«Машинное обучение»

Метрические методы

Александр Дьяконов

21 сентября 2021 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

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

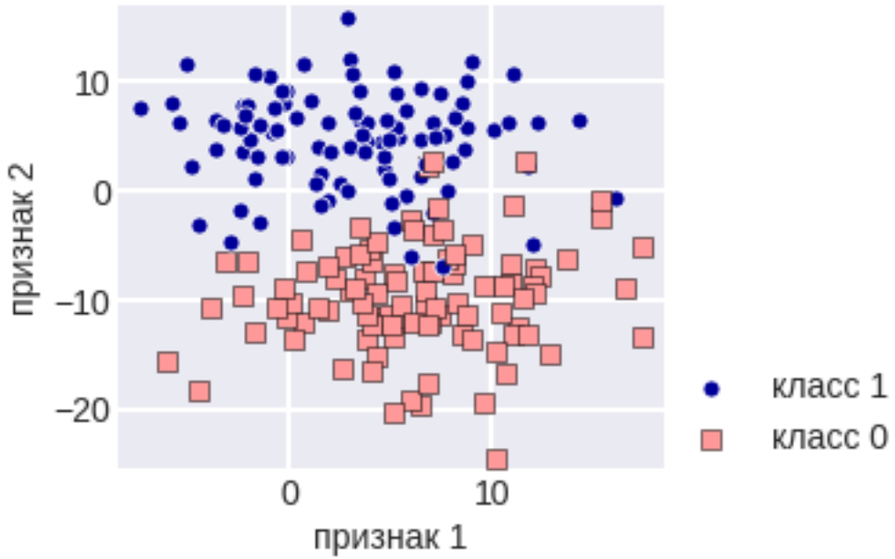
Примеры:

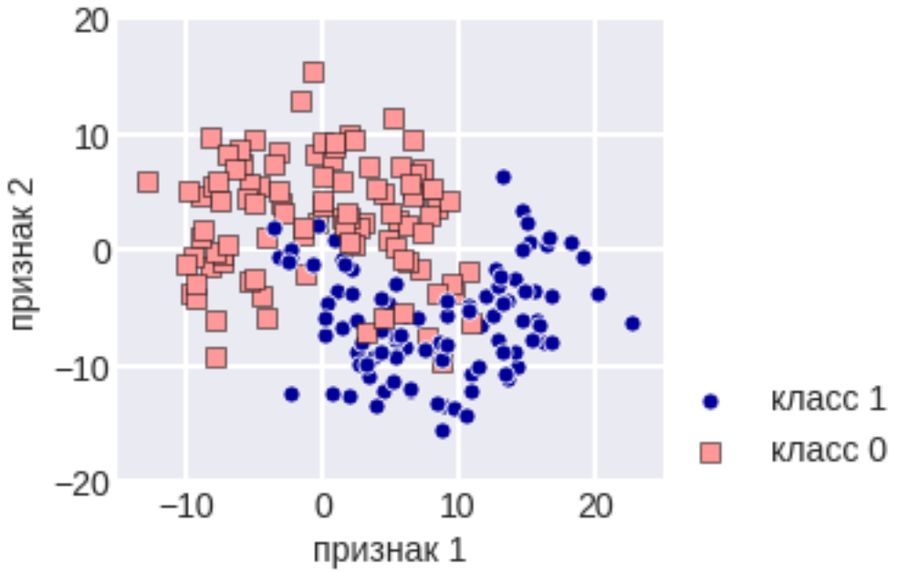
- Nearest centroids algorithm / Distance from Means
- kNN (Nearest Neighbor)

ещё называют:

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

Модельные задача классификации





на них будем показывать работу алгоритмов

Ближайший центроид (Nearest centroid algorithm)

Задача классификации на непересекающиеся классы с вещественными признаками:

$$Y = \{1, 2, \ldots, l\}, \quad x_i \in \mathbb{R}^n$$

центроиды:

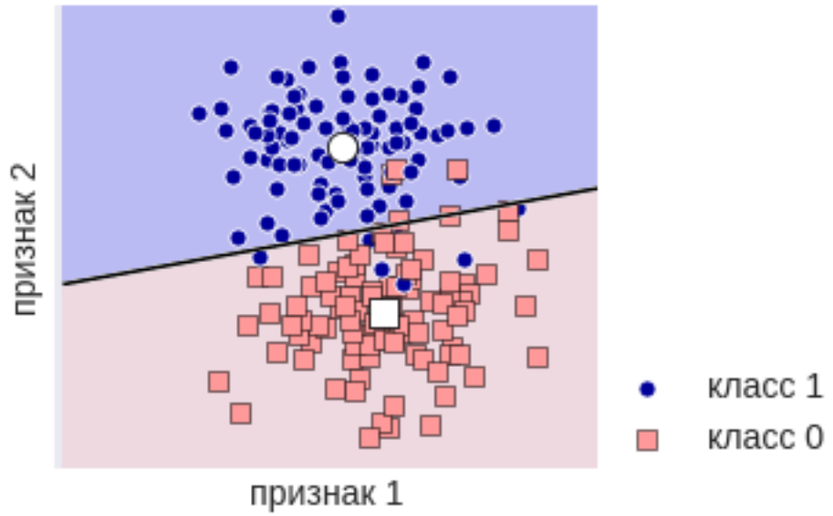
$$c_j = \frac{1}{|\{i : y_i = j\}|} \sum_{i: y_i = j} x_i$$

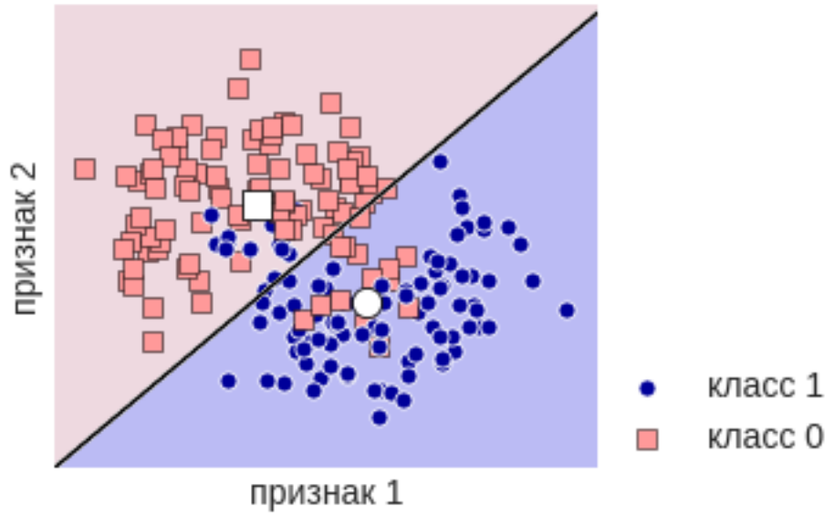
классификация:

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

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 × 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)
```

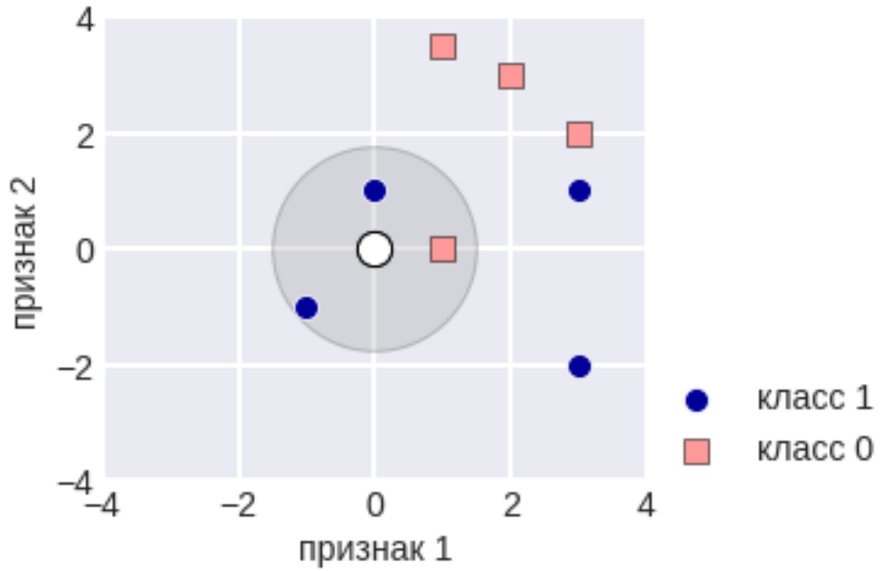
есть параметр

```
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)

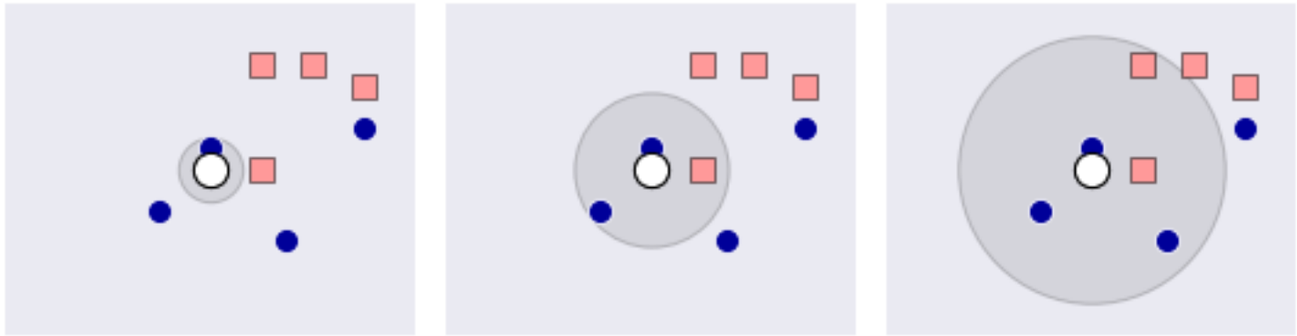
Окрестность

Если X – метрическое пространство с метрикой ρ,
пусть нумерация объектов такая, что

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

В методе k ближайших соседей (kNN = k nearest neighbours)
окрестность выбирается

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



Окрестность

Есть также «Fixed-Radius Near Neighbor»

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

про него не будем подробно

Метод k ближайших соседей (kNN)

Гиперпараметр k можно выбрать на скользящем контроле дальше

Ещё гиперпараметры (потом):

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

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

формально нет обучения – храним всю выборку

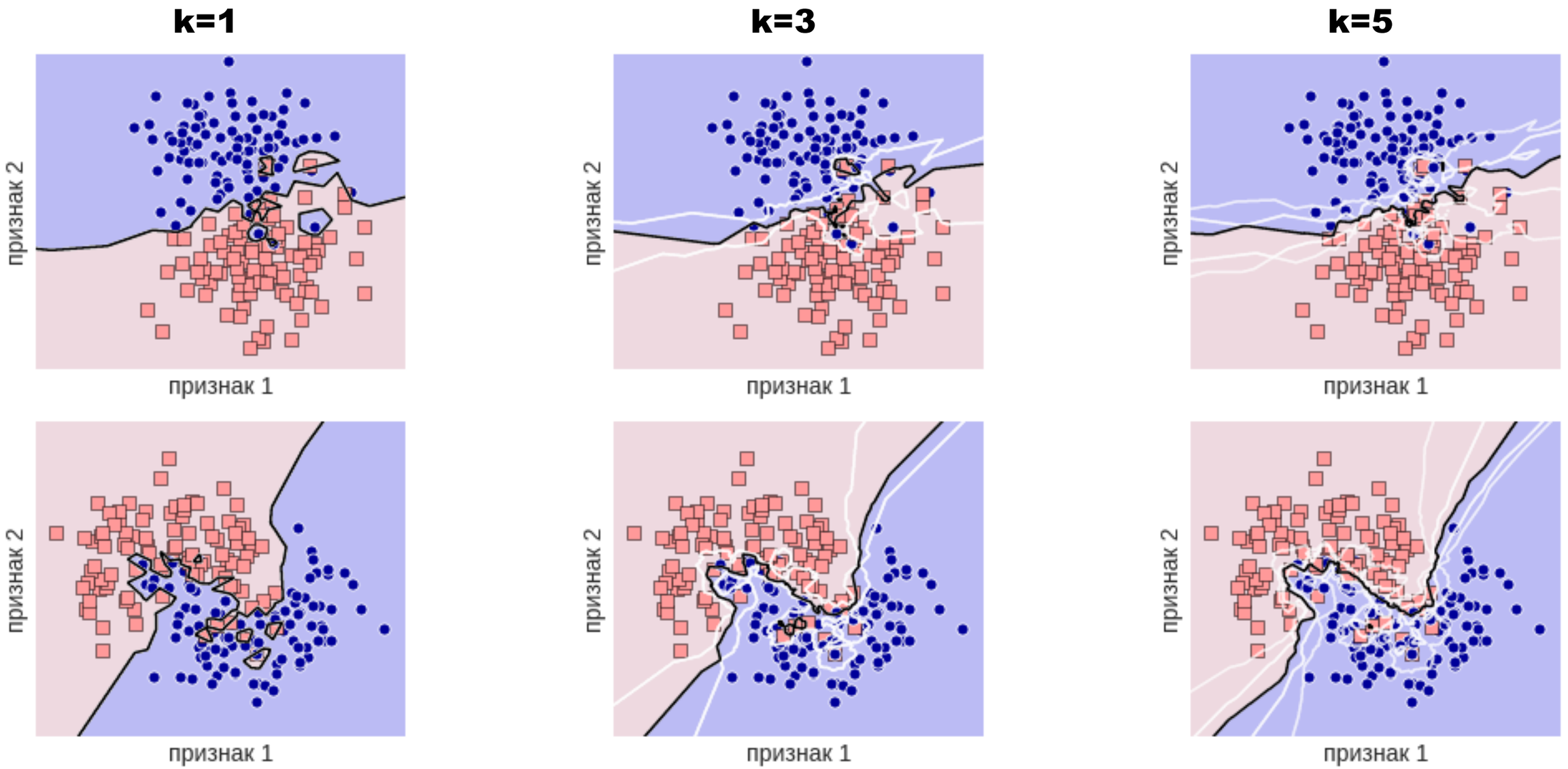
работа алгоритма – просматриваем всю выборку

(+ вычисляем расстояние до каждого объекта обучения)

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



Решение модельной задачи при разном числе соседей

Как увидим дальше, k отвечает за «сложность модели»

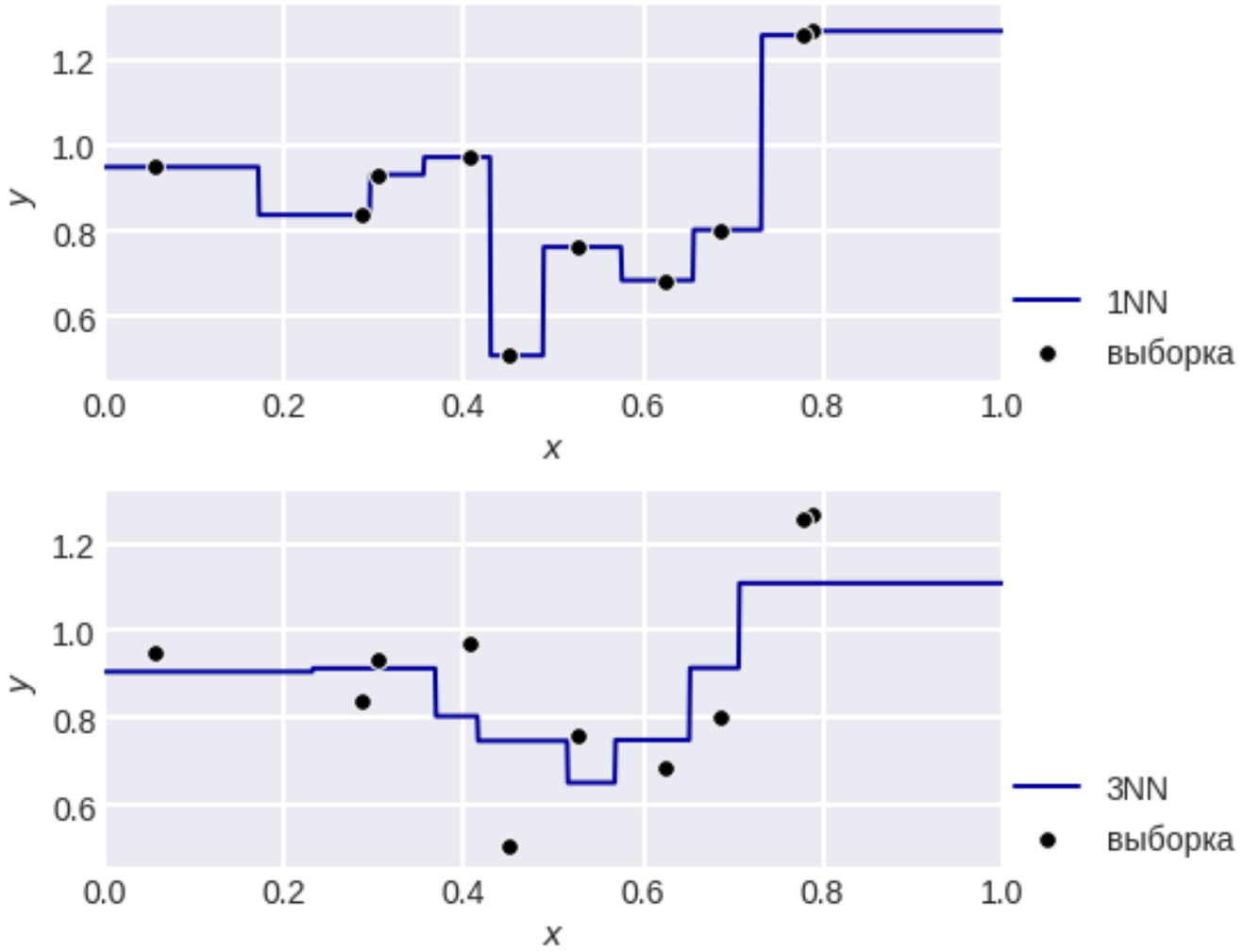
Минутка кода

```
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

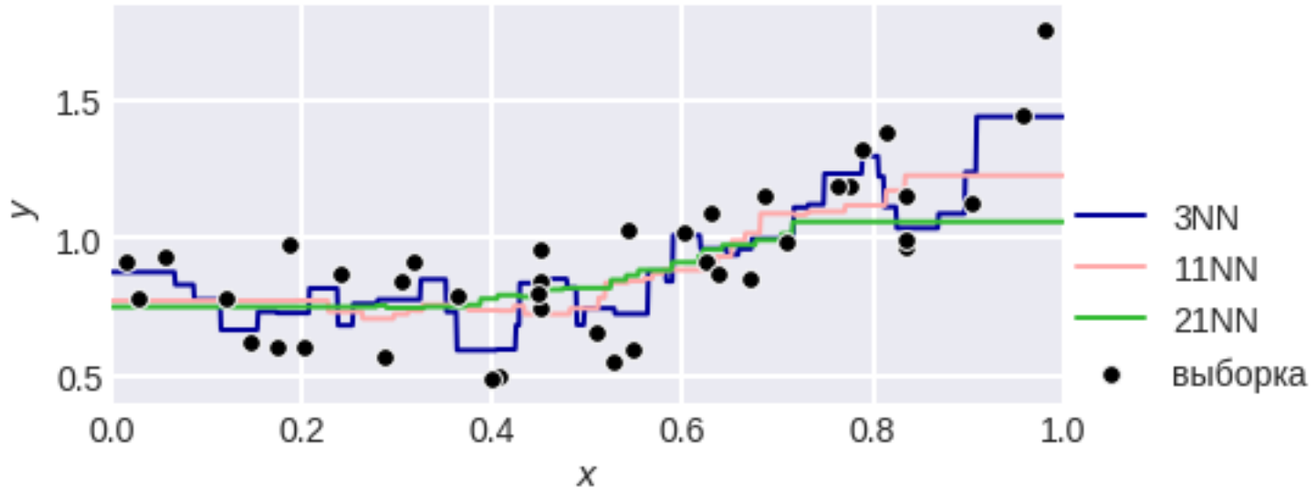
Метод ближайшего соседа

обобщается на регрессию



Метод ближайшего соседа

обобщается на регрессию



Минутка кода

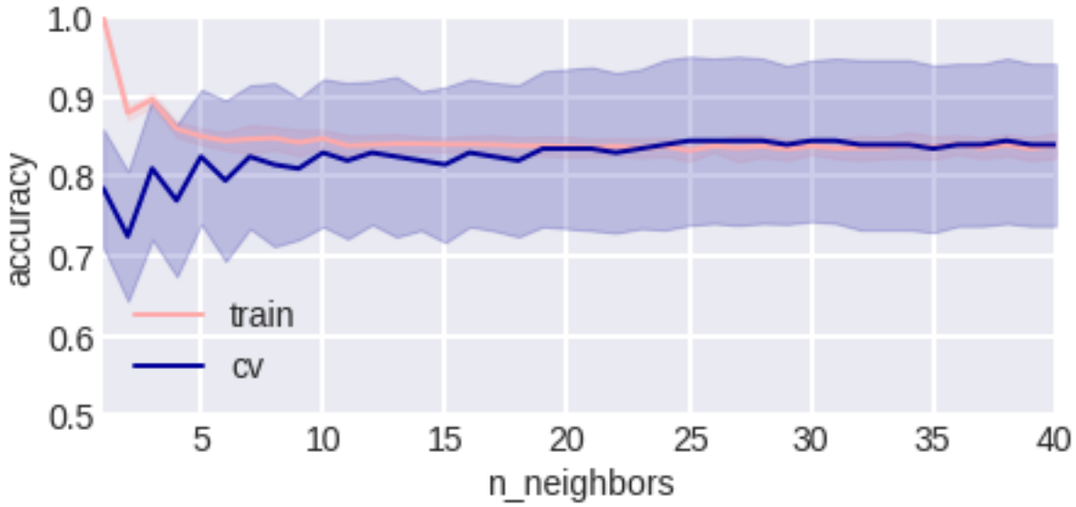
```
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=3) # kNN-регрессия
model.fit(x_train, y_train) # обучение
a = model.predict(x_test) # ответ
```

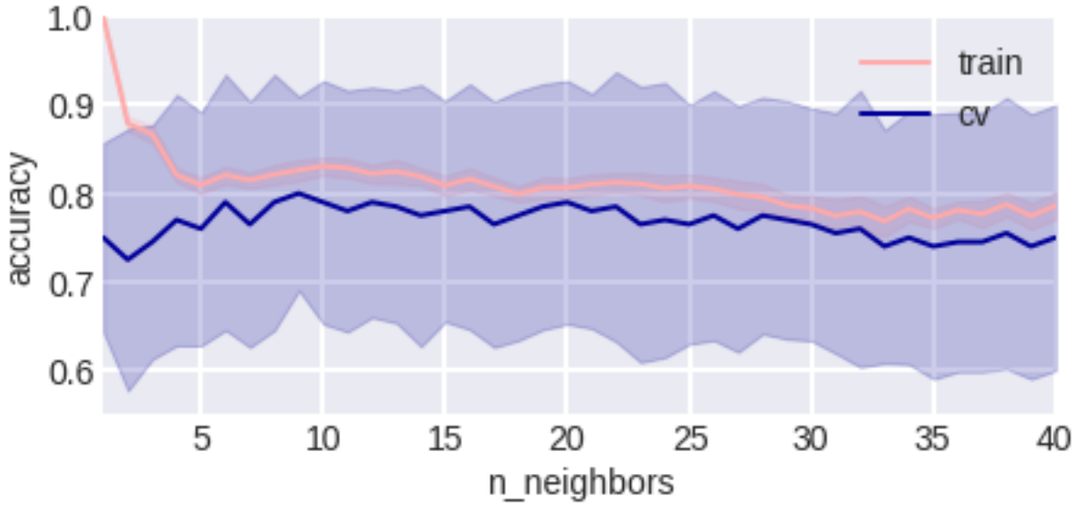
Подбор гиперпараметров специальными методами контроля

```
# 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)
```



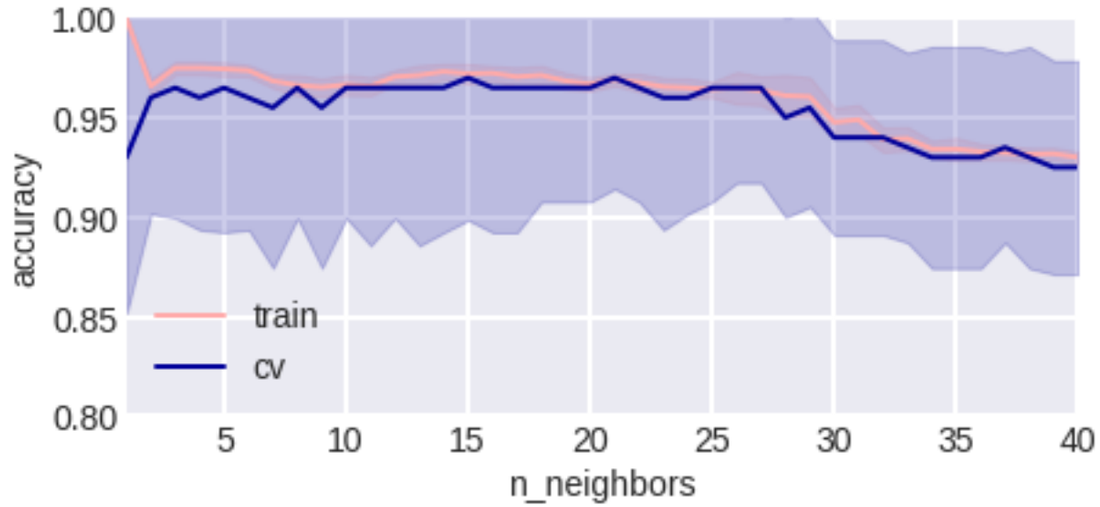


Как был получен второй график?

Подбор гиперпараметров специальными методами контроля

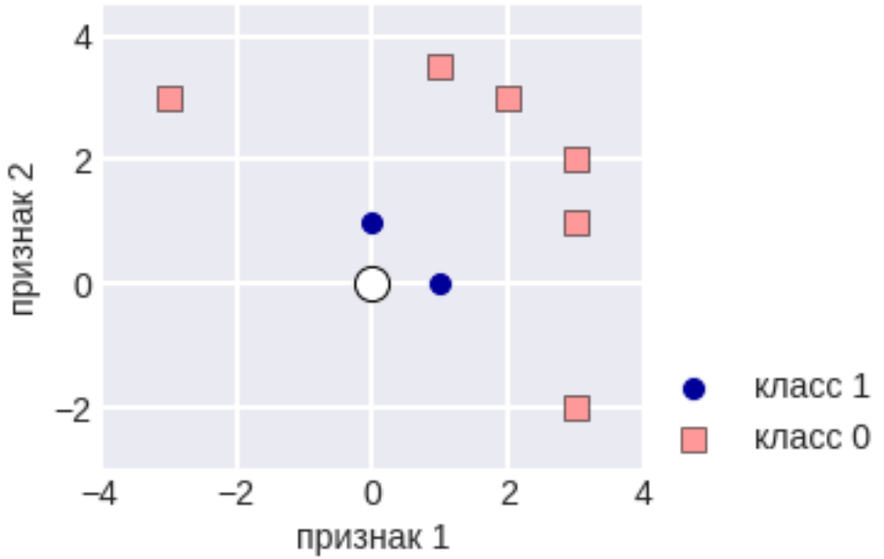
Как был получен второй график?

Сделали дисбаланс классов –
стало невыгодно делать большие k



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

Проблема классического kNN



близкие соседи должны быть важнее

Весовые обобщения kNN

классика:

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

обобщение:

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

разные весовые схемы:

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

Весовые схемы

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

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

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

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

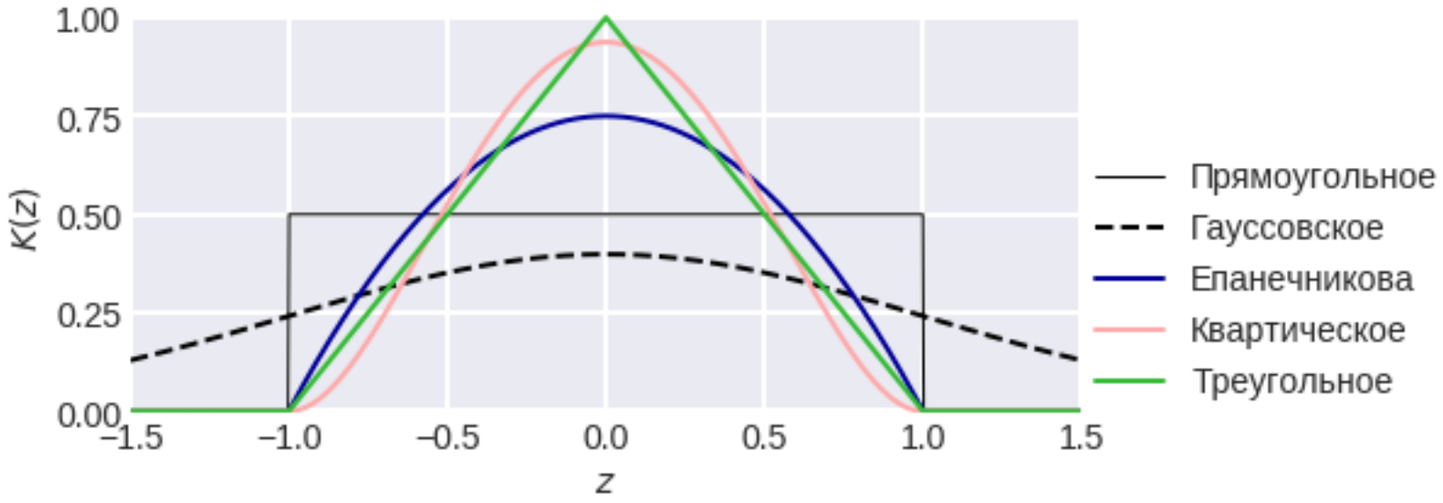
$$w_t = K\left(\frac{\rho(x, x_t)}{h(x)}\right)$$

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

Различные виды ядер (одномерных)



Прямоугольное / tophat

$$K(z) = \frac{1}{2} I[|z| \le 1]$$

Треугольное / linear

$$K(z) = \max(\min(1-z, 1+z), 0)$$

Гауссовское / gaussian

$$K(z) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^T z}{2}\right)$$

Квартическое

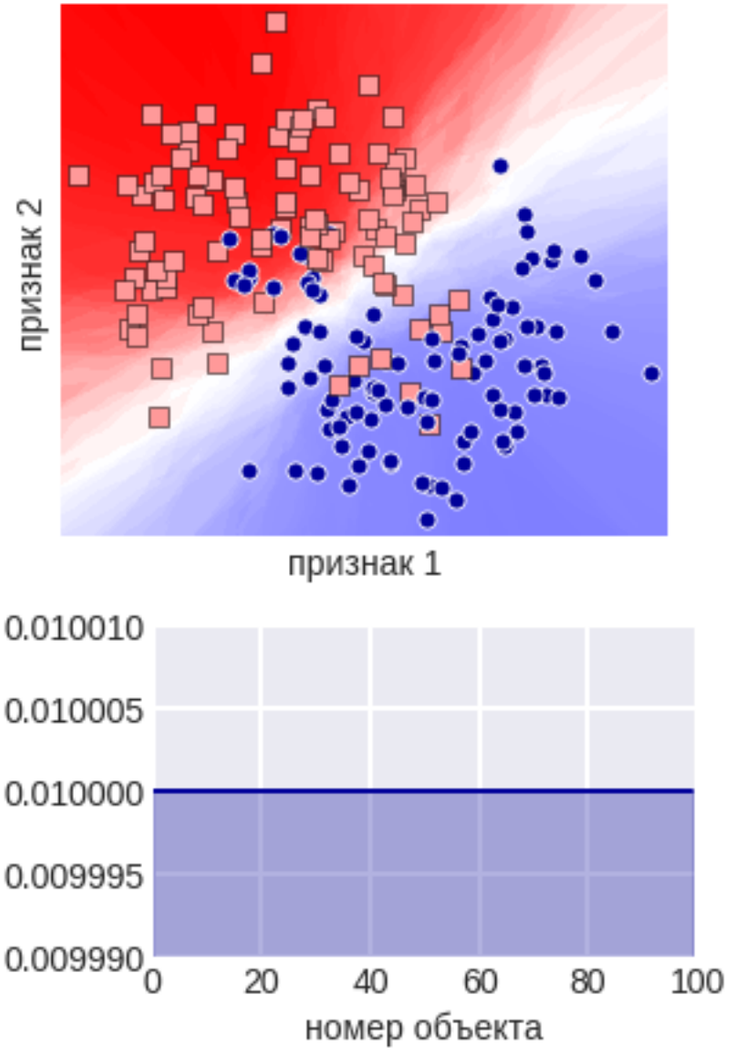
$$K(z) = \frac{15}{16}(1-z^2)^2 I[|z| \le 1]$$

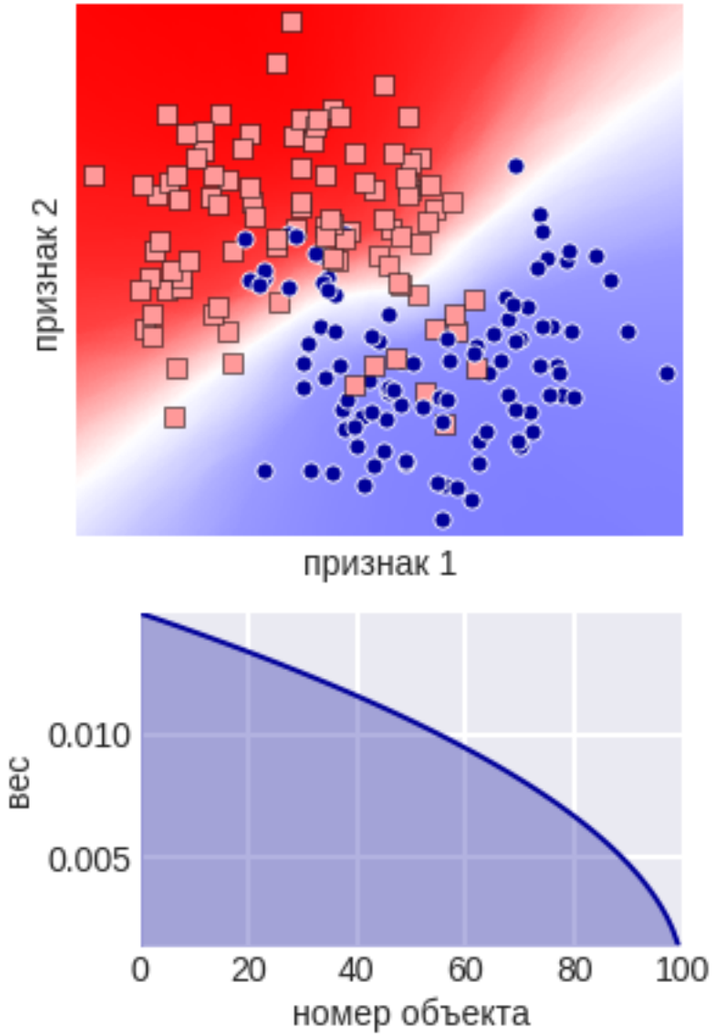
Епанечникова / epanechnikov

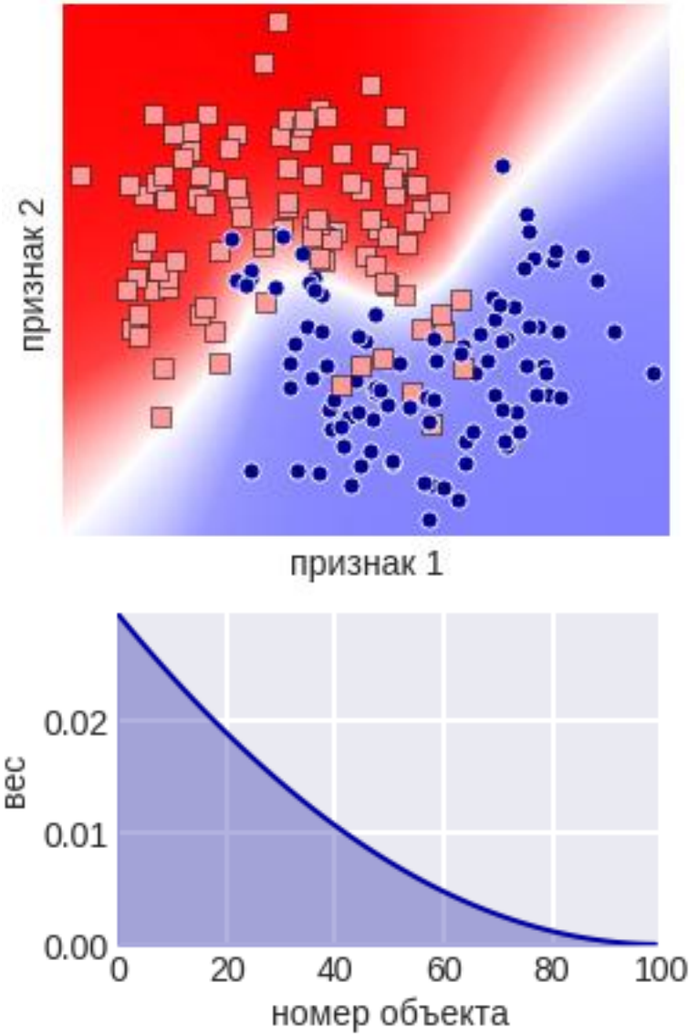
$$K(z) = \frac{3}{4}(1-z^2) I[|z| \le 1]$$

https://scikit-learn.org/stable/auto_examples/neighbors/plot_kde_1d.html

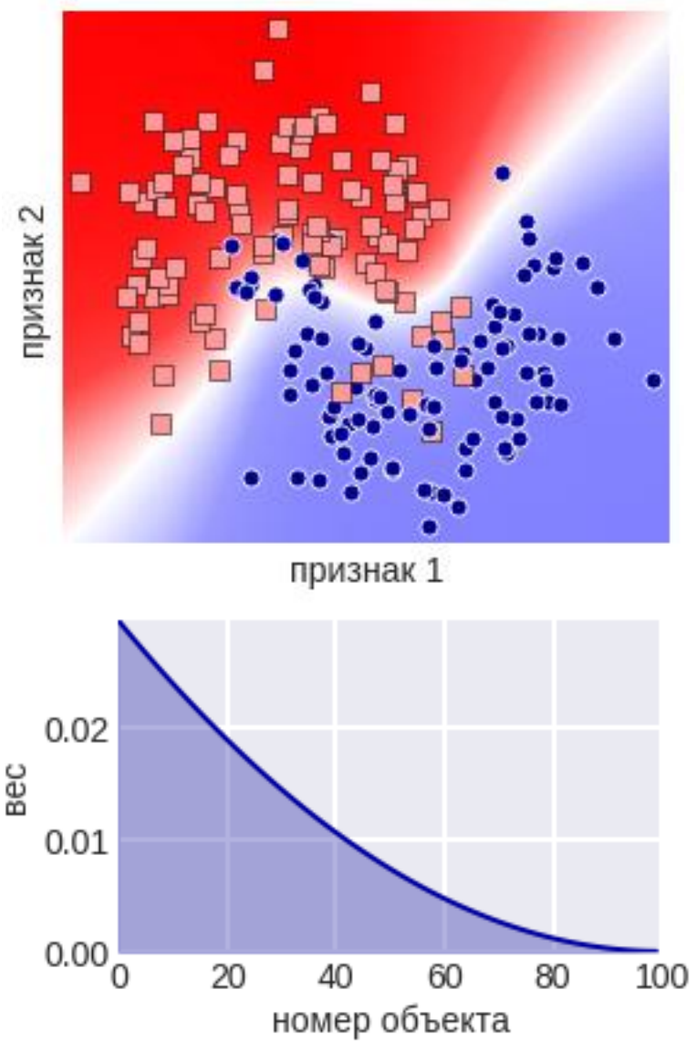
Весовые обобщения kNN

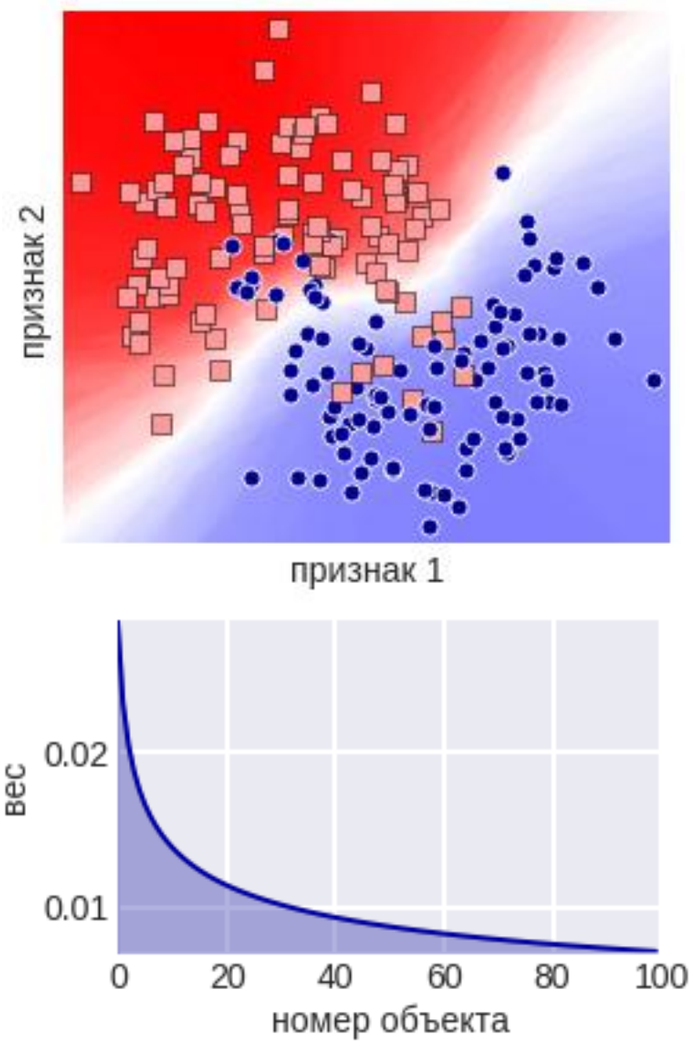


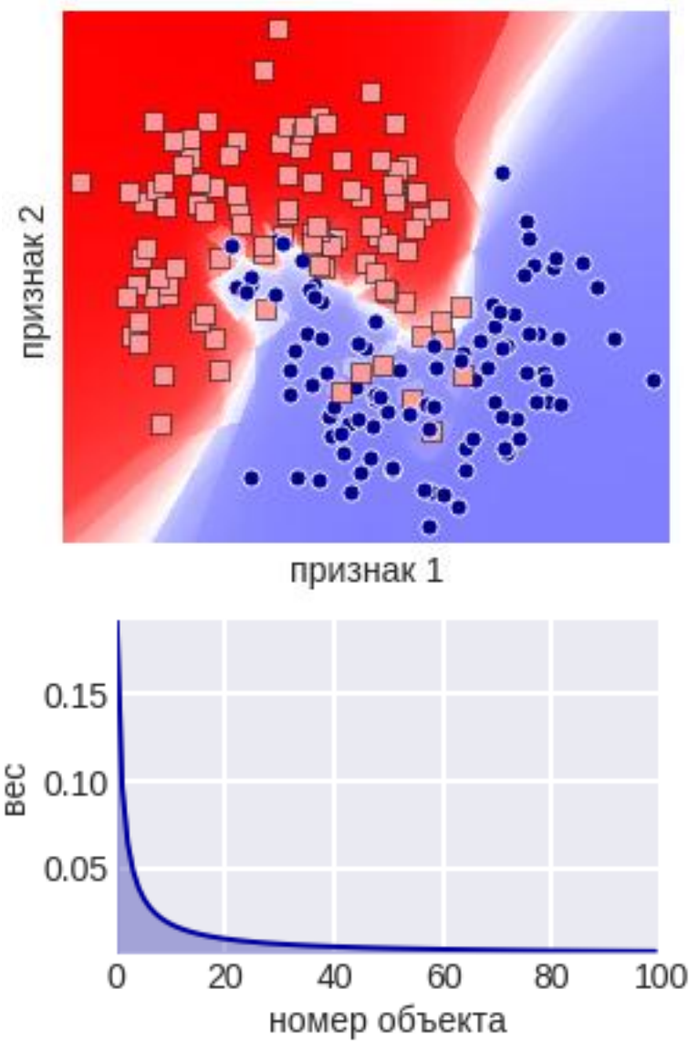




Весовые обобщения kNN



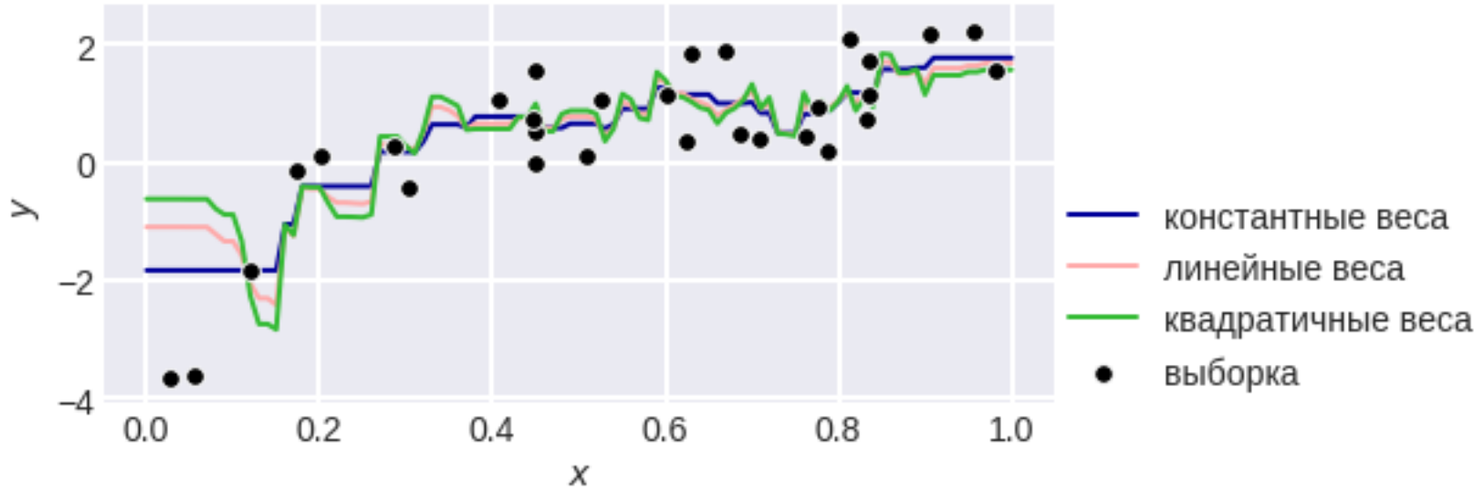




Весовые обобщения в регрессии

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

пример для 5NN



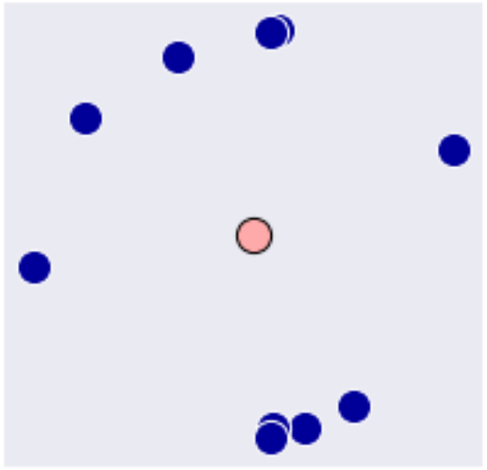
Эффект почти не заметен, дальше будет обобщение – регрессия Надарая-Ватсона

Проблема выбора метрики

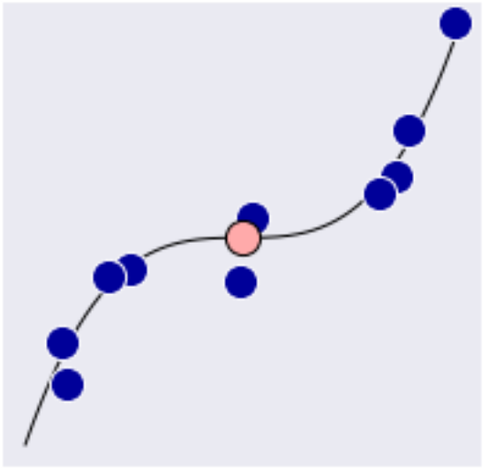
- **зависимость от масштаба**
  - нормировка признаков
  - однородные признаки
  - смесь метрик
- **можно выбирать не метрику, а близость**
  - пример: косинусная мера сходства
- **часто выбор функции расстояния, как ни странно, довольно прост…**

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)

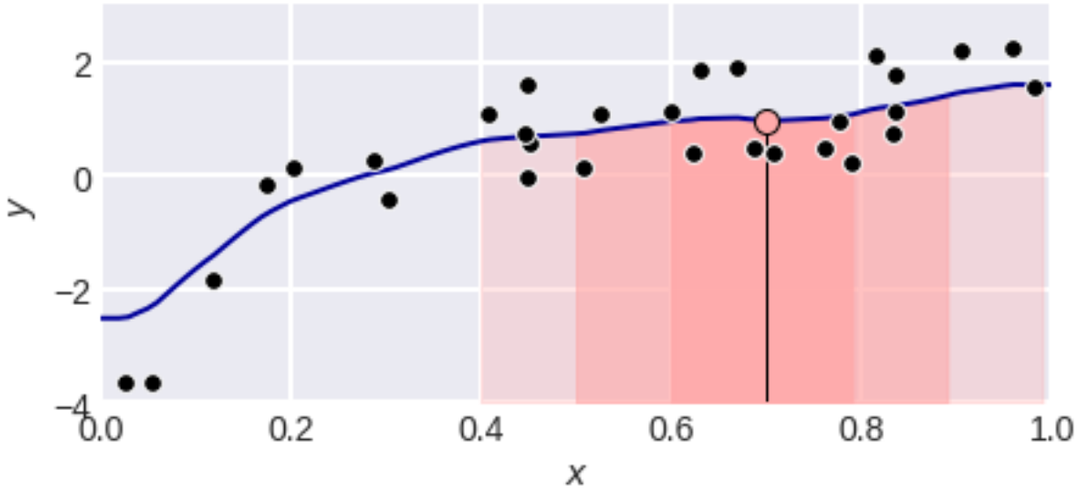
Метрические алгоритмы

+ 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)

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

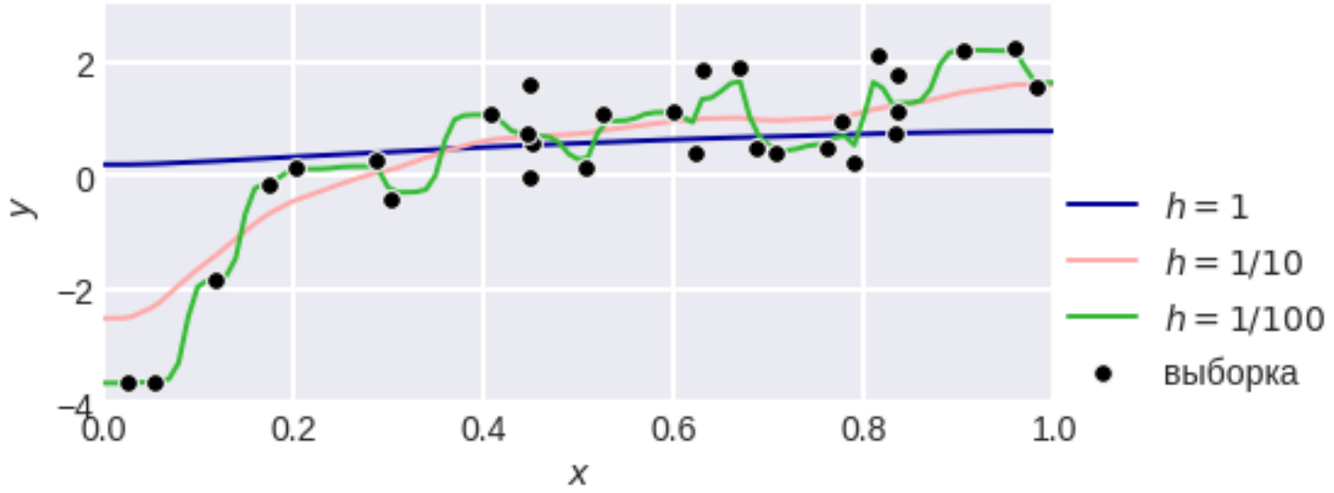
Смысл весов – чем ближе объект обучения, тем скорее ответ похож на его метку

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

Ядро с шириной h.

Здесь также как выше… (про функции ядра)

Регрессия Надарая-Ватсона (Nadaraya-Watson regression)



пример регрессии при разных значениях ширины ядра

Регрессия Надарая-Ватсона (Nadaraya-Watson regression)

Смысл:

ответ алгоритма – решение оптимизационной задачи

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

Свойства:

+ хорошее решение задачи сглаживания

– не решает задачи экстраполяции

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

Ссылки

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

Дополнения

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**
- **Махаланобиса**

- **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_∞)

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

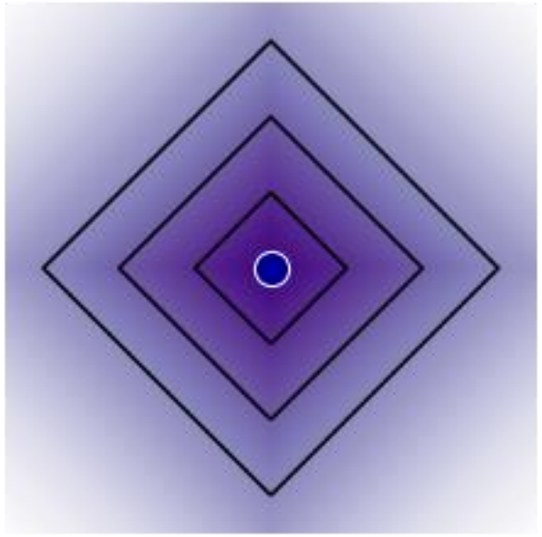
Частный случай – Манхэттенская (L_1)

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

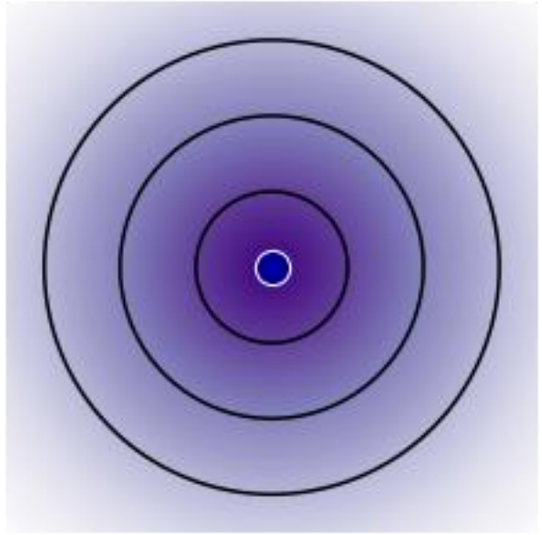
здесь $x = (x_1, \ldots, x_n), \quad z = (z_1, \ldots, z_n)$

Различные метрики

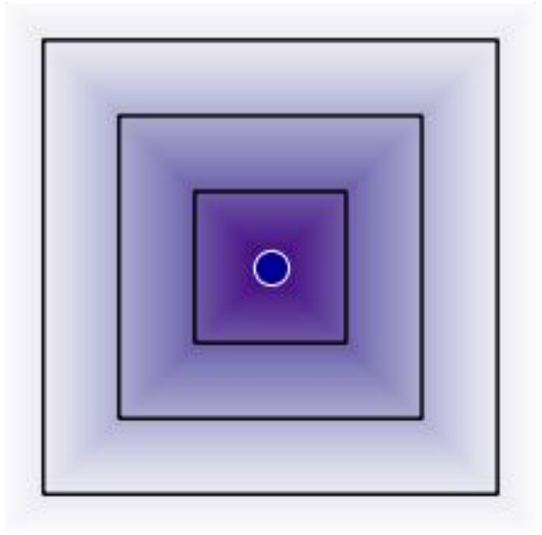
$$\left(| x_1 - z_1 |^p + | x_2 - z_2 |^p \right)^{1/p}$$



L_1



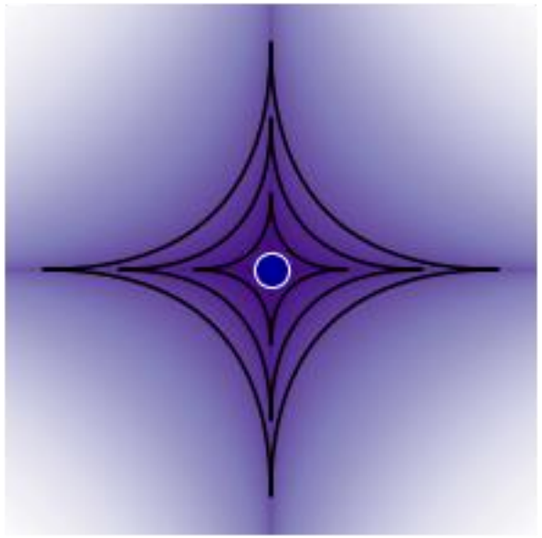
L_2



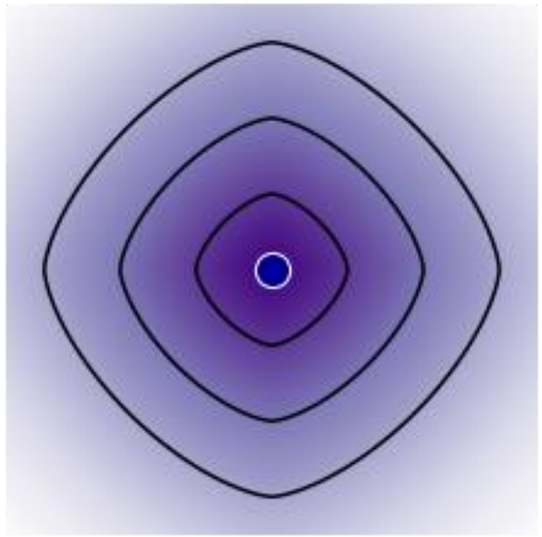
L_∞

Различные метрики

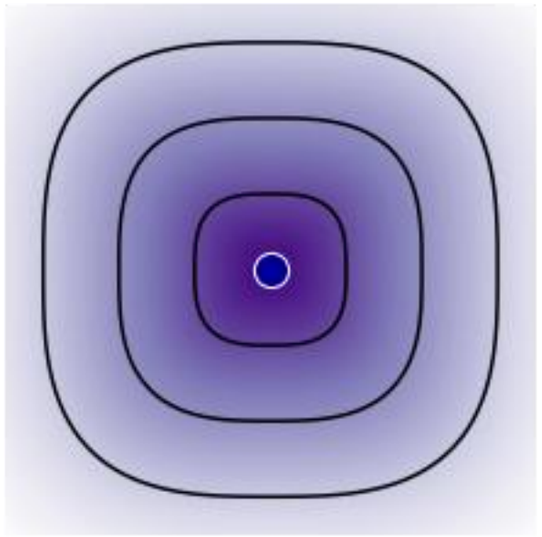
$$\left(| x_1 - z_1 |^p + | x_2 - z_2 |^p \right)^{1/p}$$



$L_{0.5}$



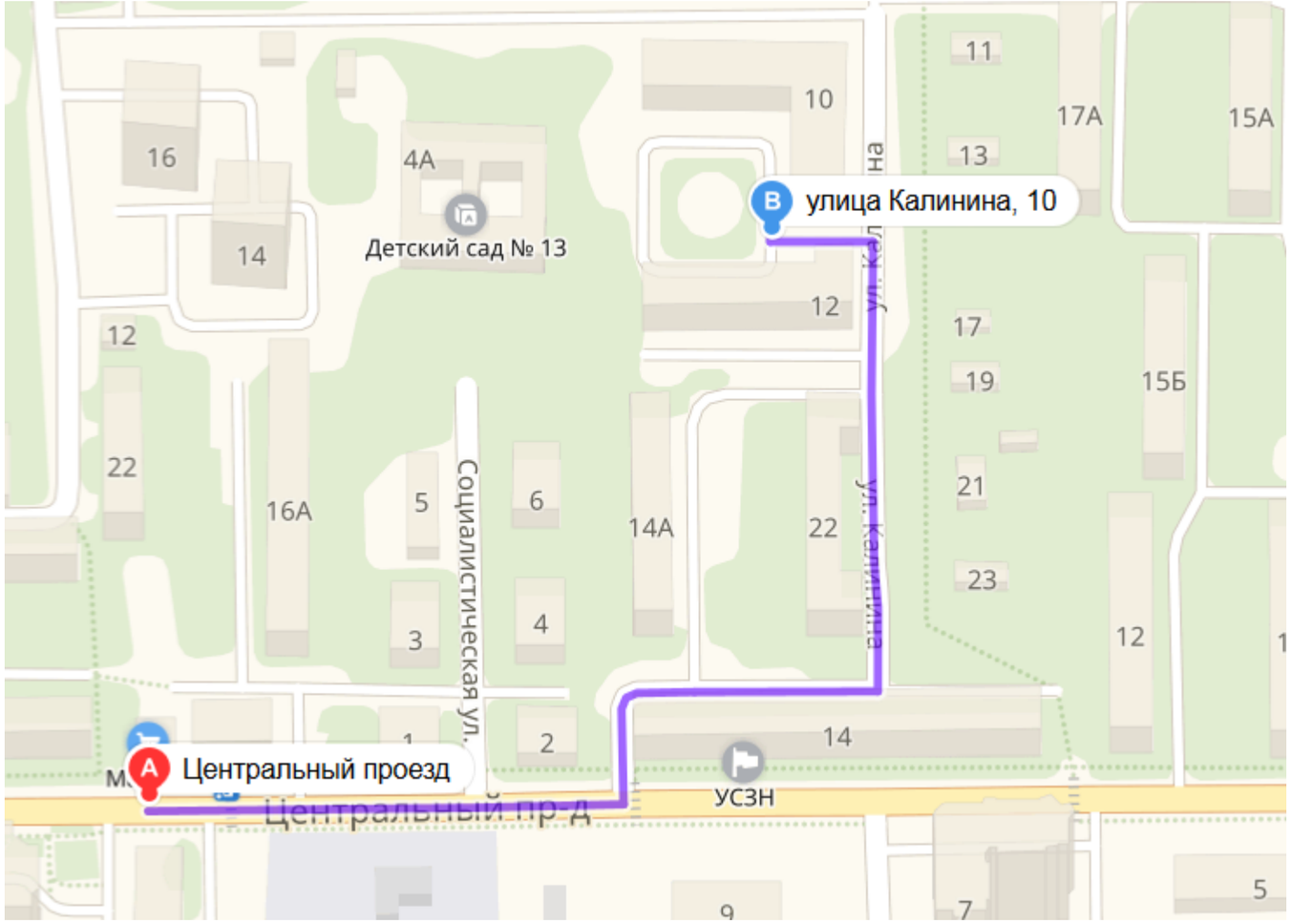
$L_{1.5}$



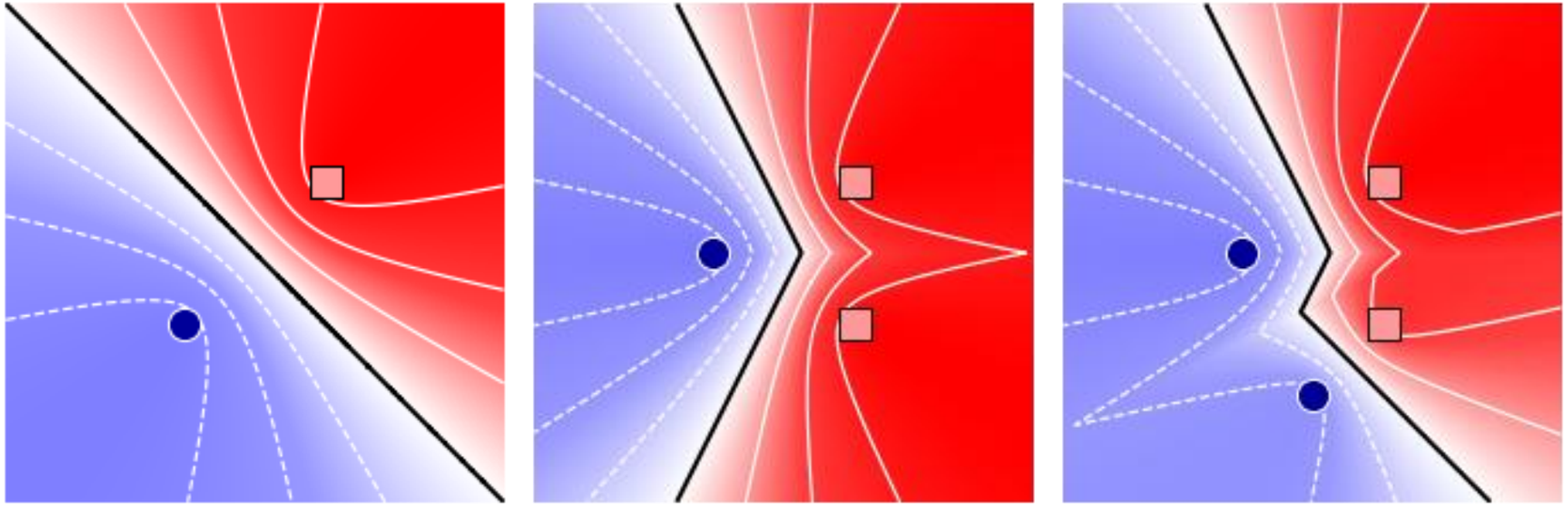
L_3

что такое L_0?

Различные метрики



Разделяющие поверхности L_2



Разделяющие поверхности L_1

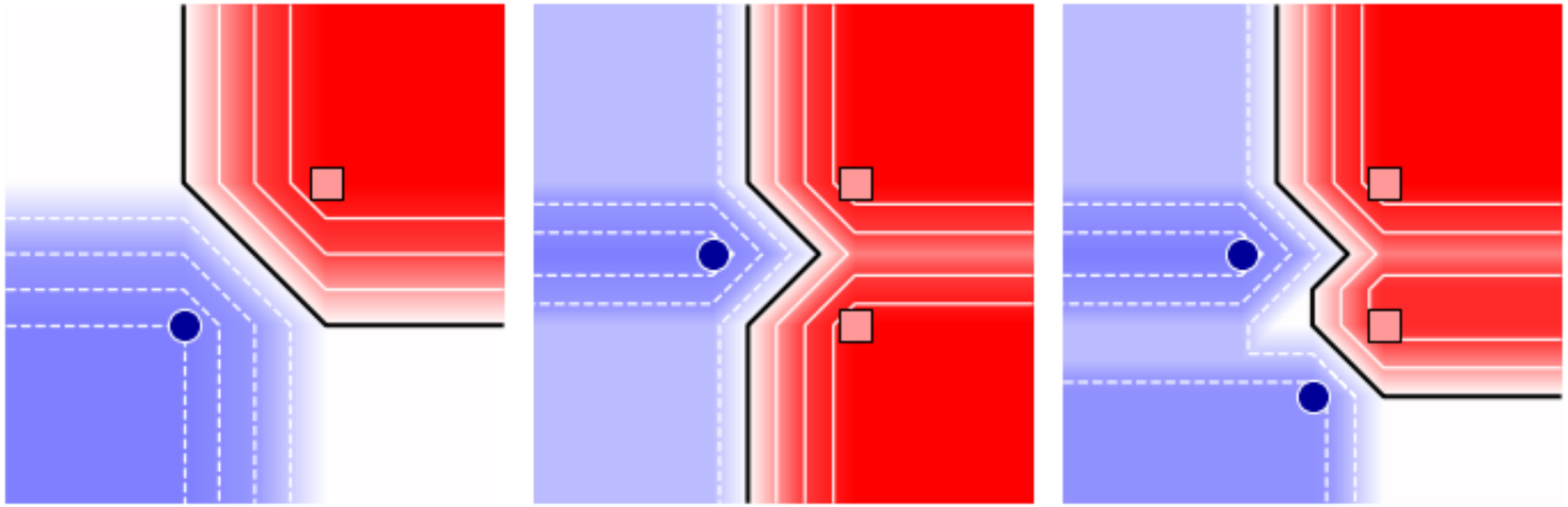
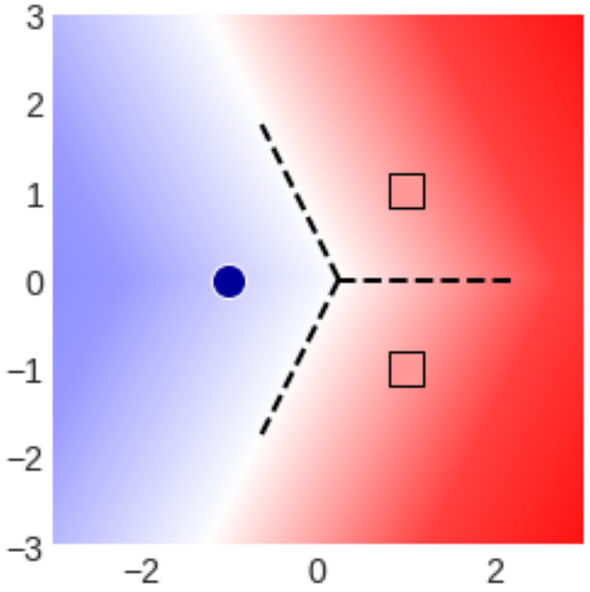
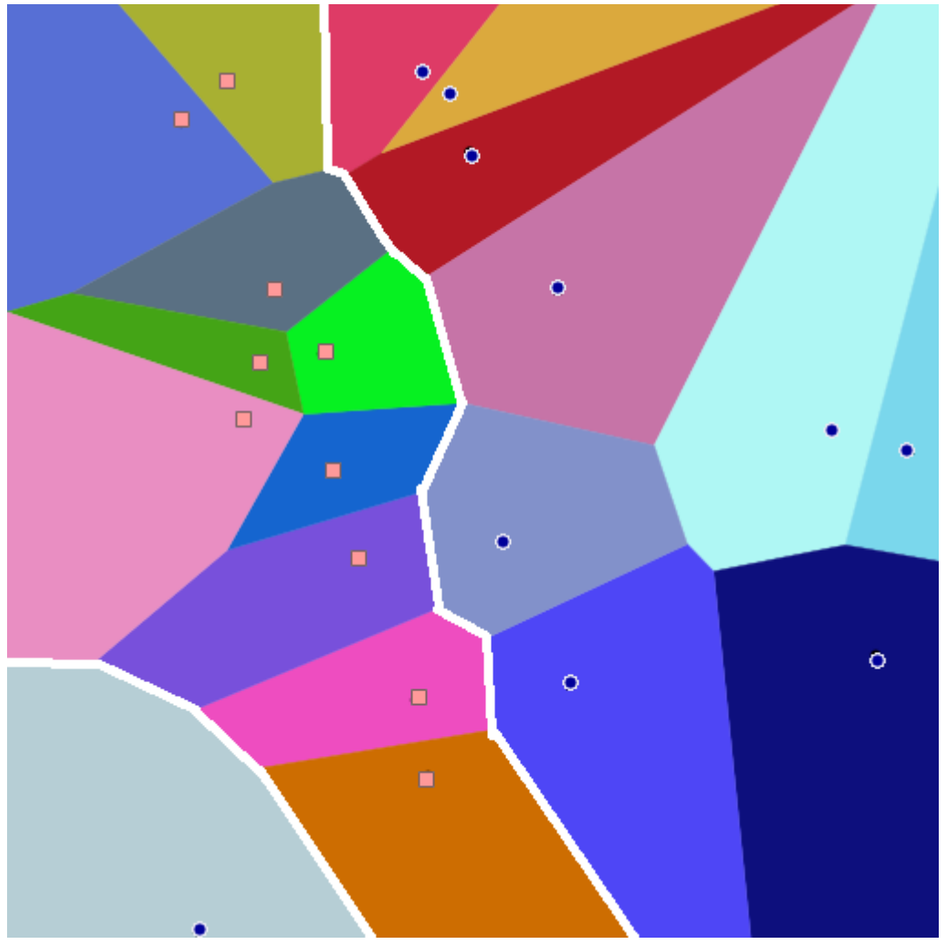


Диаграмма Вороного (Voronoi diagram)

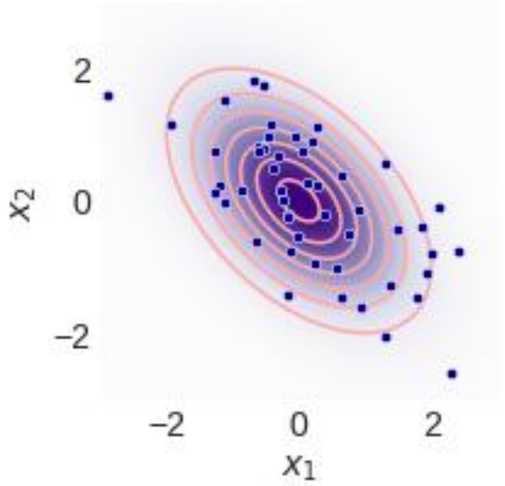




https://en.wikipedia.org/wiki/Voronoi_diagram

Расстояние Махаланобиса (Mahalanobis distance)

– евклидово расстояние после преобразования $x \to \varphi(x) = \Sigma^{-1/2}(x - \mu)$

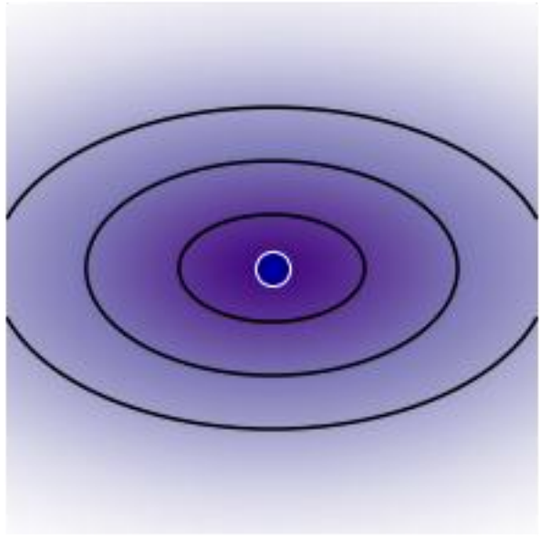


стандартизует нормальные данные

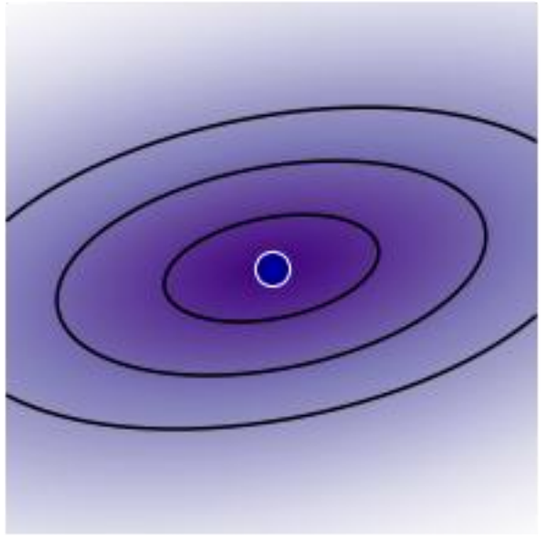
$$\text{norm}(\mu, \Sigma) \to \text{norm}(0, I)$$

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

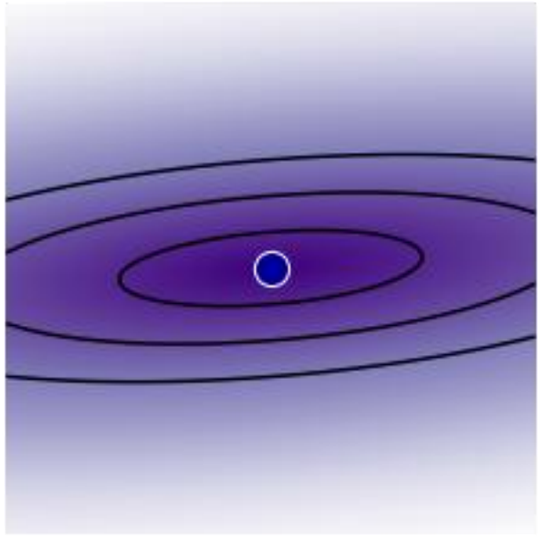
Расстояние Махаланобиса



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



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



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

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 (не расстояние)

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

если работать с нормированными векторами,
достаточно рассматривать скалярное произведение

Расстояния

Расстояние Джаккарда (на множествах)

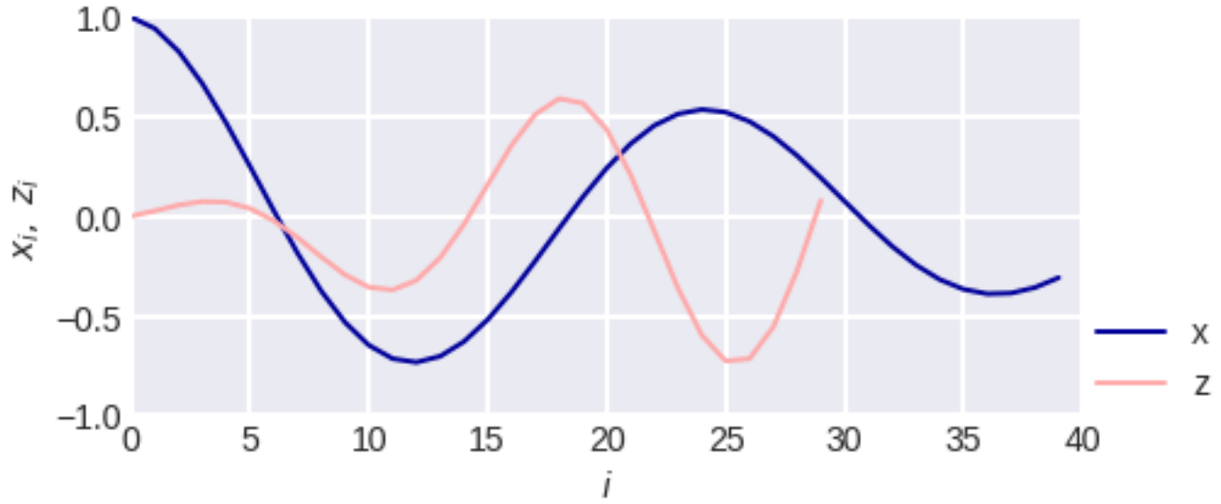
$$1-\frac{|X\cap Z|}{|X\cup Z|}$$

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…

Метрики на временных рядах



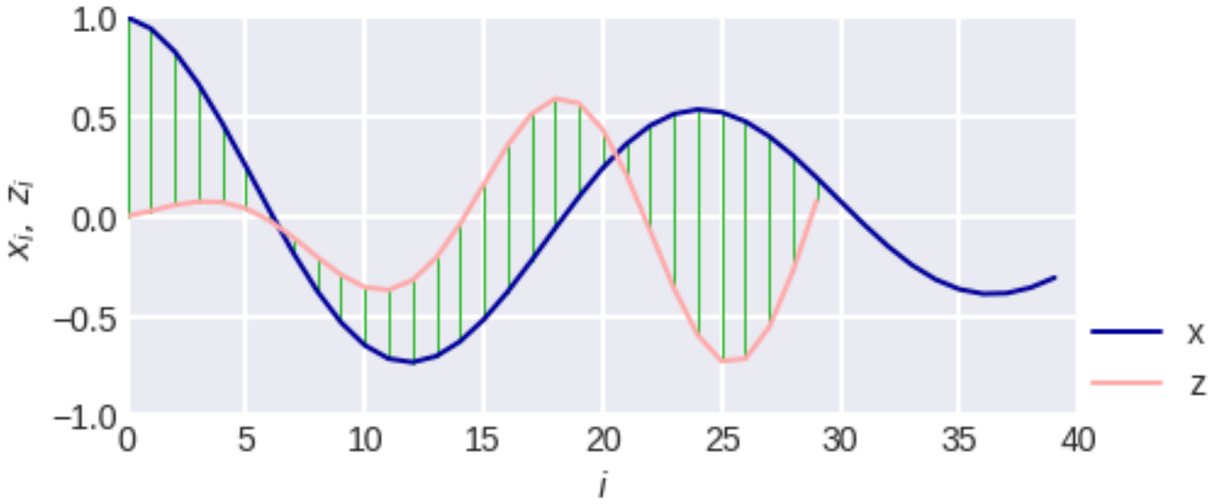
как ввести расстояние?

Ряды могут быть разной длины…

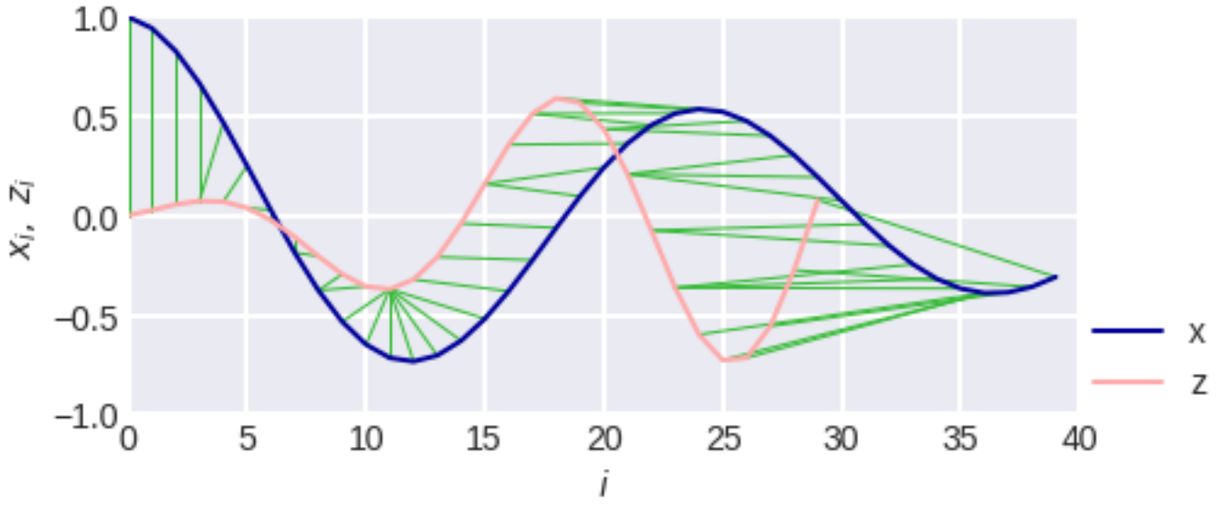
Как оценить похожесть формы?

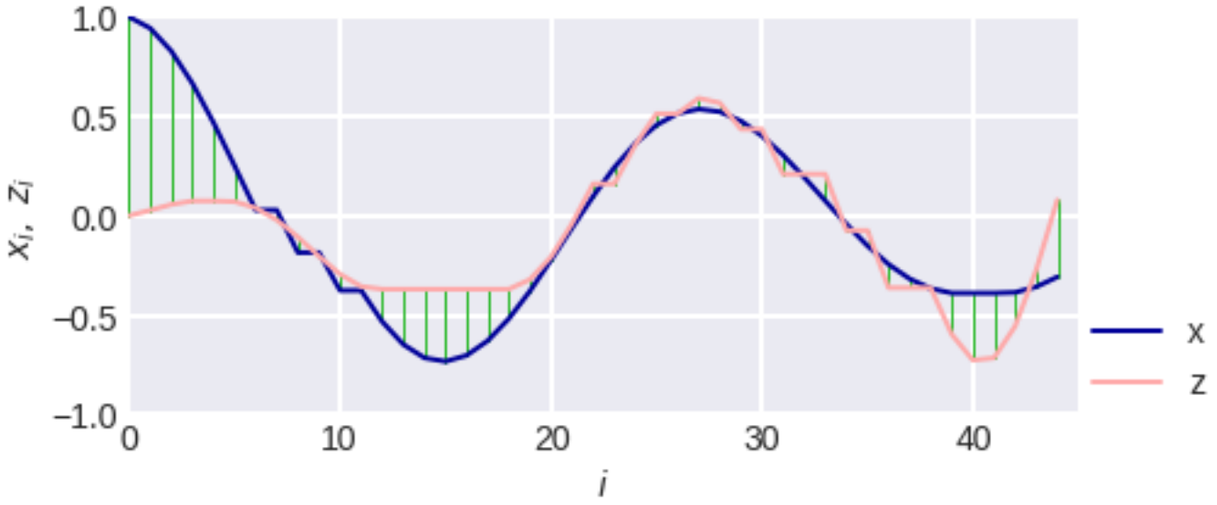
Метрики на временных рядах

Евклидово расстояние



DTW = Dynamic time warping





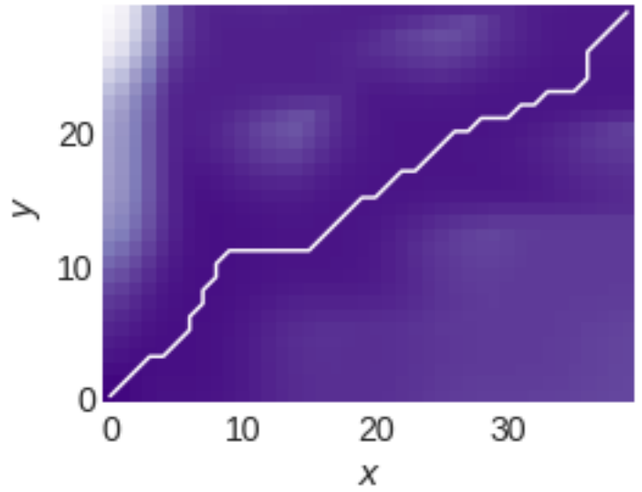
Метрики на временных рядах

матрица расстояний между точками

$$\begin{array}{c|cccc} & x_1 & x_2 & \cdots & x_m \\ \hline z_1 & |x_1 - z_1| & |x_2 - z_1| & \cdots & |x_m - z_1| \\ z_2 & |x_1 - z_2| & |x_2 - z_2| & \cdots & |x_m - z_2| \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_n & |x_1 - z_n| & |x_2 - z_n| & \cdots & |x_m - z_n| \end{array}$$

Надо найти соответствие…

$$1 = g(1) \le g(2) \le \ldots \le g(k) = m$$
$$\{g(1), g(2), \ldots, g(k)\} = \{1, 2, \ldots, m\}$$
$$1 = h(1) \le h(2) \le \ldots \le h(k) = n$$
$$\{h(1), h(2), \ldots, h(k)\} = \{1, 2, \ldots, n\}$$
$$\sum_{i=1}^{k} |x_{g(i)} - z_{h(i)}| \to \min$$



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:

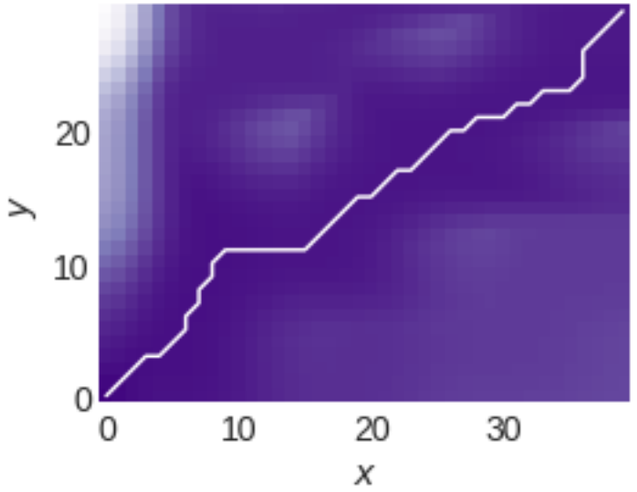
$$\mathrm{DTW}(x[:0], z[:0]) = 0$$
$$\mathrm{DTW}(x[:i], z[:0]) = \mathrm{DTW}(x[:0], z[:i]) = \infty \quad \text{при} \quad i > 0$$

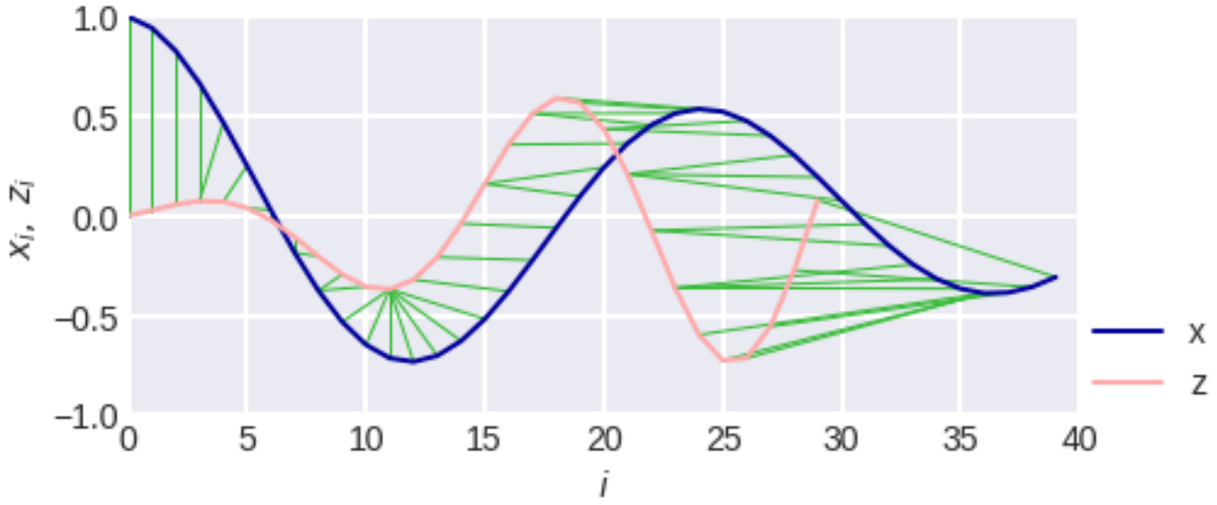
здесь нет нормировки… хотя может быть

Метрики на временных рядах

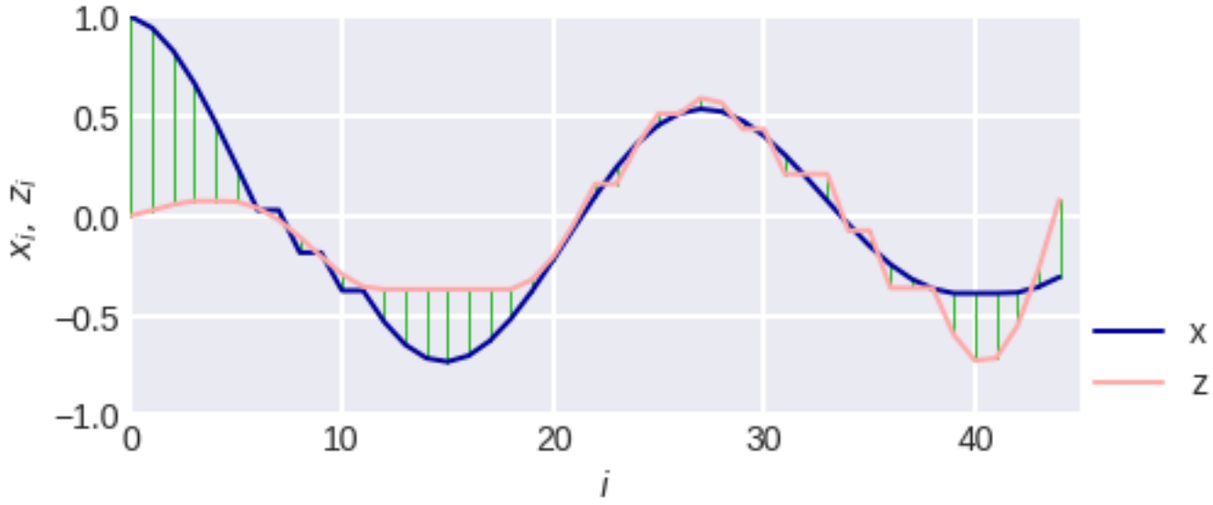
Для адекватности и быстроты часто

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





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)

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

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