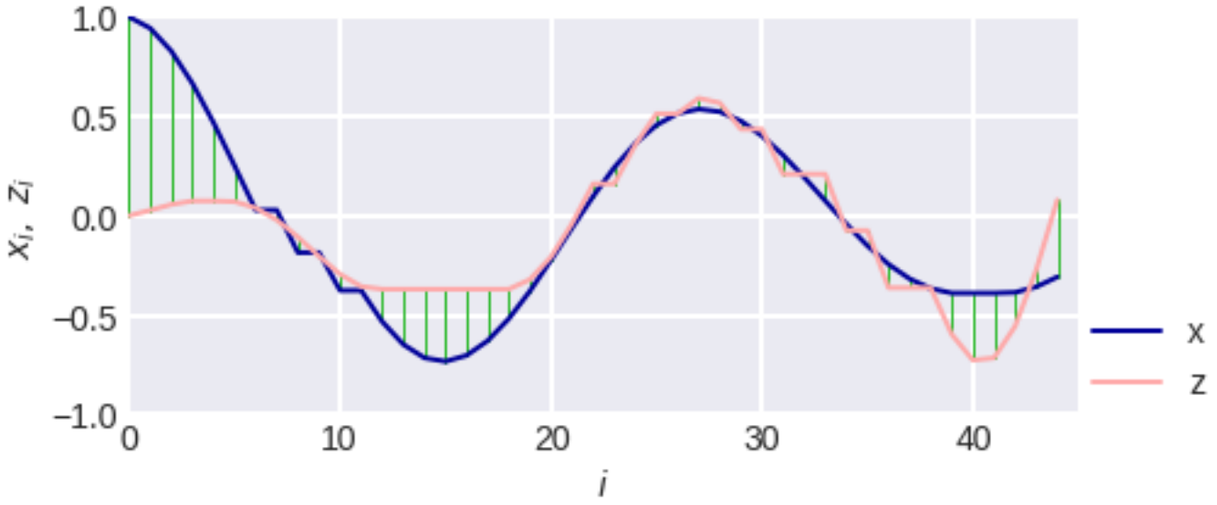
Метрики на временных рядах

Евклидово расстояние



DTW = Dynamic time warping





Метрики на временных рядах

Пусть есть векторы (временные ряды)

$$x = (x_1, \ldots, x_m)$$

$$z = (z_1, \ldots, z_n)$$

срез –

$$x[:i] = (x_1, \ldots, x_i)$$

рекурсивное определение –

$$\mathrm{DTW}(x[:i], z[:j]) = \rho(x_i, z_j) + \min \begin{cases} \mathrm{DTW}(x[:i-1], z[:j-1]) \\ \mathrm{DTW}(x[:i-1], z[:j]) \\ \mathrm{DTW}(x[:i], z[:j-1]) \end{cases}$$

начальные условия рекурсивного определения

$$\mathrm{DTW}(x[:0], z[:0]) = 0$$

$$\mathrm{DTW}(x[:i], z[:0]) = \mathrm{DTW}(x[:0], z[:i]) = \infty$$

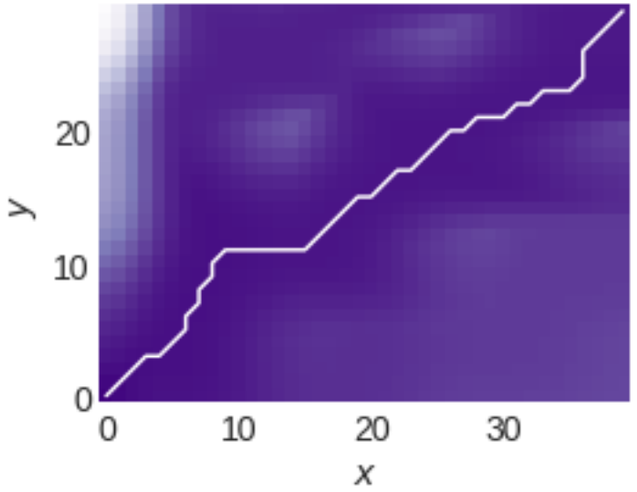
при $i > 0$

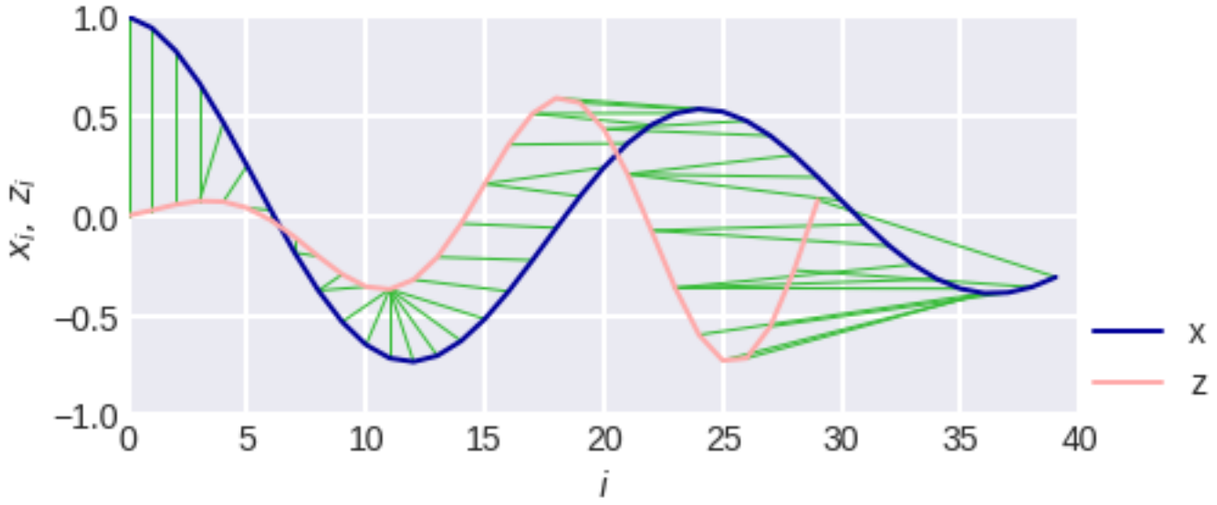
здесь нет нормировки… хотя может быть

Метрики на временных рядах

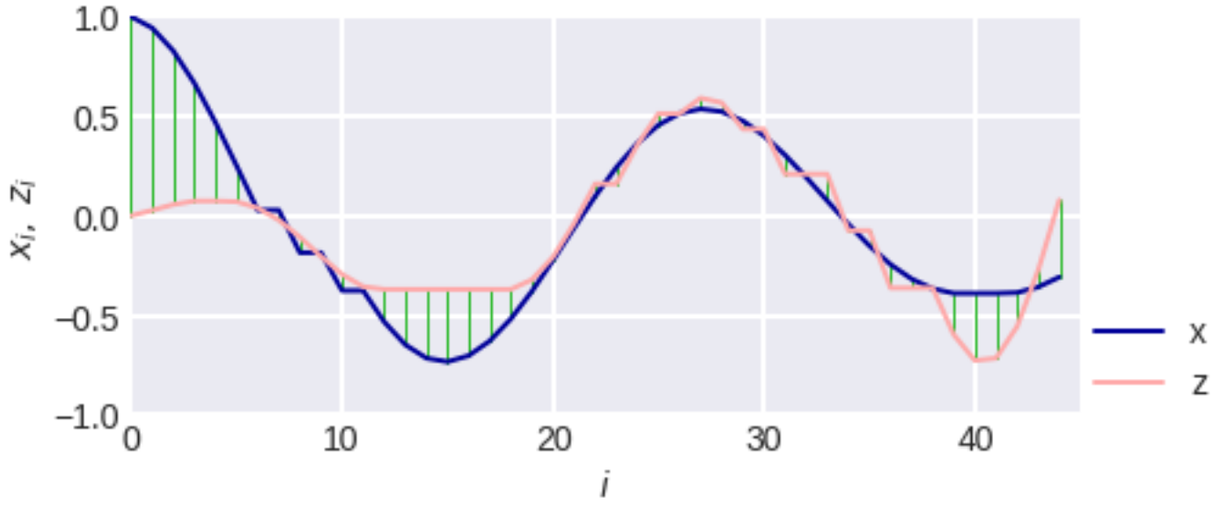
Для адекватности и быстроты часто

$$|i-j|>r \Rightarrow \mathrm{DTW}(x[:i], z[:j]) = +\infty$$





оптимальные соответствия точек



если разжать ряды соотв. образом

Расстояние Левенштейна

«Edit distance»

Расстояние между строками
Вводим элементарные операции правки:

- **вставить букву**
- **удалить букву**
- **заменить букву**

расстояние – минимальное число операций,
с помощью которых их одной строки можно получить другую

использование: исправление опечаток

Расстояние Левенштейна

определение аналогично DTW

(можно для каждой операции – свой штраф)

$$\mathrm{D}(x[:i], z[:j]) = \min \begin{cases} \mathrm{sub}(x_i, z_j) + \mathrm{D}(x[:i-1], z[:j-1]) \\ \mathrm{add}(x_i) + \mathrm{D}(x[:i-1], z[:j]) \\ \mathrm{add}(z_j) + \mathrm{D}(x[:i], z[:j-1]) \end{cases}$$

$$\mathrm{sub}(x_i, z_j) = \begin{cases} 0, & x_i = z_j, \\ 1, & x_i \neq z_j, \end{cases}$$

Приложения метрического подхода: нечёткий матчинг таблиц

пример АМ

матчинг таблиц разных аудиторий для рекламного агенства

Приложения метрического подхода: Ленкор

«VideoLectures.Net Recommender System Challenge»
(ECML/PKDD Discovery Challenge 2011)
– написать рекомендательную систему в режиме холодного старта

Описание лекции

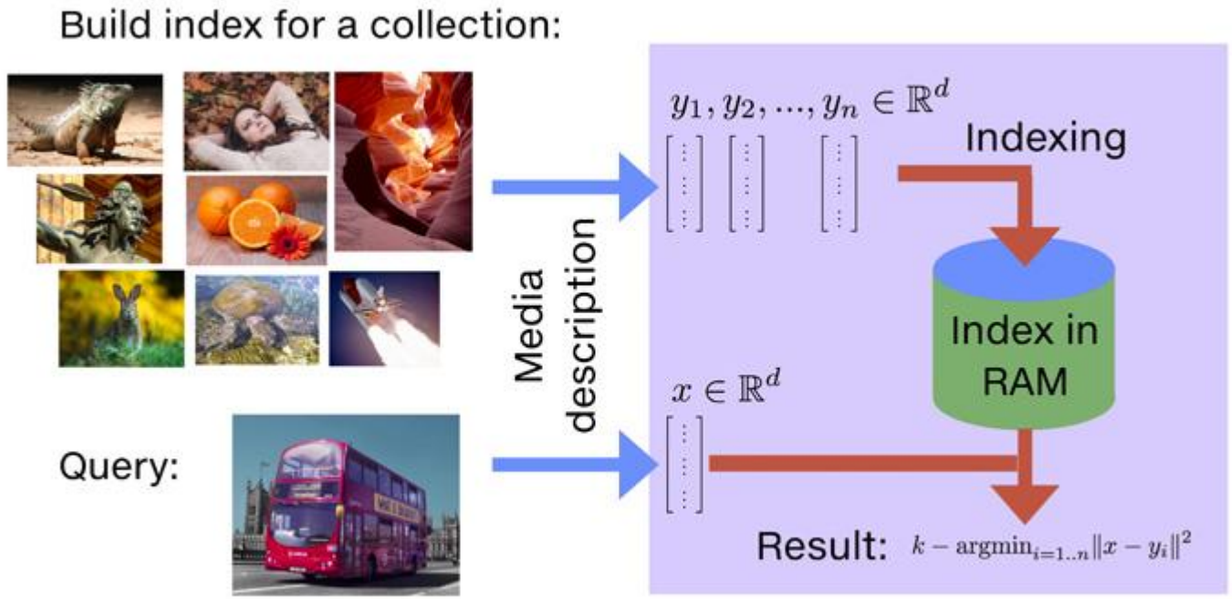
101, 'Lecture', 'eng', 'biology', '2008-12-04', '2009-02-12', 'Implementing a common framework on business', 'Professor Rudolf Smith', …

$$\rho(\text{Lecture}_1, \text{Lecture}_2) =$$
$$= c_1 \cdot \rho_1(\text{Author}_1, \text{Author}_2) + c_2 \cdot \rho_2(\text{Title}_1, \text{Title}_2) + \ldots + c_r \cdot \rho_r(\text{Subject}_1, \text{Subject}_2)$$

метрики можно параметризовать и настраивать параметры
«хитрый весовой учёт близости» – см. совместные просмотры

Дьяконов А.Г. Алгоритмы для рекомендательной системы: технология LENKOR // Бизнес-Информатика, 2012, №1(19), С. 32–39.

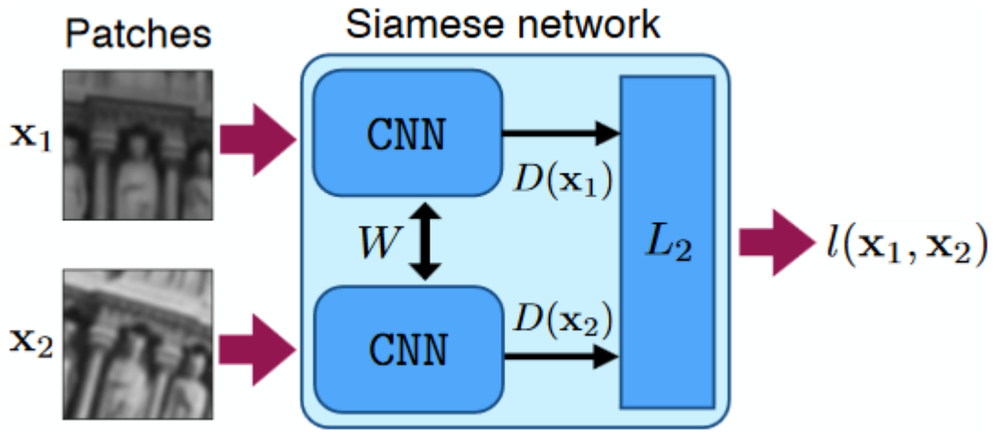
Приложения метрического подхода: поиск похожих объектов



звука, изображения, видео, …

https://code.fb.com/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

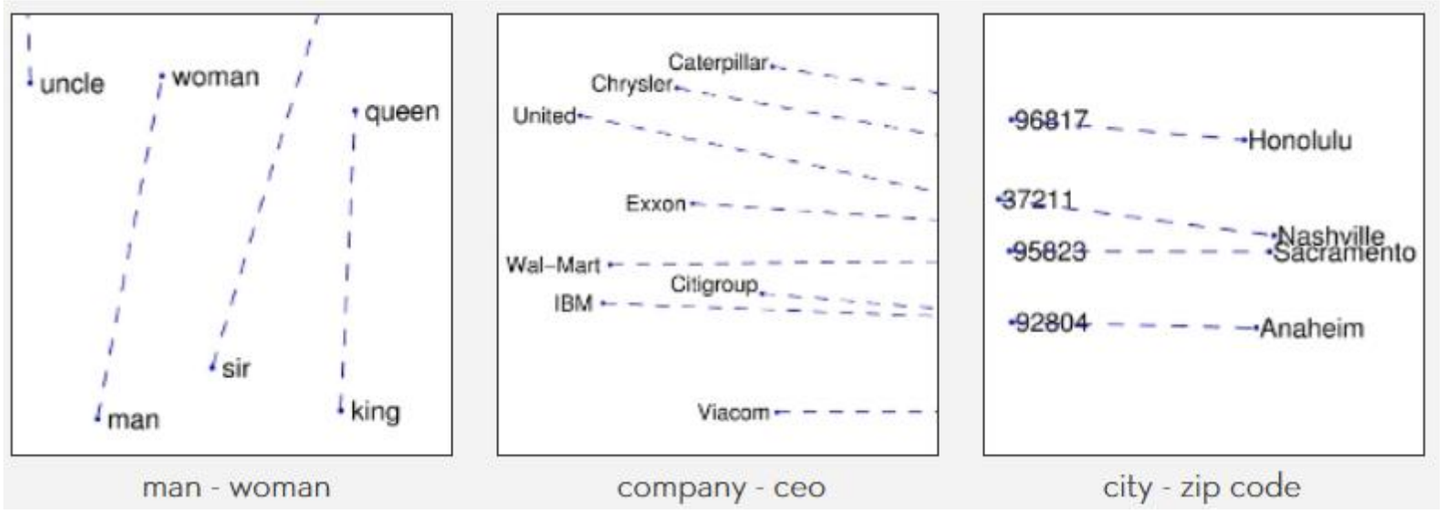
Приложения метрического подхода: Сиамские сети



Discriminative Learning of Deep Convolutional Feature Point Descriptors

[Simo-Serra et al., 2015 http://icwww.epfl.ch/~trulls/pdf/iccv-2015-deepdesc.pdf]

Приложения метрического подхода: GLOVE word vectors



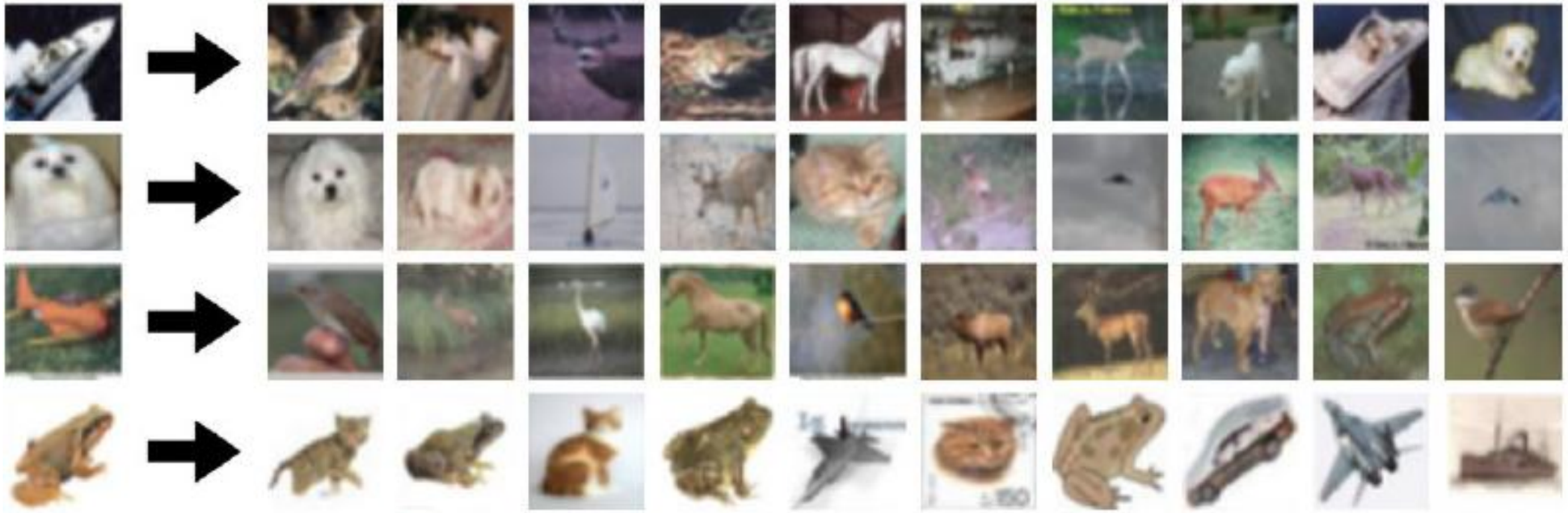
https://nlp.stanford.edu/projects/glove/

Приложения метрического подхода: SIFT image descriptors

TEXMEX

http://corpus-texmex.irisa.fr/

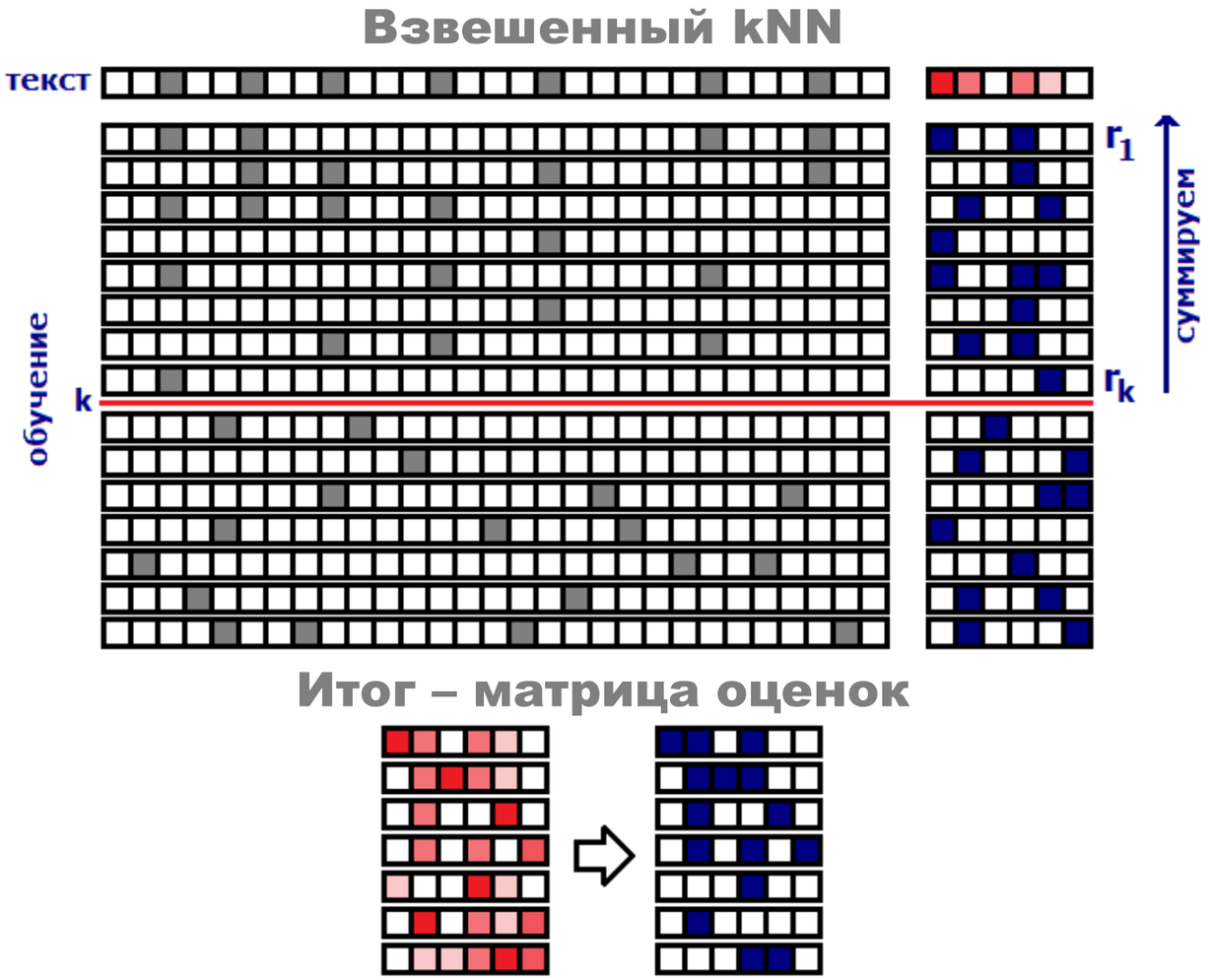
Приложения метрического подхода: простота интерпретаций



http://cs231n.stanford.edu/2017/syllabus.html

Приложения метрического подхода: классификация текстов

задача «Large Scale Hierarchical Text Classification»



https://www.kaggle.com/c/lshtc

Приложения метрического подхода: классификация текстов

задача «Large Scale Hierarchical Text Classification»

вычисление центроидов

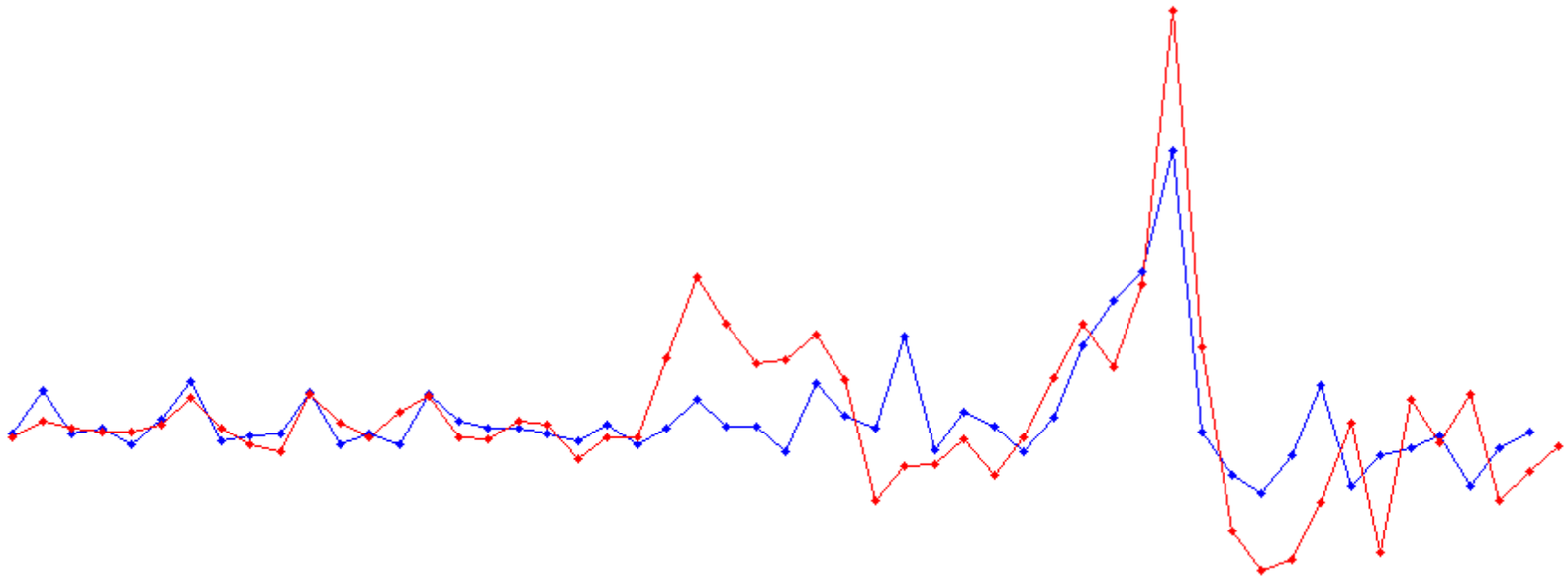
Приложения метрического подхода: прогнозирование рядов

$$\tilde{f} = (f_1, \ldots, f_n) \rightarrow \tilde{g} = (f_1, \ldots, f_n, f_{n+1}, \ldots, f_{n+t})$$

$$\| A(f_{k-l+1}, \ldots, f_k) - (f_{n-l+1}, \ldots, f_n) \| \rightarrow \min_{k,\, \tilde{a}}$$

$$A(x_1, \ldots, x_l) = (a_1 x_1 + a_2, \ldots, a_1 x_l + a_2)$$

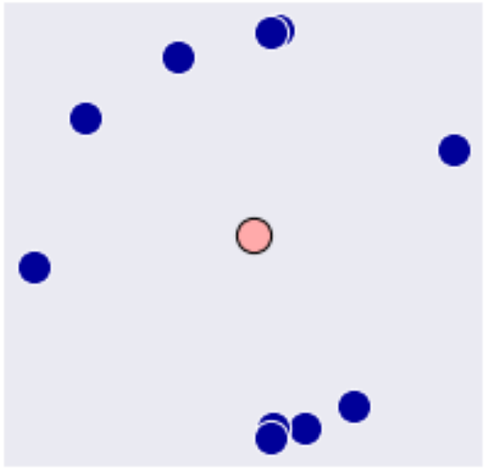
$$\sum_k c_k A(f_{k+1}, \ldots, f_{k+l}), \sum_k c_k = 1$$



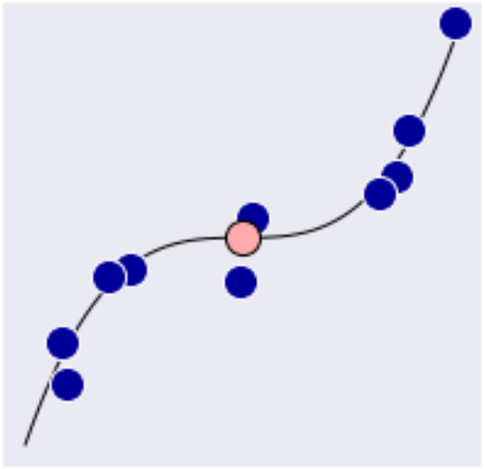
http://www.neural-forecasting-competition.com/NN5/results.htm

Проблема проклятия размерности

в пространствах большой размерности все объекты примерно на одном расстоянии



но, к счастью, на реальных данных…



есть внутренняя (intrinsic) размерность,
все объекты лежат около низкоразмерного многообразия
(low-dimensional manifold)

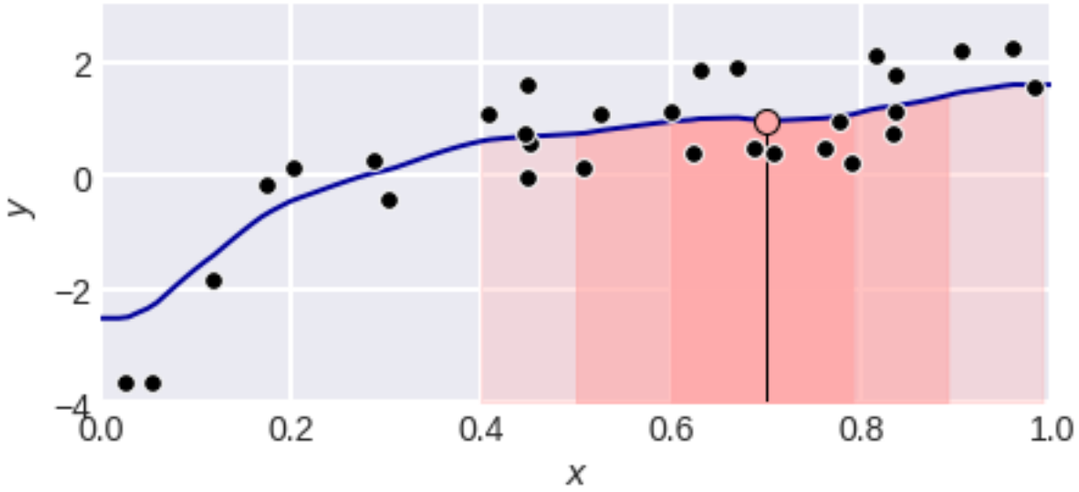
Метрические алгоритмы

+ не требуется признаковых описаний
(достаточно уметь измерять расстояния / близости)
+ легко реализуемы
+ интерпретируемость
+ нет обучения
+ мало гиперпараметров (хотя… есть метрика)
+ можно учитывать контекст (с помощью метрики)

– медленная классификация (зависит от объёма обучения)
– требуется хранение всей обучающей выборки
– требует подбора метрики (нормировки признаков)

Считается, что в пространствах гигантских размерностей стандартные метрики неадекватны (проклятие размерности),
но в реальности расположение объектов неслучайно – есть геометрия!!!

Регрессия Надарая-Ватсона (Nadaraya-Watson regression, 1964)



ответ – взвешенное усреднение целевых значений

$$a(x) = \frac{w_1(x)y_1 + \ldots + w_m(x)y_m}{w_1(x) + \ldots + w_m(x)}$$

Регрессия Надарая-Ватсона (Nadaraya-Watson regression)

$$a(x) = \frac{w_1(x)y_1 + \ldots + w_m(x)y_m}{w_1(x) + \ldots + w_m(x)}$$

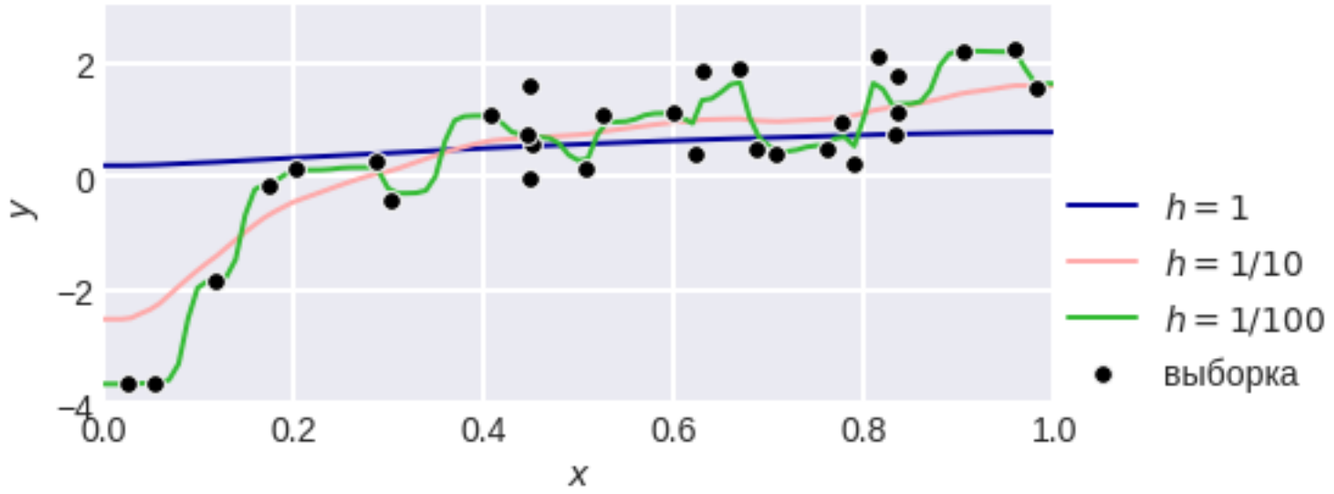
Смысл весов – чем ближе объект обучения, тем скорее ответ похож на его метку

$$w_i(x) = K\left(\frac{\rho(x, x_i)}{h}\right)$$

Ядро с шириной h.

Здесь также как выше… (про функции ядра)

Регрессия Надарая-Ватсона (Nadaraya-Watson regression)



Регрессия Надарая-Ватсона (Nadaraya-Watson regression)

Смысл:

ответ алгоритма – решение оптимизационной задачи

$$\sum_{i=1}^{m} w_i(x)(a - y(x_i))^2 \to \min_a$$

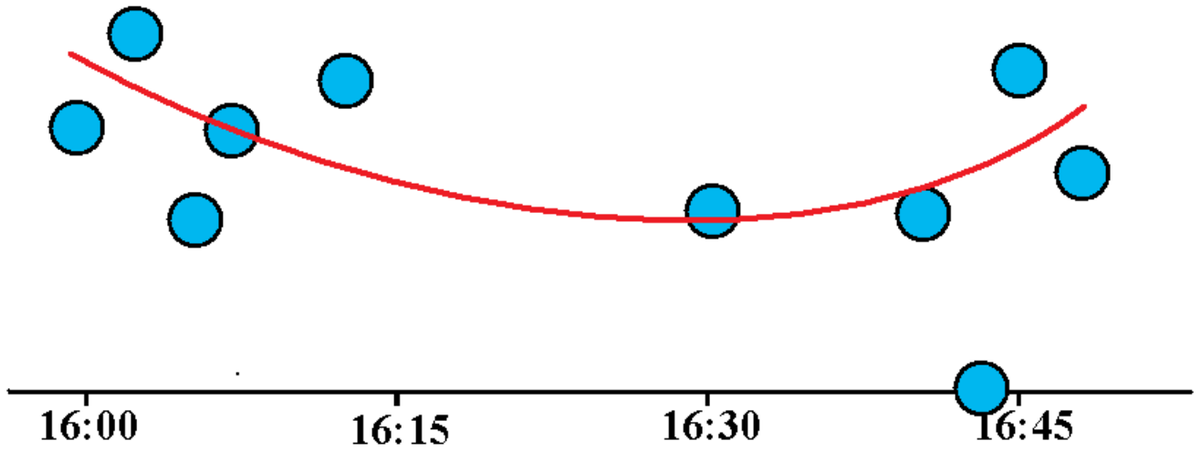
+ хорошее решение задачи сглаживания

– не решает задачи экстраполяции

Приложения регрессии Надарая-Ватсона

1. Сглаживание сигналов

2. «Многомерные» усреднения



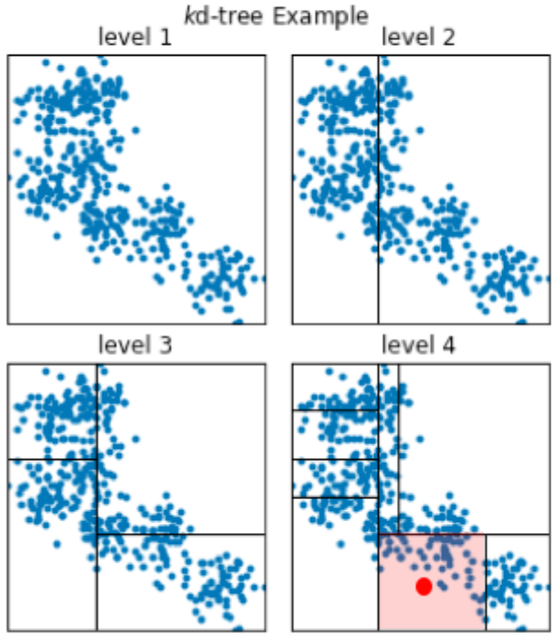
Эффективность

быстрые точные и приближённые методы поиска соседей

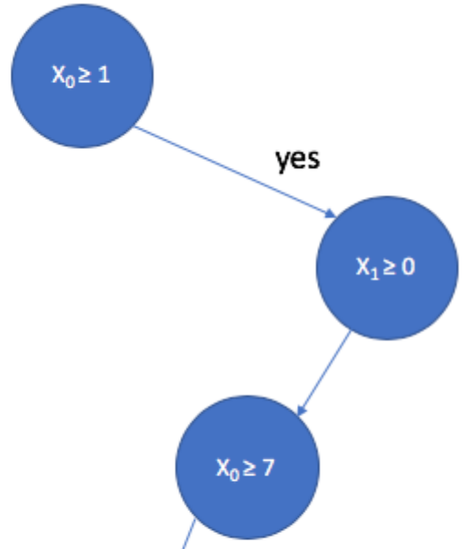
Аппроксимация ~ с большой вероятностью находим точку, которая не сильно дальше, чем ближайшая

Эффективность: точное нахождение 1NN (Exact Nearest Neighbors)

kD-деревья / k-d tree (Bentley, 1979) для евклидова пространства



Query: (4, -2)



Now we only have to calculate distances against a small **subset** of vectors

строим дерево

для точки – находим лист области пространства, которой она принадлежит

ищем ближайшую в листе (для точности надо смотреть соседние листья)

https://www.jeremyjordan.me/scaling-nearest-neighbors-search-with-approximate-methods/

Эффективность: точное нахождение 1NN (Exact Nearest Neighbors)

kD-деревья / k-d tree (Bentley, 1979) для евклидова пространства

Как строить дерево? Например, так:

ищем признак с максимальной дисперсией,

разбиваем по его медиане

Когда ищем ближайший

поднимаемся по дереву,

оценивая расстояние до области

(если оно меньше до текущего ближайшего – ищем там)

Эффективность: Random Projection Trees

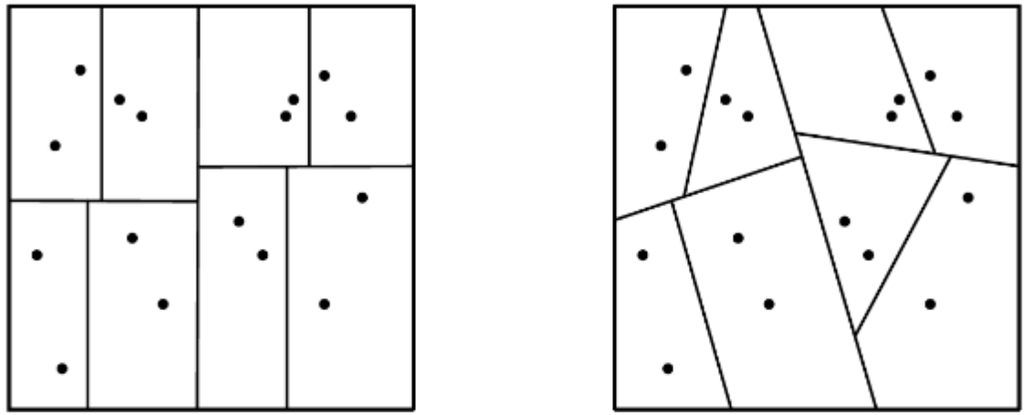


Figure 6.3: A comparison between how a k-d tree and a randomized partition tree divide up the feature space for two dimensional data (figure source: Dasgupta and Sinha 2015). Left: a k-d tree uses axis-aligned splits. Right: a randomized partition tree uses splits with directionality that is randomized (such as being drawn randomly from a unit ball).

Для приближённого поиска можно перебирать только объекты в листе

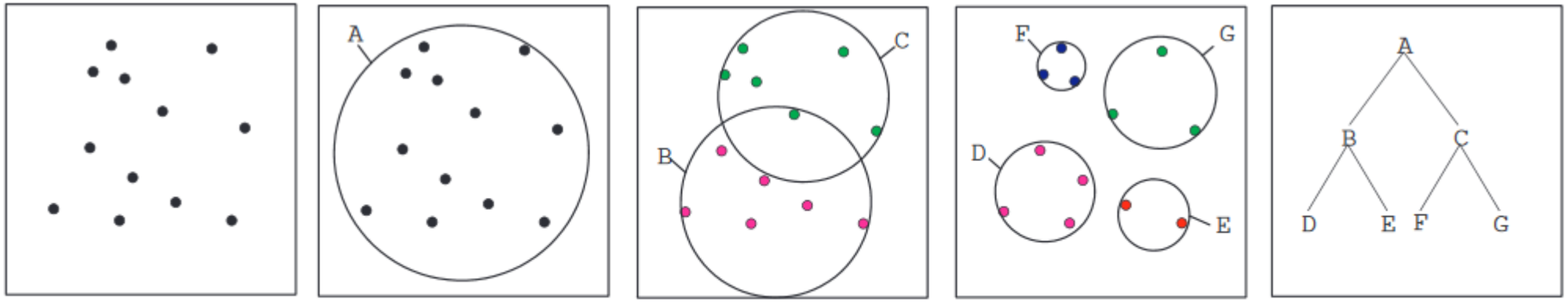
Можно построить несколько деревьев…

G. H. Chen, D. Shah Explaining the Success of Nearest Neighbor Methods in Prediction // Publisher: Now Foundations and Trends

Ещё название «Annoy»

https://erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces.html

Эффективность: Ball Tree



1a. A dataset 1b. Root node 1c. The 2 children 1d. The 4 grandchildren 1e. The internal tree structure

Figure 1: An example of ball-tree structure

http://people.ee.duke.edu/~lcarin/liu06a.pdf

Эффективность: Approximate Nearest Neighbors – Locality-Sensitive Hashing (LSH)

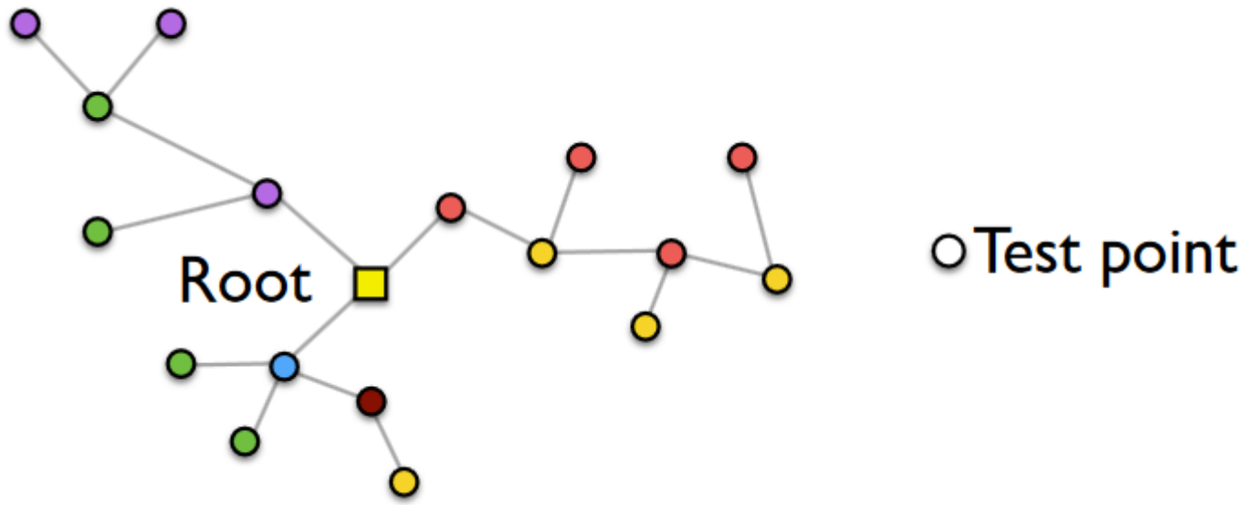
LSH ~ hash-функция: близкие объекты имеют похожие хэши

хэши короткие $\Rightarrow$ легко сравнивать

смотрим на все объекты с таким же hash-значением

если их нет, то на все (или используем другую hash-функцию)

Эффективность: Boundary Trees



Пусть есть построенное дерево и точка

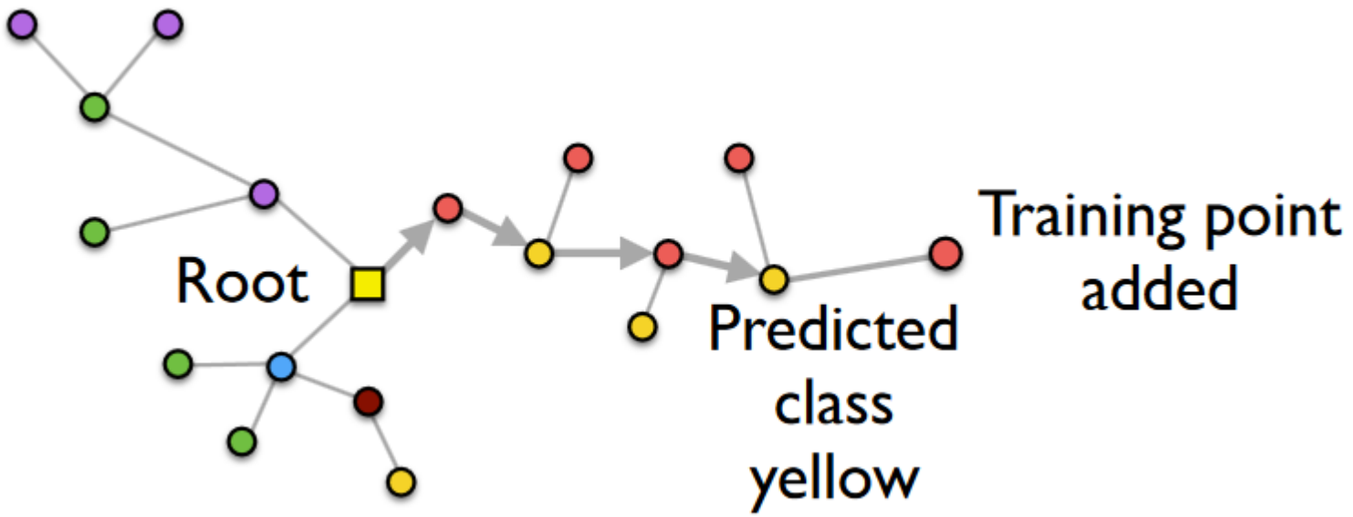
Спускаемся по дереву: смотрим расстояние до текущей вершины и её потомков

Если до какого-то потомка ближе – спускаемся (переходим в него)

Если нет (или в листе) – добавляем вершину / или нашли «соседа»

Mathy, C., N. Derbinsky, J. Bento, J. Rosenthal, and J. Yedidia (2015) «The Boundary Forest Algorithm for online supervised and unsuper-vised learning» In: AAAI Conference on Artificial Intelligence.

Эффективность: Boundary Trees



Пусть есть построенное дерево и точка

Спускаемся по дереву: смотрим расстояние до текущей вершины и её потомков

Если до какого-то потомка ближе – спускаемся (переходим в него)

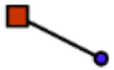
Если нет (или в листе) – добавляем вершину / или нашли «соседа»

При обучении можно добавлять вершину только,
если предсказание по текущему дереву на ней неверное

Эффективность: Boundary Trees


Boundary Tree

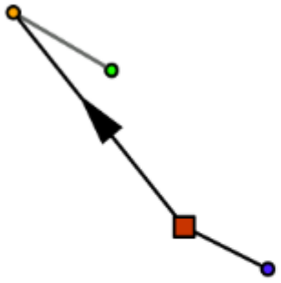

Querying BT everywhere


Boundary Tree

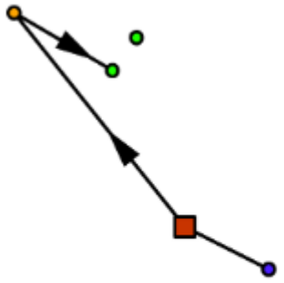

Querying BT everywhere


Boundary Tree

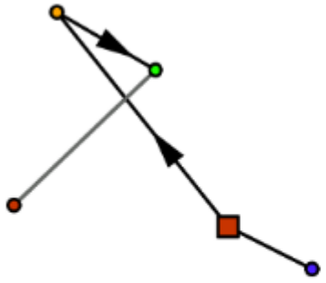

Querying BT everywhere


Boundary Tree


Querying BT everywhere

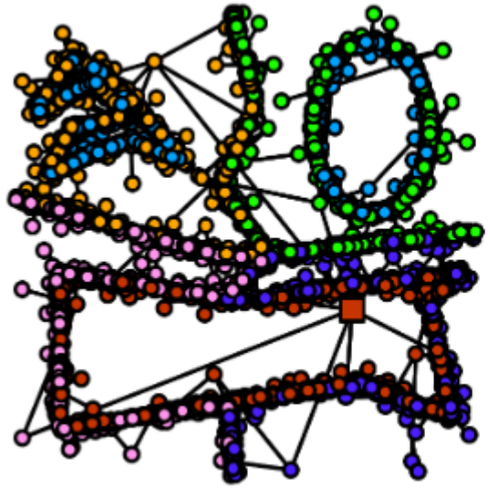

Boundary Tree


Querying BT everywhere


Boundary Tree

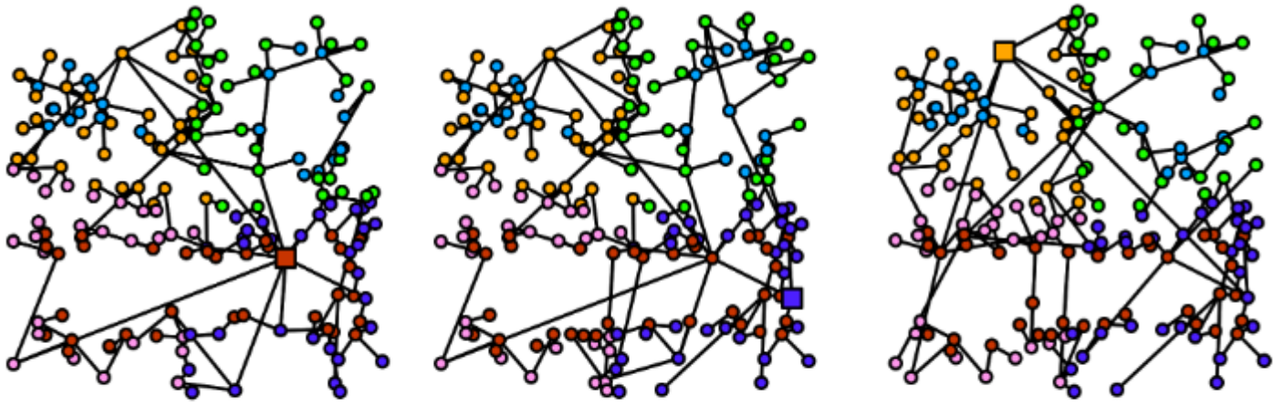

Querying BT everywhere

Эффективность: Boundary Trees



Boundary Tree

Querying BT everywhere



Shown 10,000 points

Модельная задача:

m = 100 000

в дереве |V| = 2 220

Лес:

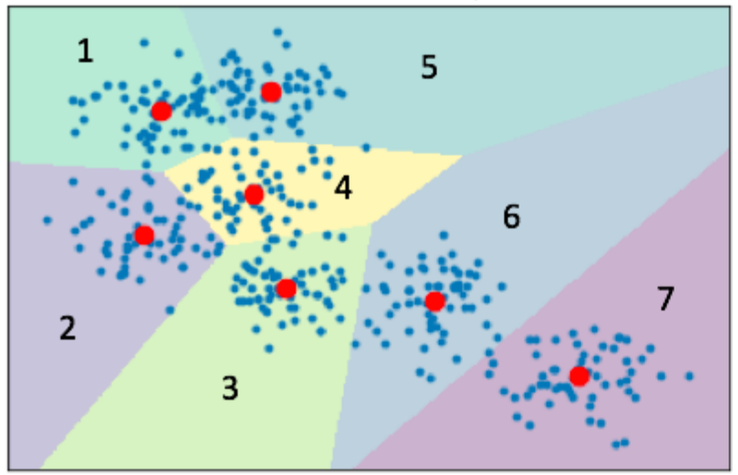
Строим с разным обходом выборки

Есть также обобщения на регрессию.

https://nn2017.mit.edu/wp-content/uploads/sites/5/2017/12/BoundaryForestNIPS2017.pdf

Эффективные методы поиска ближайших соседей

Quantization



Inverted Index

centroid	(ids)				
1	1	2	9	12	...
2	6	13	16	31	...
3	3	2	22	29	...
4					
5			...		
6					
7	4	5	7	14	...

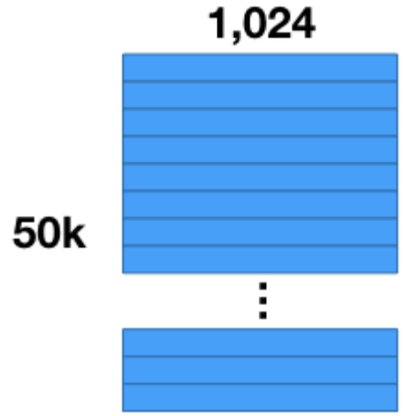
Assuming each vector has an identifier, we can maintain a list of vectors for each Voronoi cell. This type of data structure is known as an **inverted index**.

предварительная кластеризация

инвертированный индекс!

Эффективность: квантование «Product quantization»

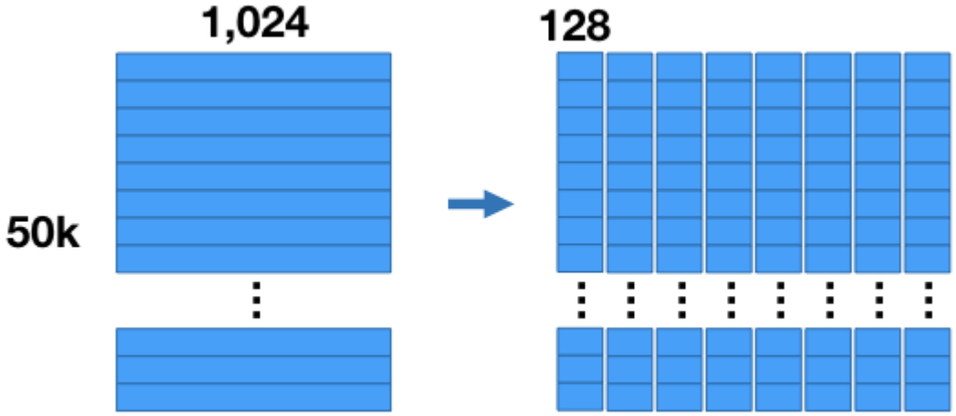
по-прежнему будем сравнивать с каждым из m объектов,
но упростим сравнение



Пример: изображение → вектор (пусть уже сделано с помощью НС)

Сейчас перейдём ещё к меньшей размерности

но это не снижение размерности (новое пр-во дискретное)

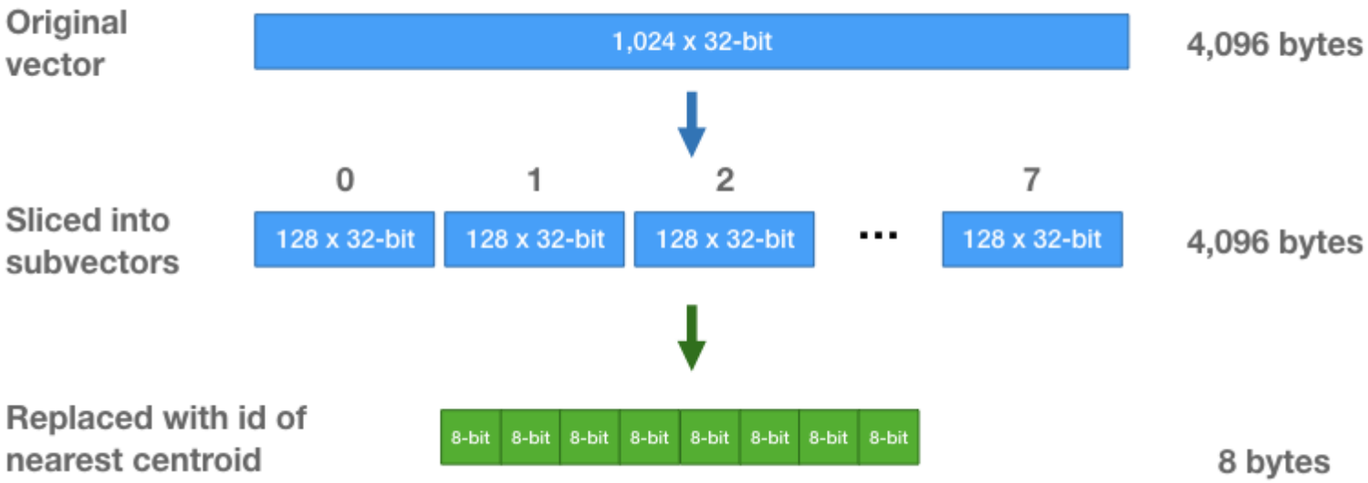


Разделим пространство на подпростанства (декартовым образом) 128*8

В каждом найдём k=256 центроидов

Будем хранить номера центроидов

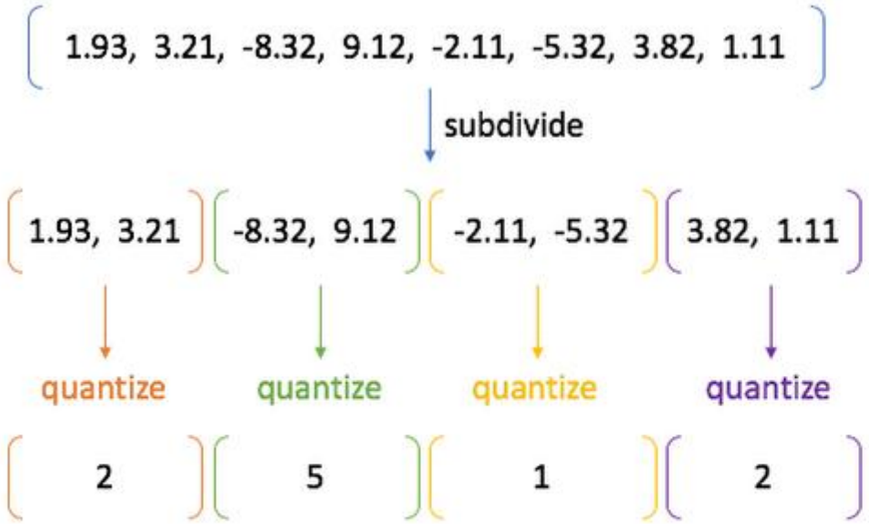
Эффективность: квантование «Product quantization»



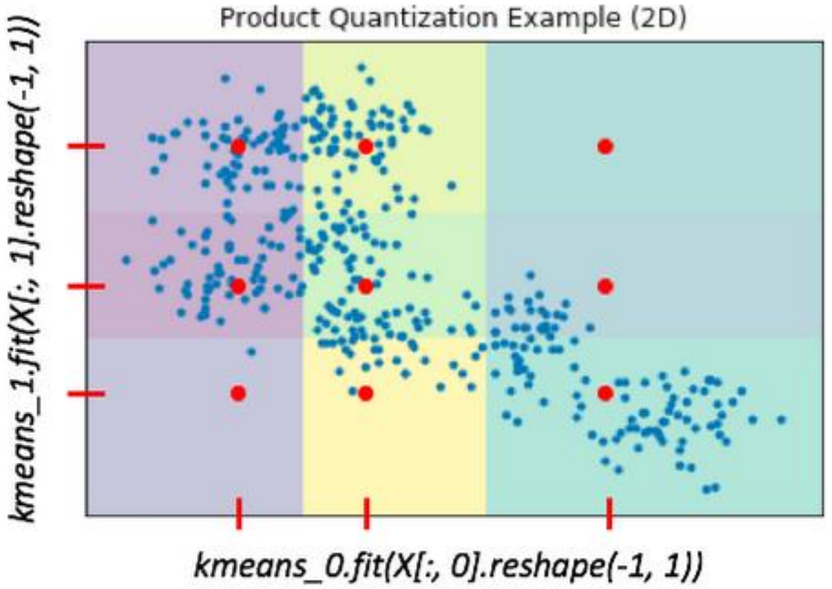
Для хранения id центроида – 8 бит

https://mccormickml.com/2017/10/13/product-quantizer-tutorial-part-1/

Эффективность: Product quantization



8D example



$X = (x_0, x_1)$

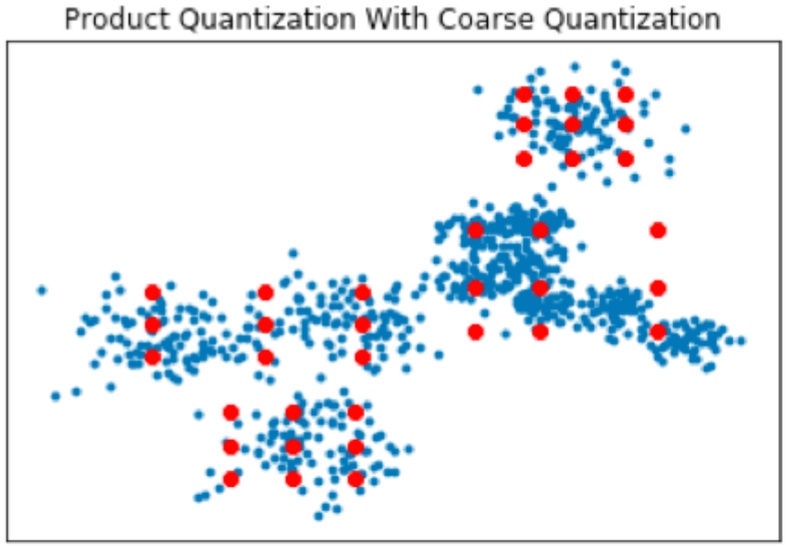
2D example (easier to visualize)

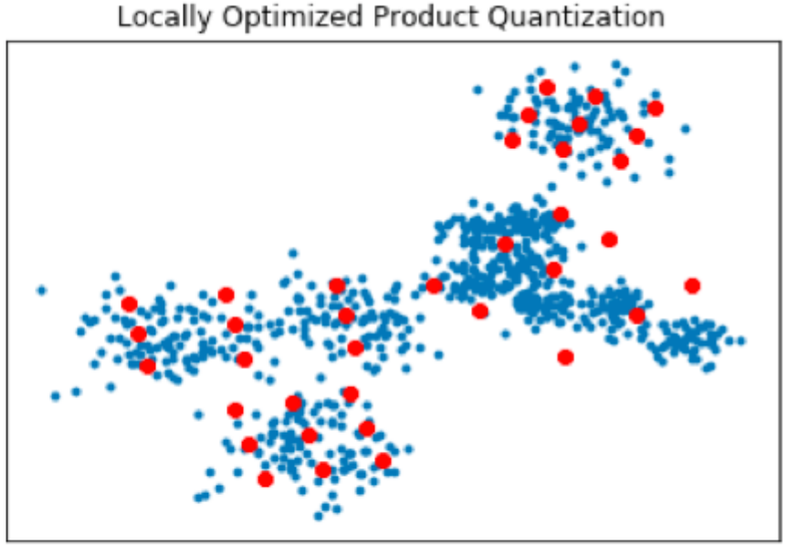
квантование по отдельным координатам (или подпространствам)

см. также

https://cache-ash04.cdn.yandex.net/download.yandex.ru/company/cvpr2012.pdf

Эффективность: Locally optimized product quantization





Locally optimize → PCA align the product centroids in each coarse cell

Эффективные методы поиска ближайших соседей

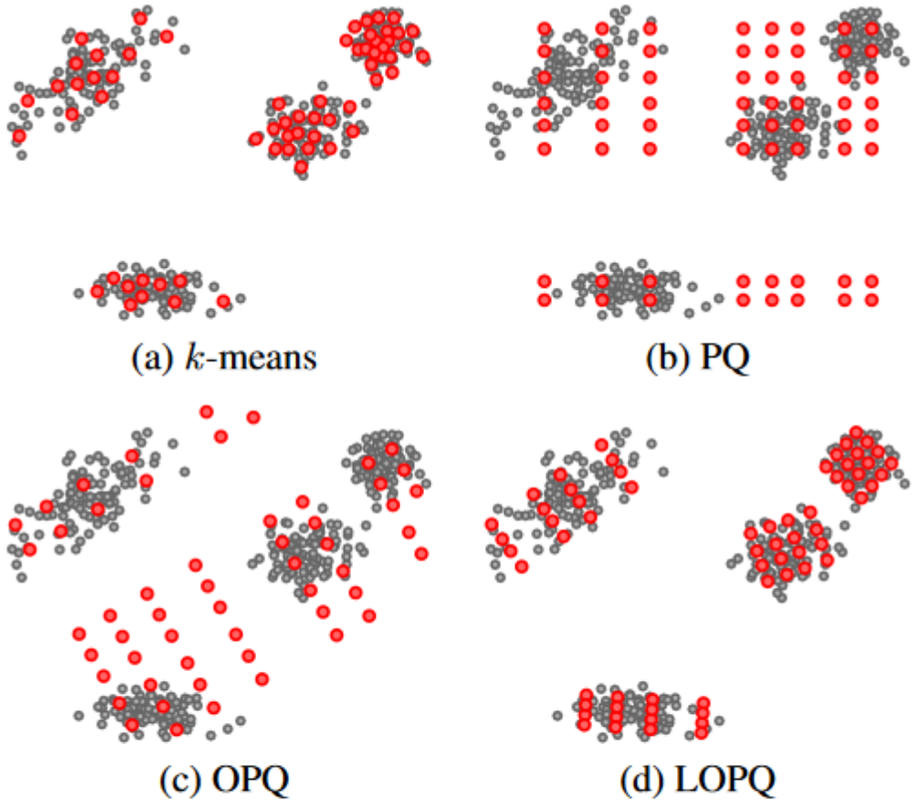
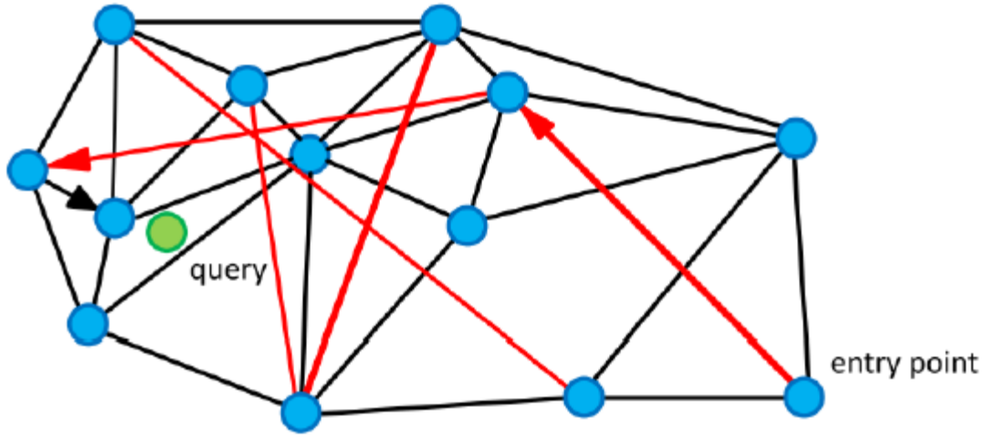


Figure 1. Four quantizers of 64 centroids (•) each, trained on a random set of 2D points (•), following a mixture distribution. (c) and (d) also reorder dimensions, which is not shown in 2D.

http://image.ntua.gr/iva/files/lopq.pdf

Эффективность: Navigable Small World (NSW)

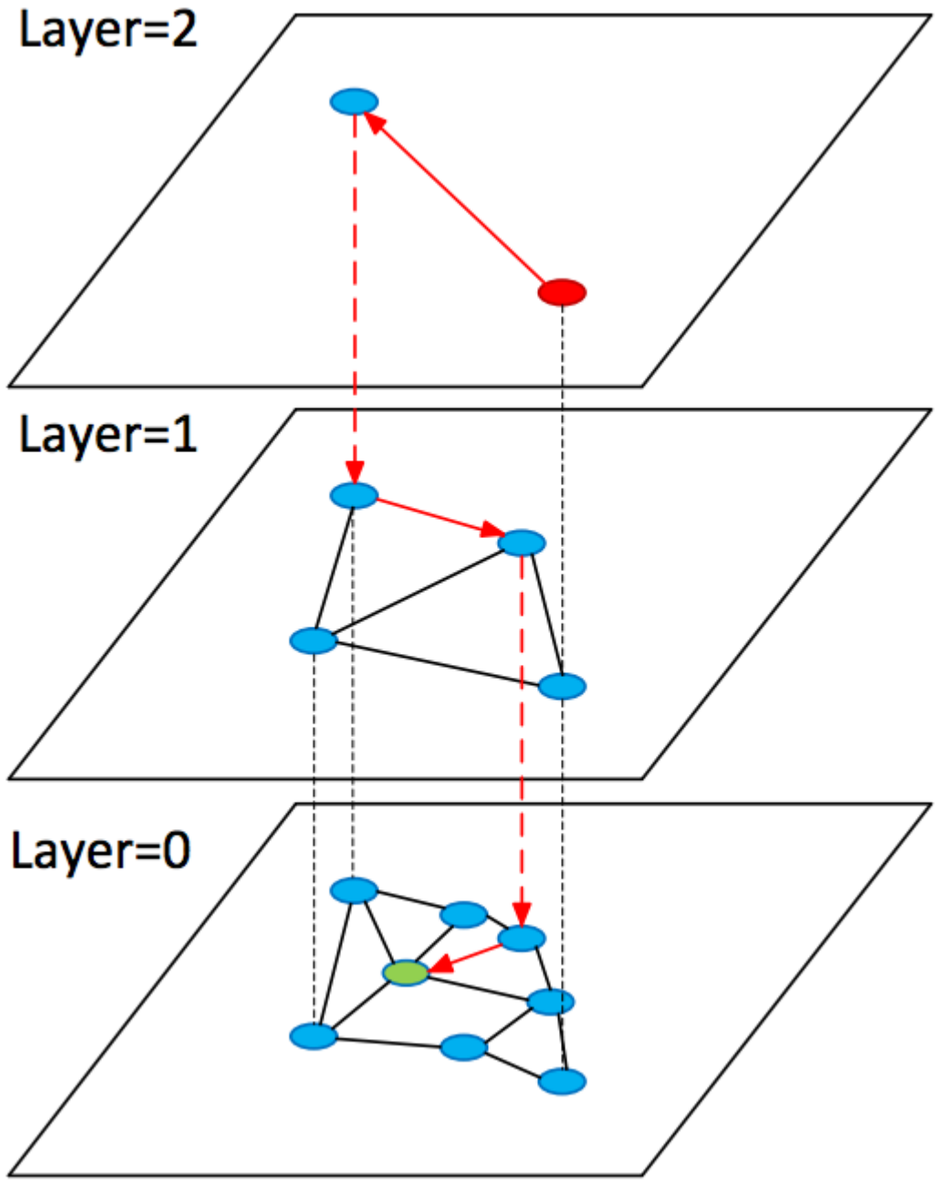
главное – найти быстро, но не факт, что найдём самого ближайшего



Одна из идей «small world graph»:
объект связан с k соседями
(в оригинале – графа Делоне, т.к. этого достаточно для поиска БС),
плюс t длинных связей

При поиске БС жадно идём по графу,
перебираем соседей текущей вершины,
считаем до них расстояния

Эффективность: Hierarchical Navigable Small World (HNSW)



Иерархические вложения:
слои – прореживание выборки

выполняем поиск послойно

+ простой и эффективный
+ эффективен по памяти

https://habr.com/ru/company/mailru/blog/338360/

Библиотеки

FALCONN - FAst Lookups of Cosine and Other Nearest Neighbors

https://github.com/FALCONN-LIB/FALCONN

Approximate Nearest Neighbor Search for Sparse Data in Python

https://github.com/facebookresearch/pysparnn

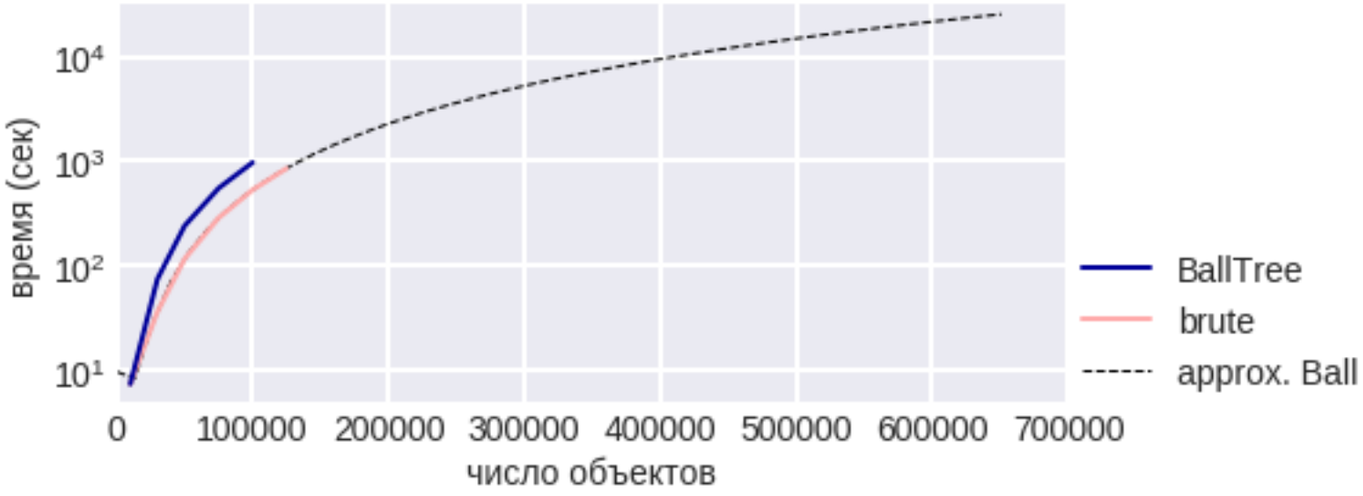
Faiss: A library for efficient similarity search

https://engineering.fb.com/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

Hubless Nearest Neighbor Search for Bilingual Lexicon Induction. ACL 2019

https://github.com/baidu-research/HNN#HNN

Эксперименты с метрикой Хэмминга



В R″ ищем 2 ближайших соседей.

С метрикой Хэмминга не поддерживается KDtree.

Слева – экстраполировали зависимость (шкала логарифмическая).

ещё по размерности пространства – вроде при n>10 можно обычный перебор использовать

Реализация sklearn.neighbors.NearestNeighbors

n_neighbors – **число соседей (5)**

radius – **ограничение пространства (1.0)**

algorithm – **алгоритм для определения БС** (auto, ball_tree, kd_tree, brute)

leaf_size – **параметр для** BallTree / KDTree

metric – **метрика (функция или строка:), см.** scipy.spatial.distance
scikit-learn:[cityblock, cosine, euclidean, l1, l2, manhattan]
scipy.spatial.distance: [braycurtis, canberra, chebyshev, correlation, dice, hamming, jaccard, kulsinski, mahalanobis, minkowski, rogerstanimoto, russellrao, seuclidean, sokalmichener, sokalsneath, sqeuclidean, yule]

p – **параметр для** minkowski **(2)**

metric_params – **дополнительные параметры для метрики**

n_jobs – **…**

Реализация KNeighborsClassifier/ KNeighborsRegressor

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
```

n_neighbors – **число соседей**

weights – **веса («uniform», «distance», функция)**

algorithm – **алгоритм для эффективного нахождения соседей («auto», «ball_tree», «kd_tree», «brute»)**

leaf_size – **для BallTree / KDTree**

p – **параметр для метрики Минковского**

metric – **метрика («minkowski»)**

metric_params – **параметры для метрики**

n_jobs – **число процессов для нахождения соседей**

```
from sklearn.neighbors import KNeighborsRegressor
```

weights – **весовая схема для объектов (**uniform, distance**, функция)**

Реализация

```
from sklearn.neighbors.nearest_centroid import NearestCentroid
```
– ближайший центроид

```
sklearn.neighbors.kneighbors_graph
```
– граф соседей

```
sklearn.neighbors.RadiusNeighborsClassifier
sklearn.neighbors.RadiusNeighborsRegressor
```
– алгоритмы с соседством по радиусу

```
sklearn.neighbors.NeighborhoodComponentsAnalysis
```
– обучение метрики

```
sklearn.neighbors.KernelDensity
```
– KDE-оценка плотности

Итог

Простые ленивые методы

можно использовать и без признаковых описаний

но задача свелась к выбору метрики

метод недооценёны!

Ближайший центроид

примитивный, но иногда эффективный

и быстрый метод

kNN

- не эффективен на больших данных (время, память)
- не более чем в 2 раза хуже оптимального алгоритма
- нет процедуры обучения
- выбросы / дисбаланс классов

Итог

весовые схемы

очень мощное оружие

метрики

много… есть специализированные

могут быть параметризованы

Эффективные способы обнаружения соседства

есть много современных способов

Код

https://github.com/Dyakonov/ml_hacks/blob/master/dj_IML_kNN.ipynb

Теория kNN

https://github.com/mlss-2019/slides/tree/master/learning_theory

«Машинное обучение и анализ данных»

Контроль качества и выбор модели

Александр Дьяконов

30 марта 2020 года

План

Проблема выбора модели (в широком смысле)

Способы контроля

отложенный / скользящий / перекрёстный / бутстреп / по времени

Три золотых правила разбиения выборки

моделируем реальность / нет утечкам / случайность

Локальный контроль

Где используется выбор CV-контроля

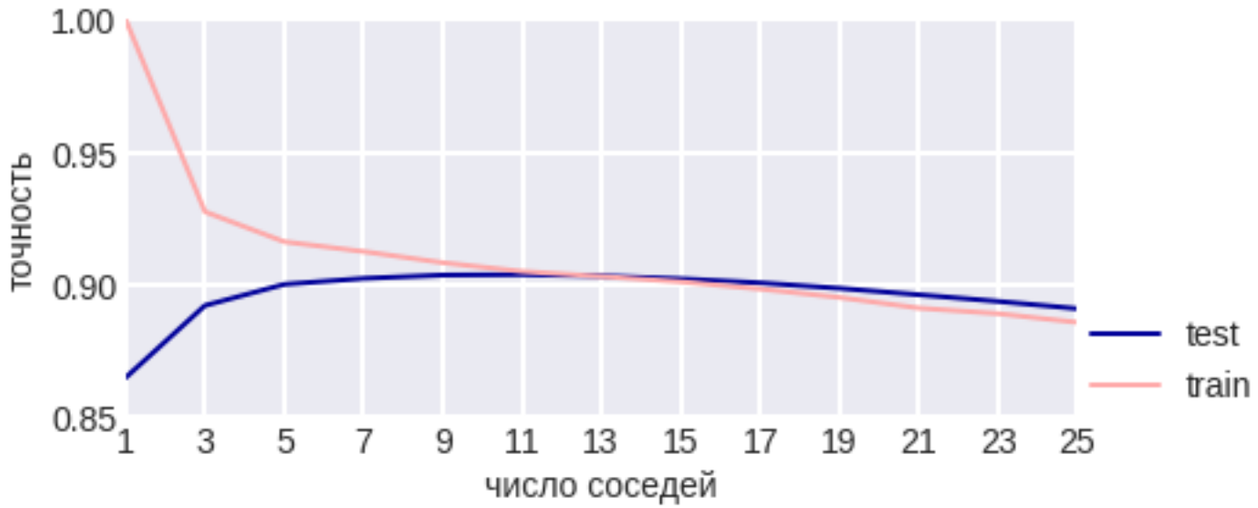
Проблема контроля качества

Ошибка на обучении (train error) и ошибка на контроле (test error) – как правило очень различаются!

Как оценить качество / ошибку алгоритма?

model performance / model error

Нужен способ оценить качество работы (в будущем) алгоритма



Проблема выбора модели (Model Selection)

Моделей очень много…

(но конечное число)

- **kNN**
- **SVM**
- **Linear**
- **NN**

…

Как выбрать модель?

В модели м.б. большое число алгоритмов

(континуальное)

В модели – как выбрать алгоритм?

1. В параметрических моделях – как выбрать параметры?

2. Как выбрать гиперпараметры (ex: k для kNN)?

Обычные параметры (model parameters) настраиваются на обучении, гиперпараметры (hyperparameters) выбираются до обучения

Очевидно, надо смотреть на качество – см. проблему выше…

MS может быть и для обучения без учителя!

Проблема выбора модели (Model Selection)

Выбор модели в широком смысле:

- **выбор модели алгоритмов**
- **выбор гиперпараметров**
- **выбор признаков**
- **выбор способа предобработки данных**

(заполнения пропусков, детектирования и удаления выбросов и т.п.)

Способы контроля

- отложенный контроль (held-out data, hold-out set)
- скользящий контроль (cross-validation)
- бутстреп (bootstrap)
- контроль по времени (Out-of-time-контроль)

используется для выбора модели и выбора гиперпараметров конкретной модели (выбирается модель с наименьшей ошибкой)

Общая схема / термины

Обучающая выборка – Training Set

обучение модели (настройка её параметров)

Валидационная выборка – Validation Set

настройка модели / (гипер)параметров модели /
выбор признакового пространства / ...
иногда: локальный контроль

Тестовая выборка – Test Set

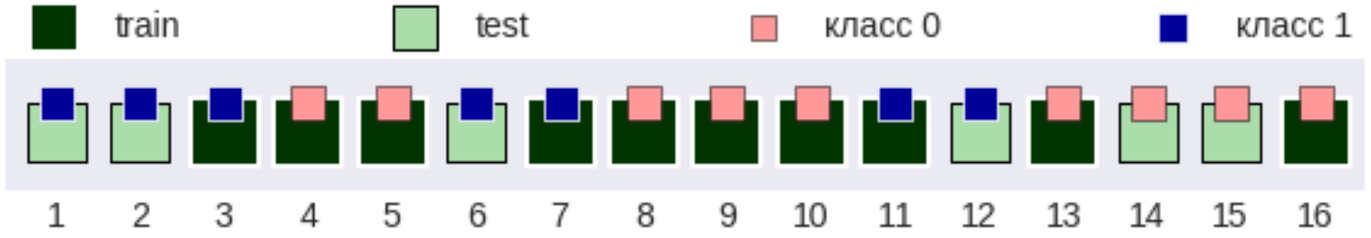
оценка качества модели
иногда: итоговая оценка

Отложенный контроль (held-out / validation data)

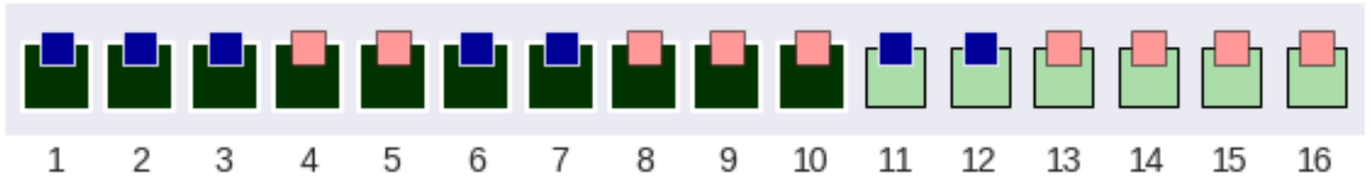
Выборку делим на две части:

- **обучение** – здесь настройка алгоритма
- **отложенный контроль** – здесь оценка качества, выбор алгоритма с наименьшей ошибкой

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=41)
```




```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, shuffle=False)
```



Отложенный контроль (held-out / validation data)

Оценка ошибки зависит от конкретной выбранной валидационной (отложенной выборки),
часто сильно меняется при другом выборе

Если переобучить алгоритм для всех данных, то мы не знаем оценку его ошибки
(в каком-то смысле, неустранимый недостаток)

В какой пропорции делить…
(обычно ~20% от выборки, это не тестовые данные!!!)

Большое обучение

- алгоритм больше похож на обученный по всем данным
- обучающая выборка более репрезентативная

Большой контроль

- оценка качества более надёжна

Три золотых правила разбиения выборки

- **Валидация моделирует реальную работу алгоритма!**

- **Нельзя явно или неявно использовать метки объектов, на которых оцениваешь ошибку (качество)**

- **Тест должен быть случайным (или специально подготовленным Вами)**

Первое правило: валидация моделирует реальную работу алгоритма

Пример: если алгоритм должен работать на новых пользователях,
для которых нет статистики,
то и в валидационной выборке должны быть такие новые пользователи

Следствие: пропорции классов должны сохраняться

`shuffle=True` **(в новой версии)**

вообще, распределения в обучении и тесте д.б. одинаковыми

вопрос: как с вещественным признаком?

Первое правило: валидация моделирует реальную работу алгоритма

Проблемы реализации:

- **наличие / отсутствие дубликатов (почти дубликатов)**
- **вхождение целиком групп данных**

пример: всё транзакции конкретного пользователя

Проблема реальности:

нестационарность (Nonstationarity – изменение параметров со временем)

- **Covariate shift – меняются распределения (популярность постов, доходы и т.п.)**
- **Concept drift – меняются правильные ответы (купил товар, уже не нужен)**

Второе правило: нельзя явно или неявно использовать метки объектов, на которых оцениваешь ошибку (качество)

Это для любых целей!

Пример (курс Тибширани): нельзя найти подмножество признаков, максимально коррелирующих с целевым, а потом методом k-fold CV оценить ошибку этого метода – будет неверной!

Третье правило: тест должен быть случайным (или подготовленным)

По умолчанию: `shuffle=True`

Random Subsampling Cross-Validation

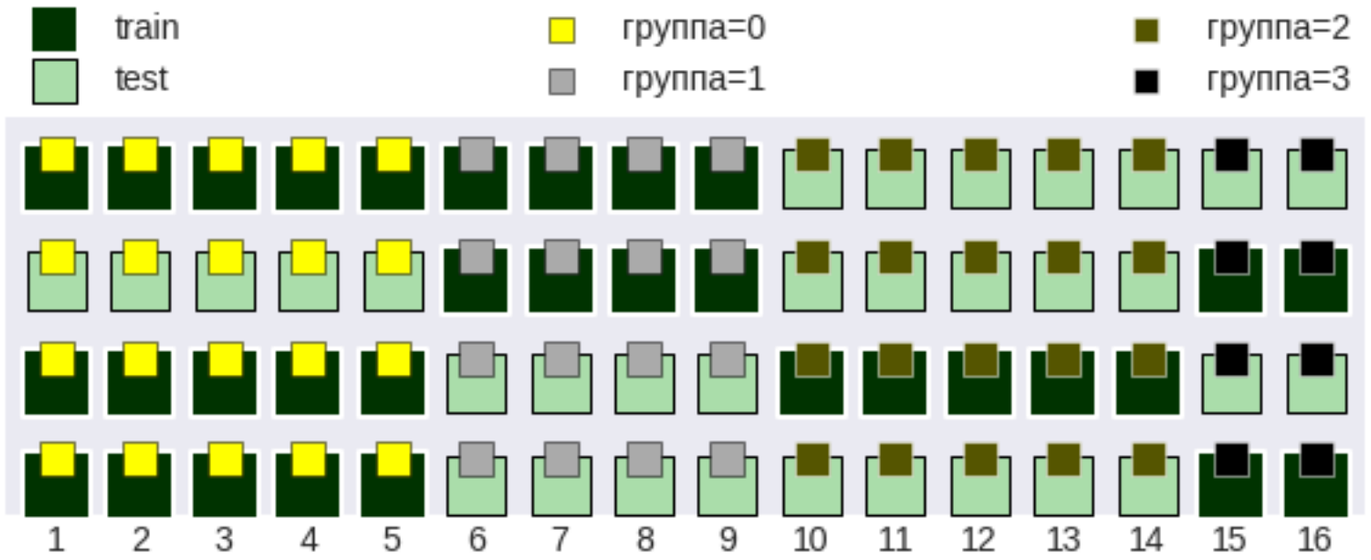
***k* раз случайно выбираем отложенный контроль, усредняем ошибки на всех отложенных выборках**

```
sklearn.model_selection.ShuffleSplit(n_splits=4,
                                      test_size=0.3,
                                      train_size=None,
                                      random_state=None)
```

Random Subsampling Cross-Validation: не забываем правила

без разбиения групп вопрос: когда это нужно?

```
sklearn.model_selection.GroupShuffleSplit(n_splits=4,
                                          test_size=0.3,
                                          train_size=None,
                                          random_state=None)

for t, (itrain, itest) in enumerate(cv.split(x, groups=g)):
    ...
```



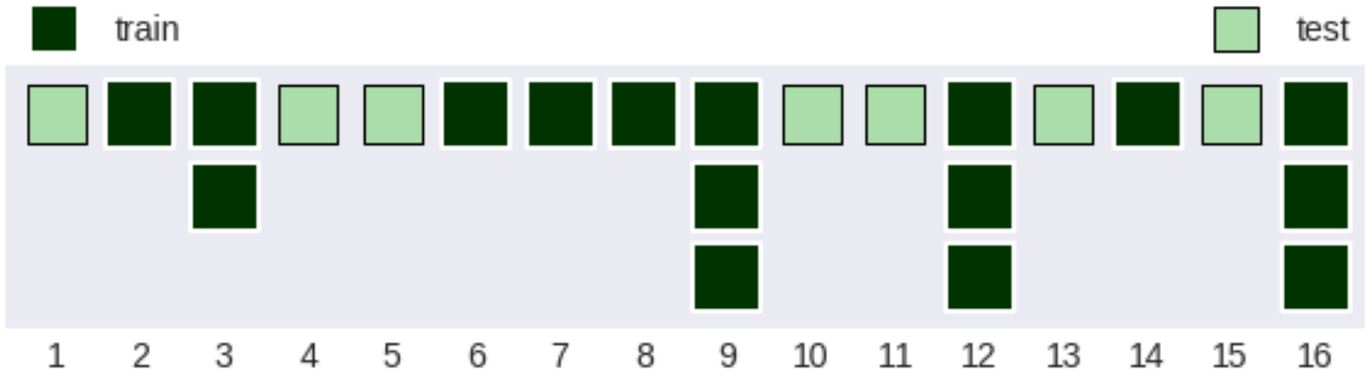
Random Subsampling Cross-Validation: не забываем правила

случайные разбиения сохраняя пропорции классов

```
sklearn.model_selection.StratifiedShuffleSplit(n_splits=4,
                                                test_size=0.3,
                                                train_size=None,
                                                random_state=None)

for t, (itrain, itest) in enumerate(cv.split(x, groups=y)):
    ...
```

Бутстреп (Bootstrap)

с помощью выбора с возвращением
формируется подвыборка полного объёма *m*,
на которой производится обучение модели
на остальных объектах (которые не попали в обучение) – контроль




```
i_train = [9, 16, 14, 9, 7, 12, 3, 12, 9, 8, 3, 2, 16, 12, 6, 16]
i__test =  [1, 4, 5, 10, 11, 13, 15]
```

Бутстреп (Bootstrap)

В контроль попадает примерно

$$\left(1-\frac{1}{m}\right)^m \approx e^{-1} \approx 0.37 = 37\%$$ **выборки.**

Чем хорошо:

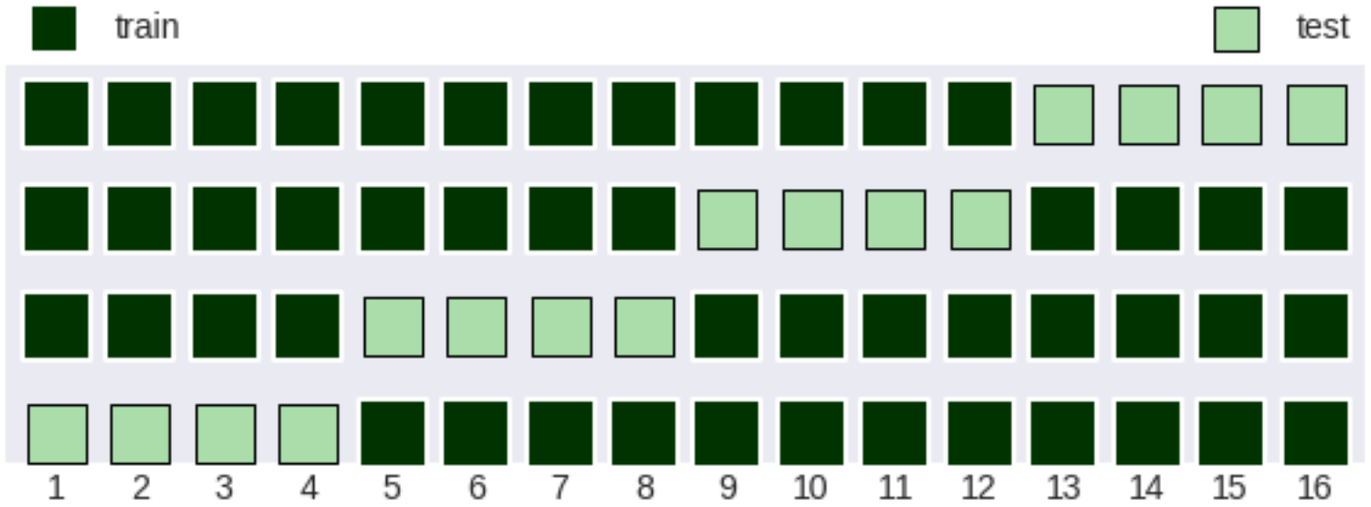
- **модель учится на выборке того же объёма,**
что и итоговая (которую мы обучим по всей)
Но… использует не все данные!
есть дубликаты

- **с точки зрения распределения бутстреп-выборка похожа на исходную**

Перекрёстная проверка по фолдам (k-fold cross-validation)

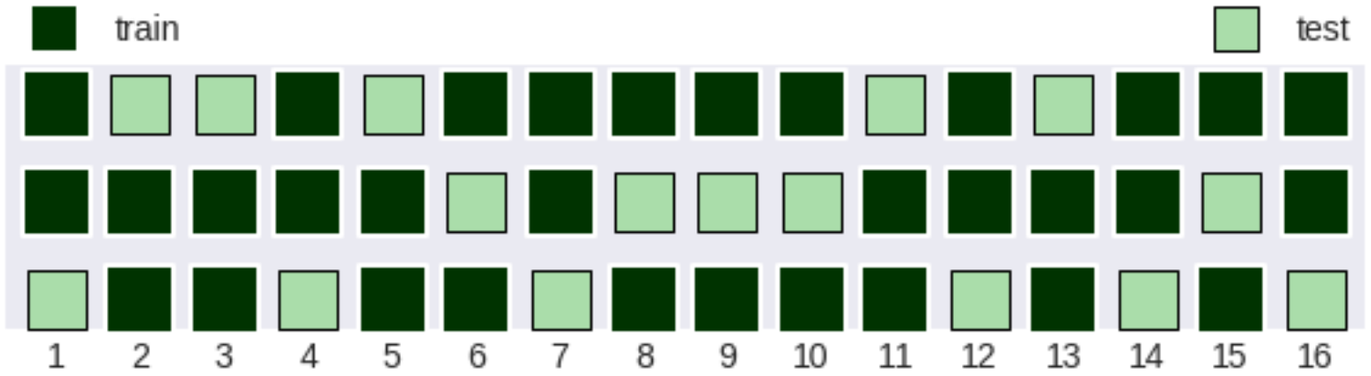
- **Разделить выборку на *k* примерно равных частей**
- **цикл по *i* = 1…*k***
  - **использовать *i*-ю часть для валидации (вычислить ошибку на ней),**
    а объединение остальных – для обучения
- **усреднить *k* ошибок, вычисленных на разных итерациях цикла**
  (можно использовать дисперсию для оценки доверия к полученному качеству)

```
sklearn.model_selection.KFold(n_splits='warn',
                              shuffle=False,
                              random_state=None)
```

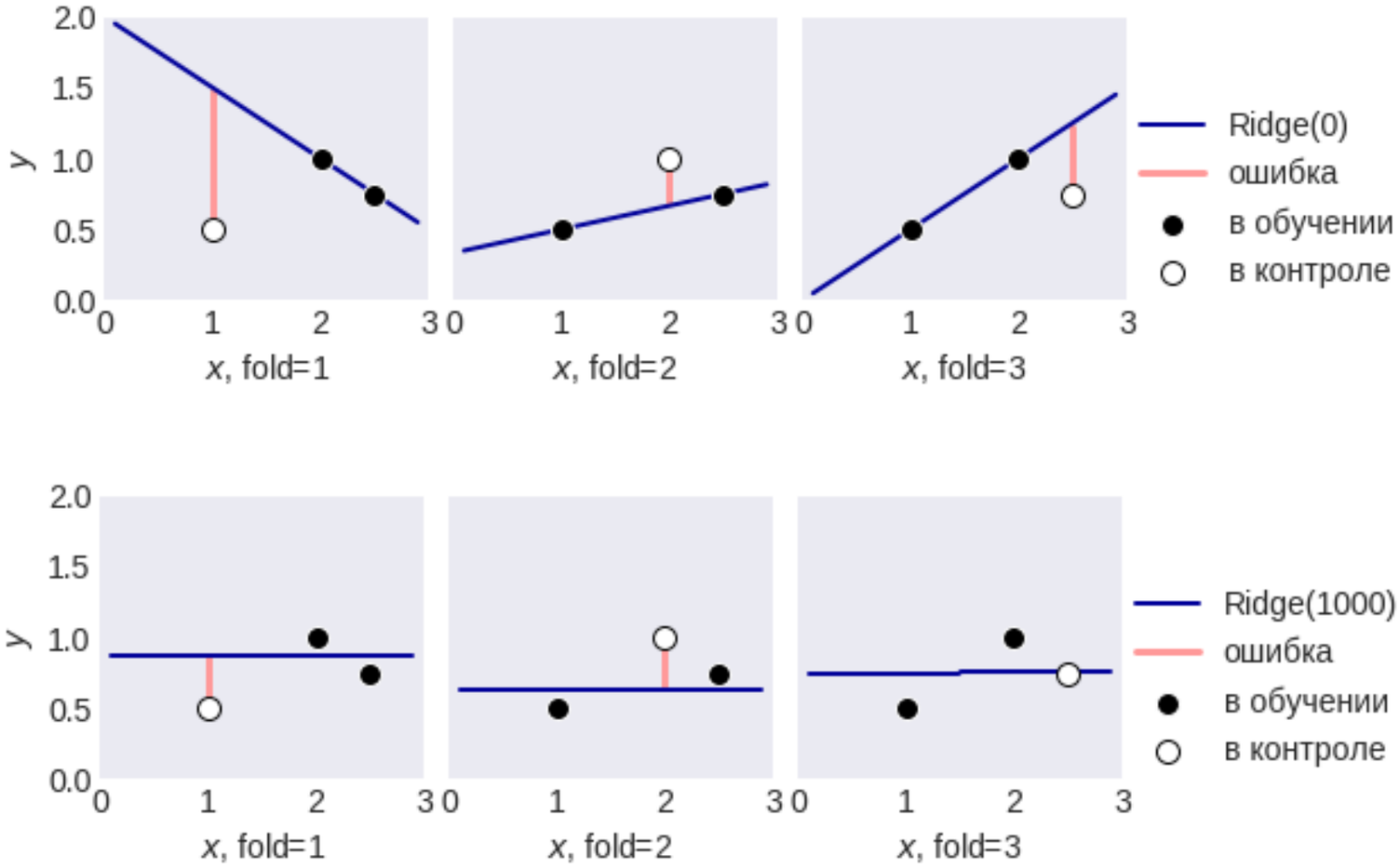


Перекрёстная проверка по фолдам (k-fold cross-validation)

```
sklearn.model_selection.KFold(n_splits=3, shuffle=True, random_state=None)
```



Перекрёстная проверка по фолдам: иллюстрация



Перекрёстная проверка по фолдам: тонкости

«k примерно равных частей»

в последней части может быть меньше объектов, чем в остальных,

если k не делит нацело объём выборки

Обычно выбирают k=10

Большие k –

- **надёжнее оценка качества**
- **обучающая выборка больше походит на все данные**
- **время контроля возрастает (линейно)!**
- **не любое качество адекватно оценивается на меленьких подвыборках**

Перекрёстная проверка по фолдам: не забываем правила

сохраняем пропорцию классов

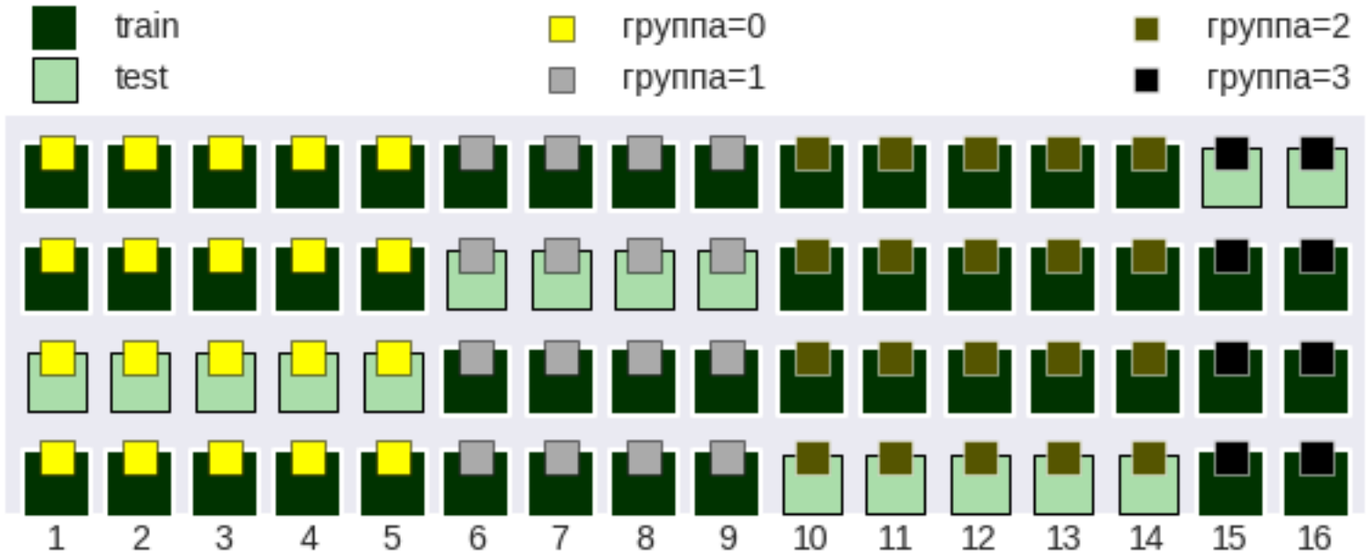
```
sklearn.model_selection.StratifiedKFold(n_splits='warn',
                                        shuffle=False,
                                        random_state=None)
```

перемешиваем: `shuffle=True`

Перекрёстная проверка по фолдам: не забываем правила

не разбиваем группы

```
sklearn.model_selection.GroupKFold(n_splits='warn')
```



ещё есть: `sklearn.model_selection.PredefinedSplit` **– разбиение индуцированное группами**

Leave-one out cross-validation (LOOCV)

k-fold при *k*=*m*

`sklearn.model_selection.LeaveOneOut`

показаны только первые 4 разбивки…

- **может слишком долго вычисляться**

ещё есть `sklearn.model_selection.LeavePOut` **– всевозможные *P*-ки**

Leave-one out cross-validation (LOOCV)

Для справки: в статистике аналогичный метод «Складной нож» (jackknife) для оценки параметров

$$\overline{x}_{-i} = \frac{1}{m-1}\sum_{j \neq i} x_j$$

$$\overline{x} = \frac{1}{m}\sum_{i=1}^{m} \overline{x}_{-i}$$

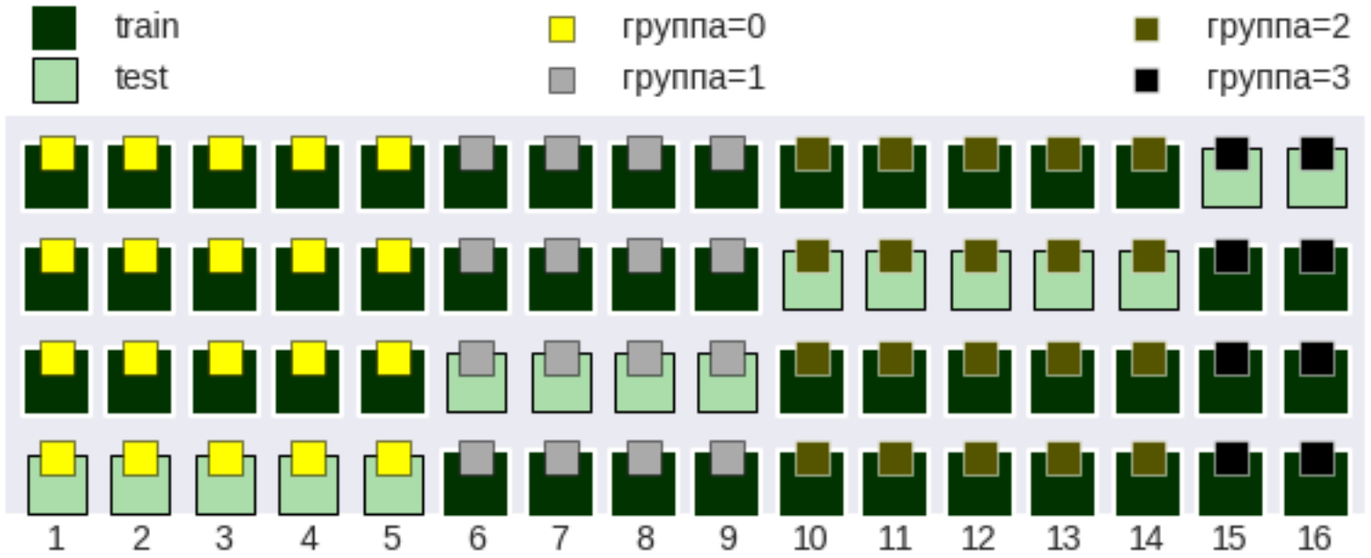
$$\overline{\text{var}} = \frac{m-1}{m}\sum_{i=1}^{m} (\overline{x} - \overline{x}_{-i})^2$$

на практике чаще применяют бутстреп-оценки

Контроль по группам: LeaveOneGroupOut

LeaveOneGroupOut: Контроль по одной группе

```
from sklearn.model_selection import LeaveOneGroupOut
```



тонкость:

при оценки ошибки можно (нужно?)

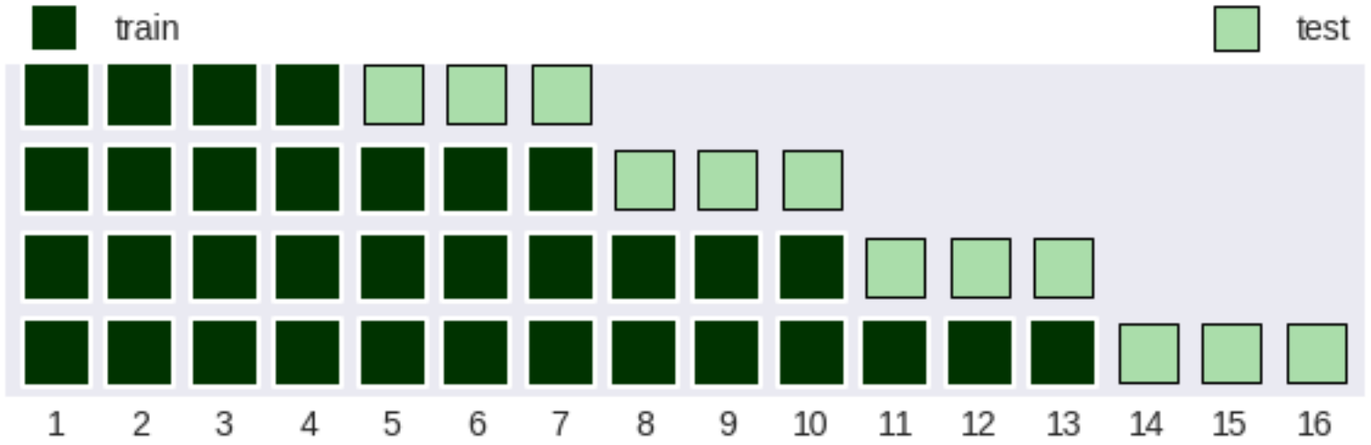
учитывать мощность групп

$$e = \sum_{t=1}^{k} \frac{m_t}{m} e_t$$

Контроль по времени (Out-of-time-контроль)

TimeSeriesSplit: разбиения временных рядов (Time series cross-validation)

```
sklearn.model_selection.TimeSeriesSplit(n_splits='warn',
                                        max_train_size=None)
```



- **часто не получится сделать много контролей**

  (слишком маленькая предыстория)

- **первое правило ⇒ знаем как организовывать**

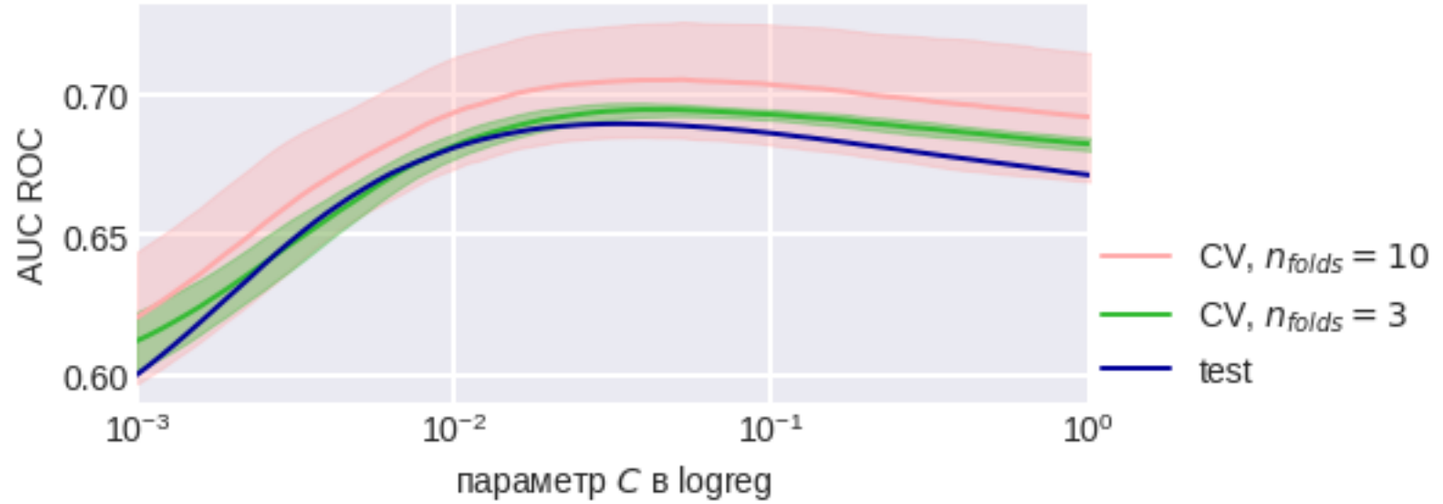
Терминология

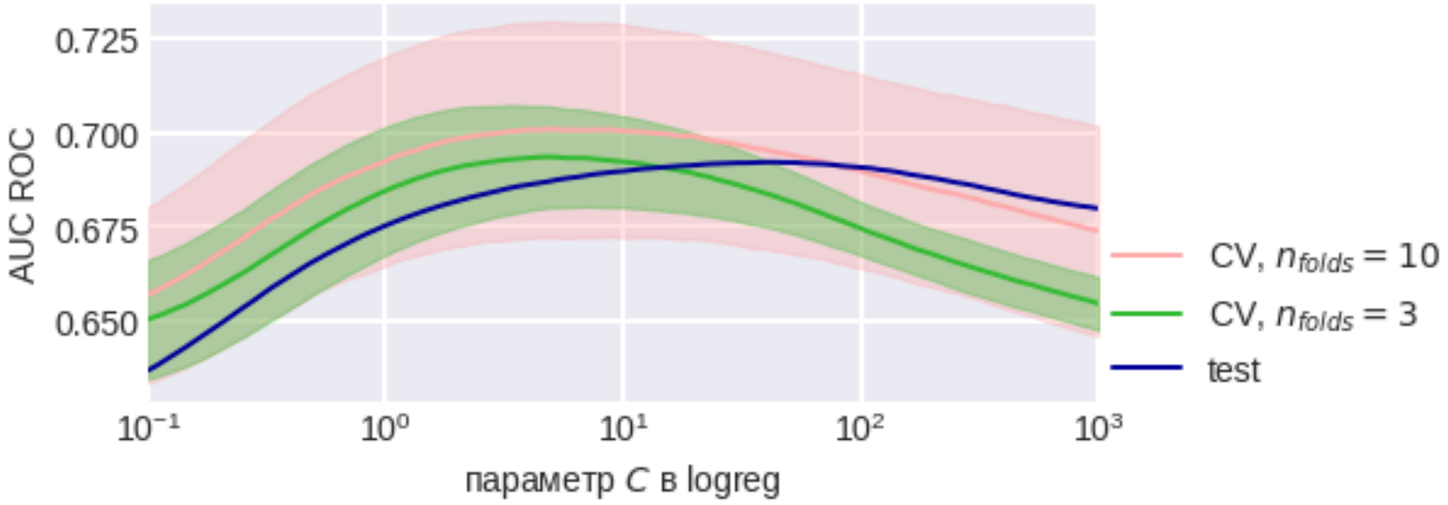
user index

Training Data	in sample out of time
out of sample in time	out of sample out of time

time

Сравнение CV и настоящей ошибки





Две похожие задачи: есть и нет согласование CV / test

Локальный контроль

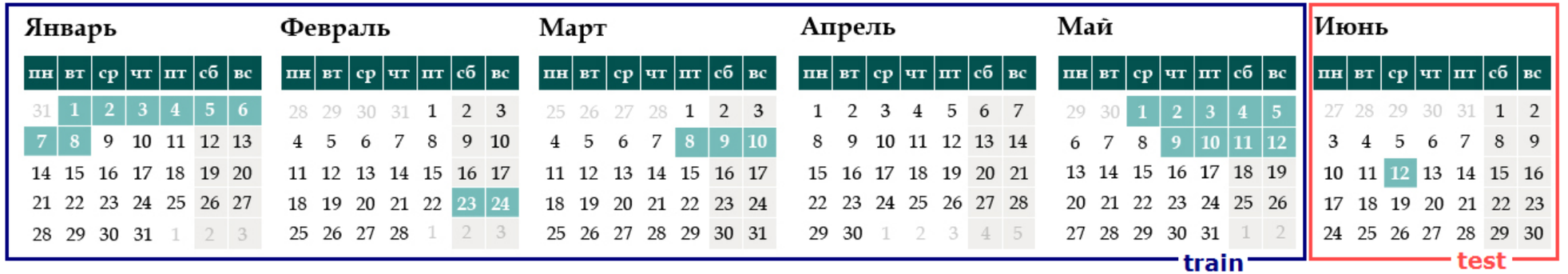
– организация контроля, когда итоговое качество алгоритма будет оцениваться на заранее заданной выборке (глобальном контроле)

- соревнования
- обучение с частичной разметкой (semi-supervised learning)

Вспоминаем первое золотое правило…

- распределения по признакам должны совпадать
- распределение выбросов, пропусков…
- если прогнозирование – на похожие период

(с того же дня недели, столько же праздников впереди и т.п.)



Корректность локального контроля при использования кодировок

Разбиения корректные (например, OOT)

Локально:	Глобально:
Определяем кодировки на train_code	Определяем кодировки на train
Кодируем train_code и valid	Кодируем train и test
Регуляризуем на train_code	Регуляризуем на train
Валидируем на valid	Обучаем на train, предсказываем на test

Где используется выбор CV-контроля

- **оценка качества модели**

- **получение «честных» ответов на обучении**

как бы алгоритм отвечал,
если бы не обучался на этих объектах

- контроль алгоритмов
- ансамблирование (метапризнаки)

- **настройка гиперпараметров**

- **построение кривых зависимостей качества от параметров**
  - validation curves (от значений)
  - learning curves (от объёма выборки)

Оценка модели с помощью выбранного контроля: минутка кода

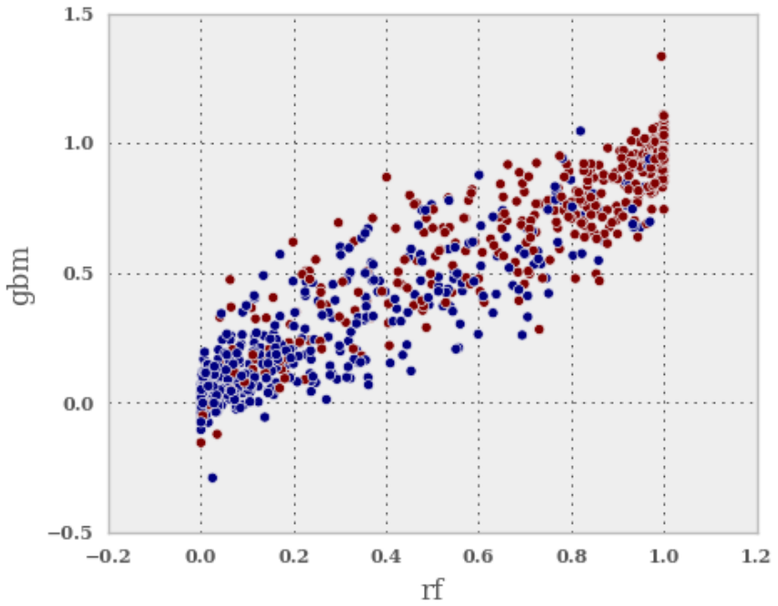
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
cv = ShuffleSplit(n_splits=3, test_size=0.1,
                  train_size=None, random_state=None)
cross_val_score(logreg, X, y, cv=cv)
```

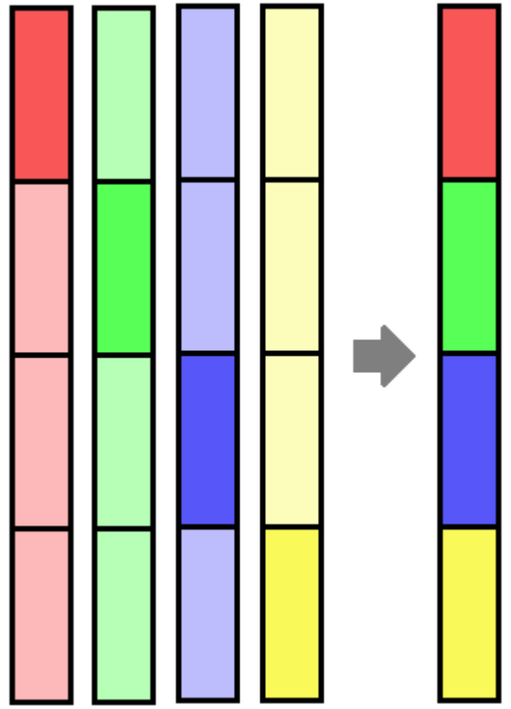
У этих функций много параметров…

Они (функции) «понимают» друг друга

Если не указываем скорер – используется встроенный (в модель)

Ответы алгоритма с помощью выбранного контроля: минутка кода



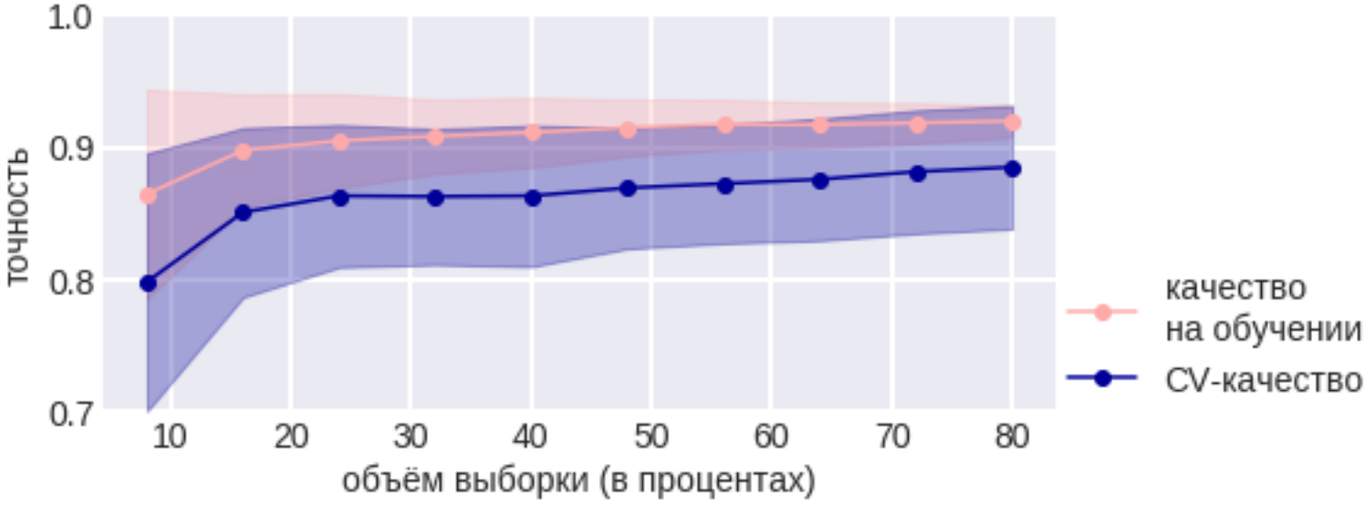


```
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
cv = KFold(n_splits=10, shuffle=True, random_state=1)
a_rf = cross_val_predict(rf, X, y, cv=cv) # ответы rf на CV
a_gbm = cross_val_predict(gbm, X, y, cv=cv) # ответы gbm на CV

plt.scatter(a_rf, a_gbm, c=y)
plt.xlabel('rf')
plt.ylabel('gbm')
```

Кривые обучения (Learning Curves)

Делим данные на обучение и контроль (м.б. очень много раз)
Обучаемся на k% от обучающей выборки для разных k
Строим графики ошибок/качества на train/CV от k

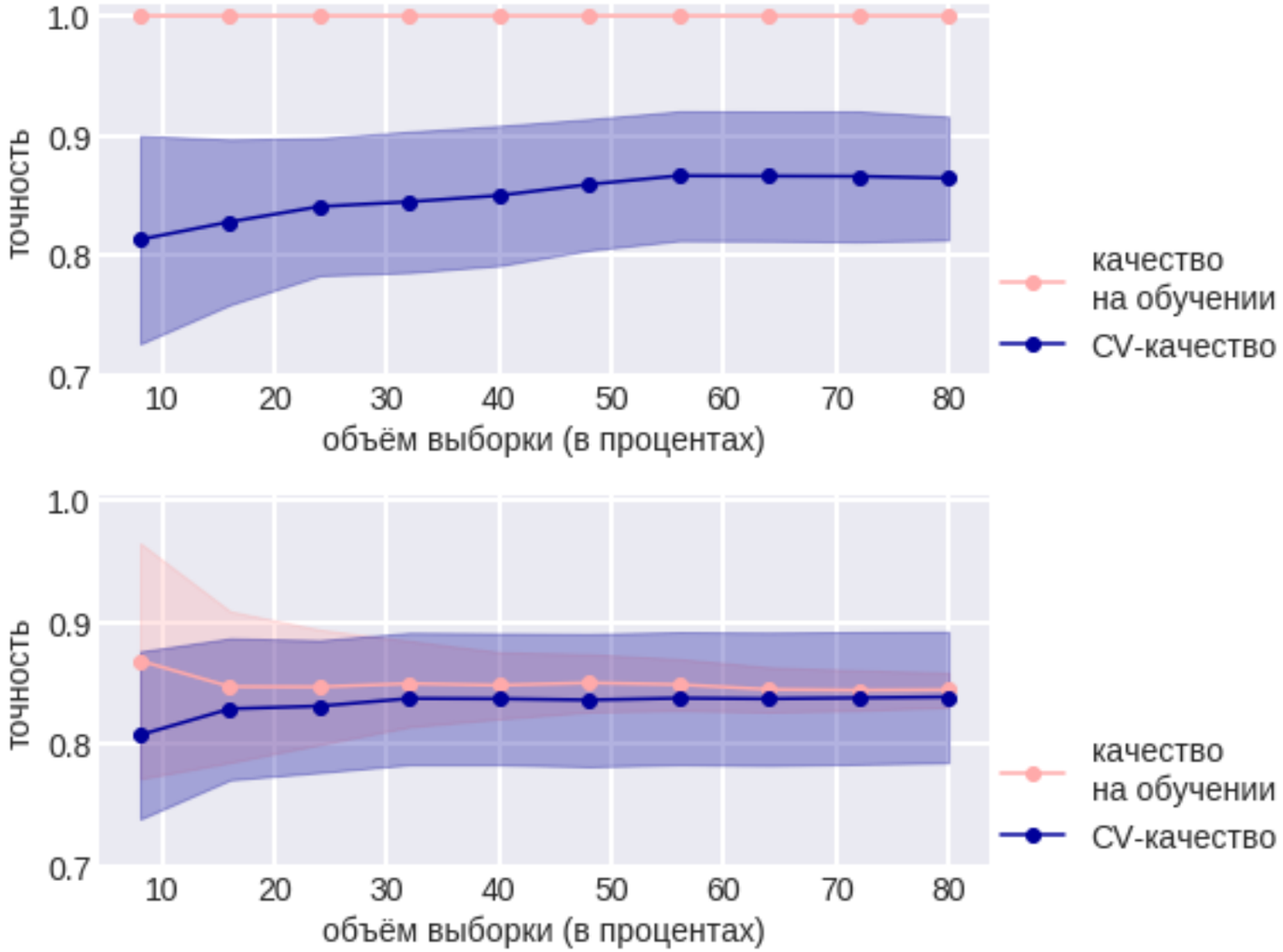
```
model_selection.learning_curve
train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv,
                                                        n_jobs=n_jobs, train_sizes=train_sizes)
```



Есть зазор между обучением и CV

Тонкость: 100% – вся выборка, но здесь `test_size=0.2`

Кривые обучения (Learning Curves)



Переобучение:
100% качество на обучении!
1NN?

Нет выгоды от объёма!

Полная согласованность между обучением и CV,
но качество низкое

Переобучение / недообучение

overfitting

плохо, когда качество на обучении ↑,
а на контроле низкое или ↓

Причины переобучения: сложность модели, шум, нерепрезентативность

underfitting

плохо, когда на обучении и контроле качество совпадает
и оно низкое

**дальше будет: более сложную модель / меньше регуляризацию,
больше тренировать,
больше данных,
другую модель**

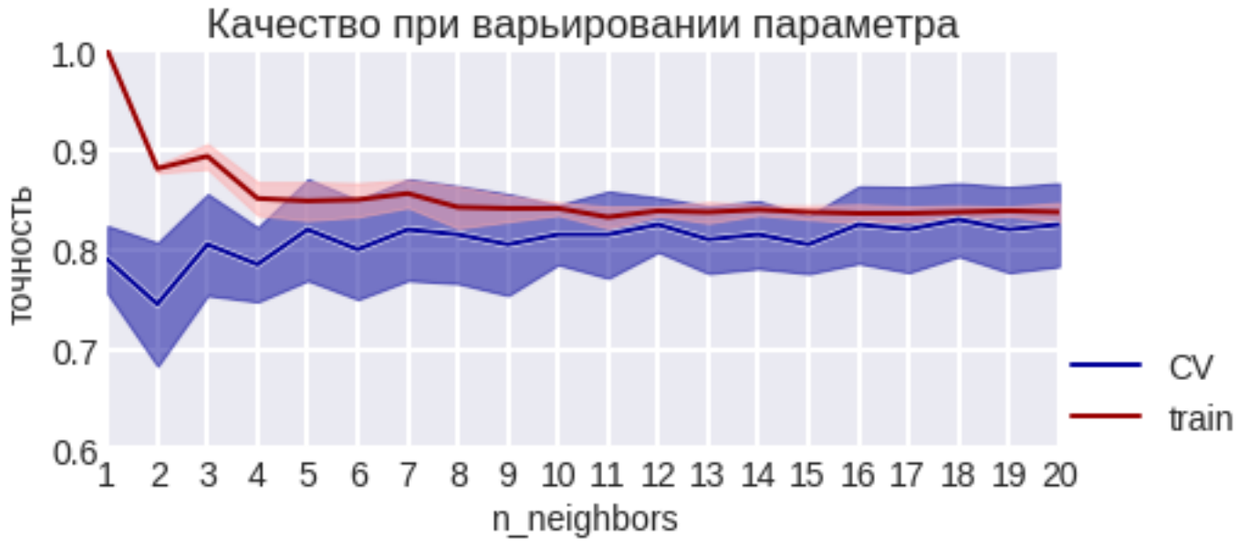
Качество от параметров

Делим данные на обучение и контроль (м.б. очень много раз)

При разных значениях параметров обучаемся и проверяем качество

Строим графики ошибок/качества на train/CV от значения параметра

`sklearn.model_selection.validation_curve`



Настройка параметров

градиентные методы
(gradient-based)

Метаоптимизация

Байесовская оптимизация
(Bayesian model-based optimization)

Перебор

Ручной перебор (Manual)

(Квази)полный перебор (Grid search)

Стохастические методы

Случайный поиск (Random search)

Эволюционные алгоритмы (evolutionary)

Замечание

Многие методы (например, RF, GBM) проще оптимизировать вручную!

дальше будем

Перебор параметров

Делим данные на обучение и контроль (м.б. очень много раз)

При разных значениях параметров обучаемся и проверяем качество

```
from sklearn.model_selection import GridSearchCV
parameters = {'metric':('euclidean', 'manhattan', 'chebyshev'),
              'n_neighbors':[1, 3, 5, 7, 9, 11], scoring='roc_auc'}
clf = GridSearchCV(estimator, parameters, cv=5)
clf.fit(X, y)
clf.cv_results_['mean_test_score']
```

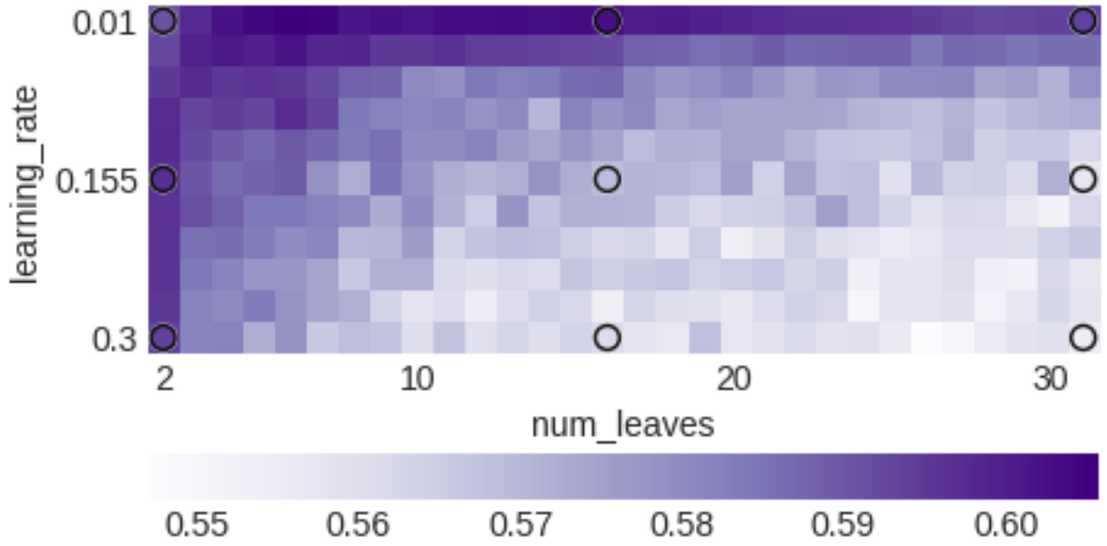
	$k=1$	$k=3$	$k=5$	$k=7$	$k=9$	$k=11$
euclidean	76.0	77.0	79.0	78.5	80.5	82.5
manhattan	74.0	74.0	79.0	79.5	80.5	81.0
chebyshev	76.5	78.5	80.0	80.0	81.0	81.5

Есть также случайный поиск `model_selection.RandomizedSearchCV`

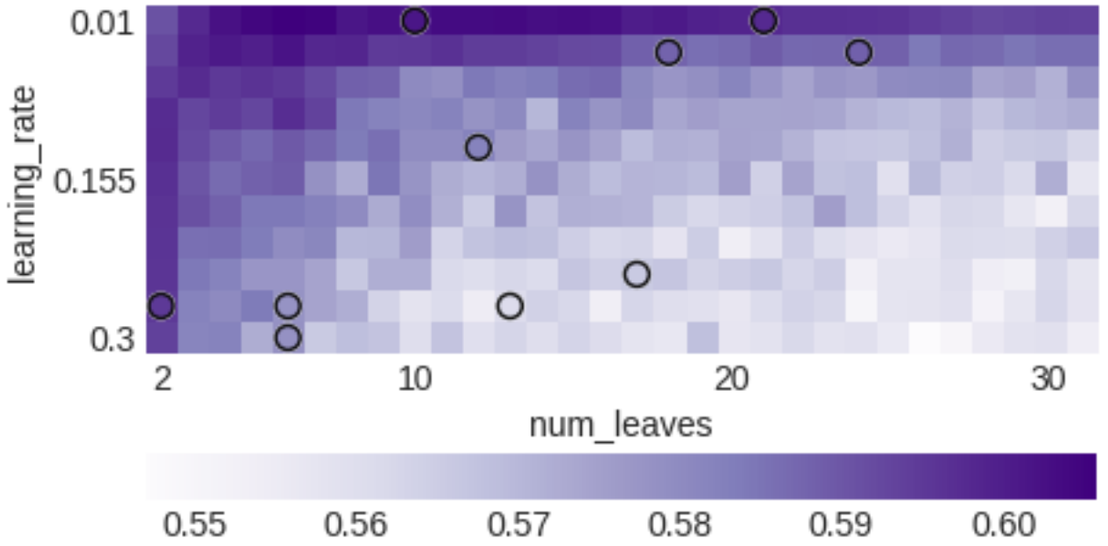
(тут есть «число итераций», можно передавать распределения параметров)

Перебор параметров

GridSearchCV



RandomizedSearchCV




```
import lightgbm as lgb
model = lgb.LGBMClassifier(n_estimators=100, subsample=0.75, colsample_bytree=0.75)

from sklearn.model_selection import GridSearchCV
parameters = {'num_leaves': np.arange(2, 32), 'learning_rate':np.linspace(0.01, 0.3, 11)}
clf = GridSearchCV(model, parameters, cv=5, scoring='roc_auc')
clf.fit(X, y)
```

Случайный поиск считают предпочтительным

(для «чёрных ящиков» с большим числом параметров)

Байесовская оптимизация *
(Bayesian model-based optimization)

Идея: есть априорная вероятностная модель целевой функции (ошибки – the objective function) – «surrogate probability model»

в отличие от функции ошибки она будет её приближением, её просто оптимизировать!

Способы представления SPM

Гауссовские процессы (Gaussian Processes)

Случайный лес (Random Forest Regressions)

Tree Parzen Estimators (TPE)

…

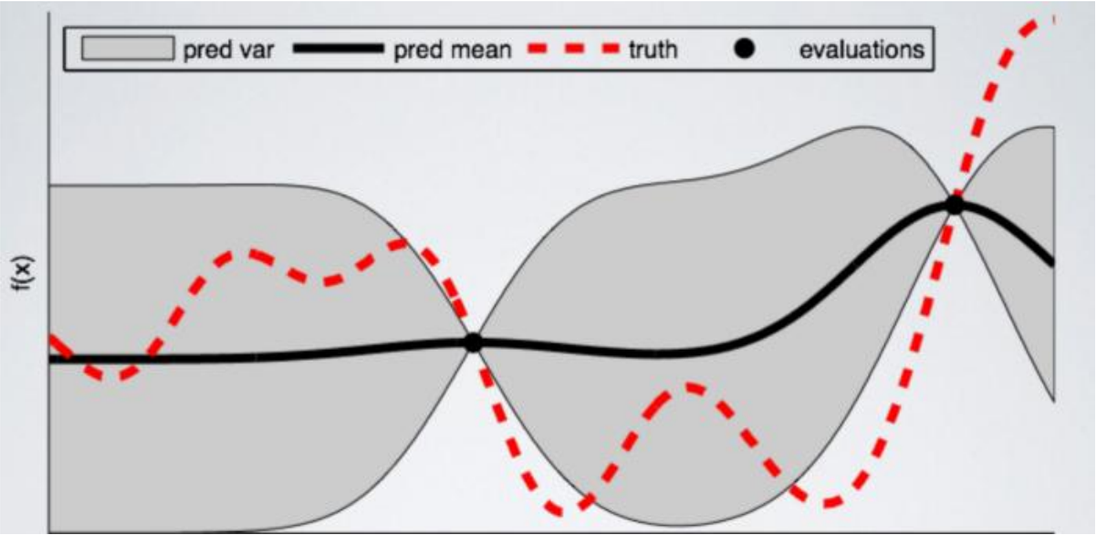
находим armax SPM / или точку для лучшего уточнения SPM

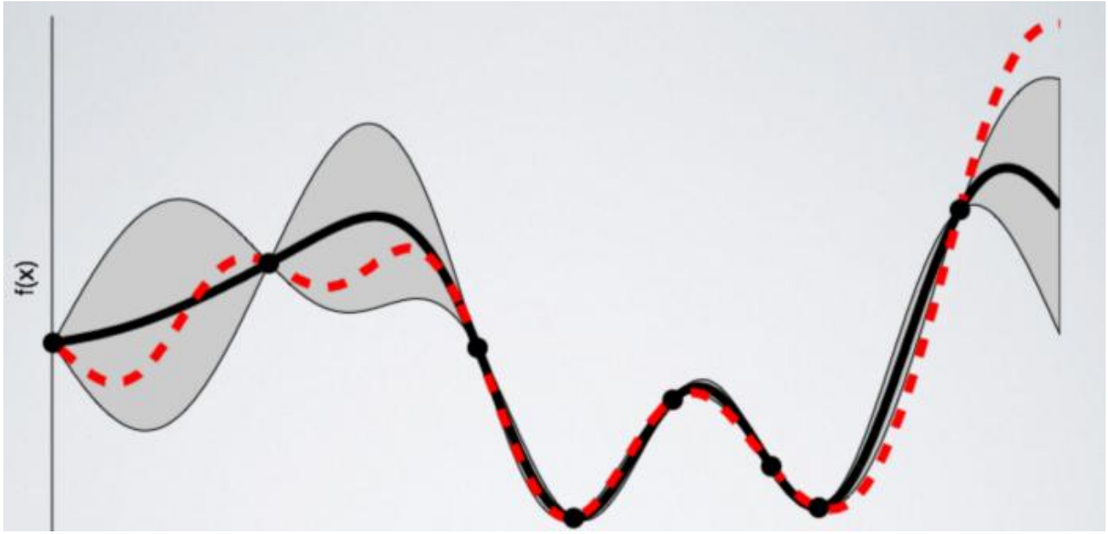
оцениваем значение в нём (ошибка при таких параметрах)

уточняем SPM

https://www.iro.umontreal.ca/~bengioy/cifar/NCAP2014-summerschool/slides/Ryan_adams_140814_bayesopt_ncap.pdf

Байесовская оптимизация *





Algorithm 1 Sequential Model-Based Optimization

Input: $f, \mathcal{X}, S, \mathcal{M}$
$\mathcal{D} \leftarrow \text{INITSAMPLES}(f, \mathcal{X})$
for $i \leftarrow |\mathcal{D}|$ **to** T **do**
  $p(y \mid \mathbf{x}, \mathcal{D}) \leftarrow \text{FITMODEL}(\mathcal{M}, \mathcal{D})$
  $\mathbf{x}_i \leftarrow \arg\max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x},\ p(y \mid \mathbf{x}, \mathcal{D}))$
  $y_i \leftarrow f(\mathbf{x}_i)$ ▷ Expensive step
  $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{x}_i, y_i)$
end for

Байесовская оптимизация *

можно почитать…

«Bayesian Optimization Primer»
https://app.sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf

Пример HyperOpt
https://github.com/WillKoehrsen/hyperparameter-optimization/blob/master/Introduction%20to%20Bayesian%20Optimization%20with%20Hyperopt.ipynb

«Algorithms for Hyper-Parameter Optimization»
https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

ссылки на библиотеки и код
https://www.jeremyjordan.me/hyperparameter-tuning/

Минутка кода

Как получить параметры модели

```
model.get_params()
```

```
{'boosting_type': 'gbdt',
 'class_weight': None,
 'colsample_bytree': 0.75,
 'importance_type': 'split',
 'learning_rate': 0.1,
 'max_depth': -1,
 'min_child_samples': 20,
 'min_child_weight': 0.001,
 'min_split_gain': 0.0,
 'n_estimators': 100,
 'n_jobs': -1,
 'num_leaves': 31,
 'objective': None,
 'random_state': None,
 'reg_alpha': 0.0,
 'reg_lambda': 0.0,
 'silent': True,
 'subsample': 0.75,
 'subsample_for_bin': 200000,
 'subsample_freq': 0}
```

Советы

Не забывайте указать метрику качества

`score`

а лучше несколько

Распараллеливание

`n_jobs=-1`

Можно сделать вычисления устойчивым к ошибкам

`error_score=0`

Оптимизировать целый пайплайн!

Итог

Правильная организация контроля – важная часть обучения

Помним 3 золотых правила:

- моделируем реальность,
- не допускаем утечек,
- случайность

CV для

- **контроля качества**
- **формирования ответов на обучении**
- ансамблировании **будет**
- **настройка гиперпараметров**

Кривые качества

- validation curves (от значений)
- learning curves (от объёма выборки)

Ссылки

- **презентация по sklearn**

https://github.com/Dyakonov/IML/blob/master/IML2018_06_scikitlearn_10.pdf

– есть код для различных организаций контроля