# Метод k ближайших соседей

**Теоретический базис:**

Пусть на множестве объектов $X$ задана функция расстояния $\rho: X \times X \to [0, \infty)$. Существует целевая зависимость $y^*: X \to Y$, значения которой известны только на объектах обучающей выборки $X^\ell = (x_i, y_i)_{i=1}^\ell, y_i = y^*(x_i)$. Множество классов $Y$ конечно. Требуется построить алгоритм классификации $a: X \to Y$, аппроксимирующий целевую зависимость $y^*(x)$ на всём множестве $X$.

Для произвольного объекта $u \in X$ расположим элементы обучающей выборки $x_1, \ldots, x_\ell$ в порядке возрастания расстояний до $u$: $\rho(u, x_u^{(1)}) \leq \rho(u, x_u^{(2)}) \leq \cdots \leq \rho(u, x_u^{(\ell)})$, где через $x_u^i$ обозначается i-й сосед объекта u. Соответственно, ответ на i-м соседе объекта u есть $y_u^i = y^*(x_u^i)$. Таким образом, любой объект $u \in X$ порождает свою перенумерацию выборки.

***Определение:*** *Метрический алгоритм классификации с обучающей выборкой $X^\ell$ относит объект $u$ к тому классу $y \in Y$, для которого суммарный вес ближайших обучающих объектов $\Gamma_y(u, X^\ell)$ максимален:*

$$a(u; X^\ell) = \arg\max_{y \in Y} \Gamma_y(u, X^\ell); \; \Gamma_y(u, X^\ell) = \sum_{i=1}^{\ell} [y_u^i = y] w(i, u);$$

*де весовая функция $w(i, u)$ оценивает степень важности -го соседа для классификации объекта $u$. Функция $\Gamma_y(u, X^\ell)$ называется оценкой близости объекта $u$ к классу $y$.*

***Алгоритм k ближайших соседей*** (k nearest neighbors, kNN).

Чтобы сгладить влияние выбросов, будем относить объект $u$ к тому классу, элементов которого окажется больше среди $k$ ближайших соседей $x_u^i, i = 1, \ldots, k$:

$$w(i, u) = [i \leq k]; \; a(u; X^\ell, k) = \arg\max_{y \in Y} \sum_{i=1}^{k} [y_u^i = y].$$

При $k = 1$ этот алгоритм совпадает с предыдущим, следовательно, неустойчив к шуму. При $k = \ell$, наоборот, он чрезмерно устойчив и вырождается в константу.

Таким образом, крайние значения $k$ нежелательны. На практике оптимальное значение параметра $k$ определяют по критерию скользящего контроля с исключением объектов по одному (leave-one-out, LOO). Для каждого объекта $x_i \in X^\ell$ проверяется, правильно ли он классифицируется по своим $k$ ближайшим соседям.

$$LOO(k, X^\ell) = \sum_{i=1}^{\ell} [a(x_i ; X^\ell \setminus \{x_i\}, k) \neq y_i] \to \min_k \quad .$$

Заметим, что если классифицируемый объект $x_i$ не исключать из обучающей выборки, то ближайшим соседом $x_i$ всегда будет сам $x_i$ , и минимальное (нулевое) значение функционала $LOO(k)$ будет достигаться при $k = 1$. Существует и альтернативный вариант метода kNN: в каждом классе выбирается $k$ ближайших к $u$ объектов, и объект $u$ относится к тому классу, для которого среднее расстояние до $k$ ближайших соседей минимально.

**Алгоритм k взвешенных ближайших соседей.**

Недостаток kNN в том, что максимум может достигаться сразу на нескольких классах. В задачах с двумя классами этого можно избежать, если взять нечётное $k$. Более общая тактика, которая годится и для случая многих классов — ввести строго убывающую последовательность вещественных весов $w_i$ , задающих вклад i-го соседа в классификацию:

$$w(i,u) = [i \leq k] w_i \; ; \; a(u; X^\ell, k) = arg \max_{y \in Y} \sum_{i=1}^{k} \left[ y_u^{(i)} = y \right] w_i \quad .$$

Выбор последовательности $w_i$ является эвристикой. Если взять линейно убывающие веса $w_i = \frac{k+1-i}{k}$ , то неоднозначности также могут возникать, хотя и реже (пример: классов два; первый и четвёртый сосед голосуют за класс 1, второй и третий — за класс 2; суммы голосов совпадают). Неоднозначность устраняется окончательно, если взять нелинейно убывающую последовательность, скажем, геометрическую прогрессию: $w_i = q^i$, где знаменатель прогрессии $q \in (0, 1)$ является параметром алгоритма. Его можно подбирать по критерию $LOO$, аналогично числу соседей $k$

***Метод парзеновского окна***

Ещё один способ задать веса соседям — определить $w_i$ как функцию от расстояния $\rho(u, x_u^{(i)})$, а не от ранга соседа $i$. Введём функцию ядра $K(z)$, невозрастающую на $[0, \infty)$. Положив $w(i,u) = K(\frac{1}{h}\rho\left(u, x_u^{(i)}\right))$ в общей формуле, получим алгоритм

$$a(u;\, X^\ell, h) = arg\max_{y \in Y} \sum_{i=1}^{\ell} \left[y_u^{(i)} = y\right] K\left(\frac{\rho\left(u, x_u^{(i)}\right)}{h}\right)$$

Параметр $h$ называется шириной окна и играет примерно ту же роль, что и число соседей $k$. «Окно» — это сферическая окрестность объекта $u$ радиуса $h$, при попадании в которую обучающий объект $x_i$ «голосует» за отнесение объекта $u$ к классу $y_i$ . Мы пришли к этому алгоритму чисто эвристическим путём, однако он имеет более строгое обоснование в байесовской теории классификации, и, фактически, совпадает с методом парзеновского окна. Параметр $h$ можно задавать априори или определять по скользящему контролю. Зависимость $LOO(h)$, как правило, имеет характерный минимум, поскольку слишком узкие окна приводят к неустойчивой классификации; а слишком широкие — к вырождению алгоритма в константу.

Фиксация ширины окна $h$ не подходит для тех задач, в которых обучающие объекты существенно неравномерно распределены по пространству $X$. В окрестности одних объектов может оказываться очень много соседей, а в окрестности других — ни одного. В этих случаях применяется окно переменной ширины. Возьмём финитное ядро — невозрастающую функцию $K(z)$, положительную на отрезке $[0, 1]$, и равную нулю вне его. Определим $h$ как наибольшее число, при котором ровно $k$ ближайших соседей объекта $u$ получают ненулевые веса: $h(u) = \rho(u, x_u^{(k+1)})$. Тогда алгоритм принимает вид

$$a(u;\, X^\ell, k) = arg\max_{y \in Y} \sum_{i=1}^{k} [y_u^{(i)} = y]\, K\left(\frac{\rho\left(u, x_u^{(i)}\right)}{\rho\left(u, x_u^{(k+1)}\right)}\right) .$$

Заметим, что при финитном ядре классификация объекта сводится к поиску его соседей, тогда как при не финитном ядре (например, гауссовском) требуется перебор всей обучающей выборки.

Выбор ядра следует осуществлять из вариантов, представленных на рисунке:

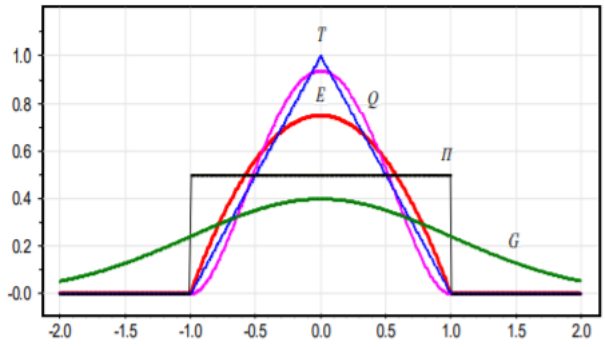


Рис. Часто используемые ядра:
$E$ — Епанечникова;
$Q$ — квартическое;
$T$ — треугольное;
$G$ — гауссовское;
$П$ — прямоугольное.

**Теория по языку Python:**

**Списки**

В языке Python доступно некоторое количество составных типов данных, использующихся для группировки прочих значений вместе. Наиболее популярные из них — список (list). Его можно выразить в тексте программы через разделённые запятыми значения (элементы), заключённые в квадратные скобки. Элементы списка могут быть разных типов.

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

Подобно индексам в строках, индексы списков начинаются с нуля, списки могут быть срезаны, объединены и так далее:

```
>>> a[0]
'spam'
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:-1]
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> 3*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']
```

В отличие от строк, являющихся неизменяемыми, изменить индивидуальные элементы списка вполне возможно:

```
>>> a
['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
```

```
>>> a
['spam', 'eggs', 123, 1234]
```

Присваивание срезу также возможно, и это действие может даже изменить размер списка или полностью его очистить:

```
>>> # Заменим некоторые элементы:
... a[0:2] = [1, 12]
>>> a
[1, 12, 123, 1234]
>>> # Удалим немного:
... a[0:2] = []
>>> a
[123, 1234]
>>> # Вставим пару:
... a[1:1] = ['bletch', 'xyzzy']
>>> a
[123, 'bletch', 'xyzzy', 1234]
>>> # Вставим (копию) самого себя в начало
>>> a[:0] = a
>>> a
[123, 'bletch', 'xyzzy', 1234, 123, 'bletch', 'xyzzy', 1234]
>>> # Очистка списка: замена всех значений пустым списком
>>> a[:] = []
>>> a
[]
```

Встроенная функция `len()` также применима к спискам:

```
>>> a = ['a', 'b', 'c', 'd']
>>> len(a)
4
```

Вы можете встраивать списки, например так:

```
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> len(p)
3
>>> p[1]
[2, 3]
>>> p[1][0]
```

```
2
```

Вы можете добавить что-нибудь в конец списка.

```
>>> p[1].append('xtra')
>>> p
[1, [2, 3, 'xtra'], 4]
```

**Оператор if**

```
if x < 0:
...      x = 0
...      print('Отрицательное значение, изменено на ноль')
... elif x == 0:
...      print('Ноль')
... elif x == 1:
...      print('Один')
... else:
...      print('Больше')
```

Блок else является не обязательным, а запись elif является сокращенной записью else if.

**Оператор for**

Оператор `for` в Python немного отличается от того, какой вы использовали в C/C++. Вместо предоставления пользователю возможности указать шаг итерации и условие остановки, оператор `for` в Python проходит по всем элементам любой последовательности (списка или строки) в том порядке, в котором они в ней располагаются. Например:

```
... a = ['cat', 'window', 'defenestrate']
>>> for x in a:
...     print(x, len(x))
```

Если вам нужно перебрать последовательность чисел, встроенная функция `range()` придёт на помощь. Она генерирует арифметические прогрессии:

```
>>> for i in range(5):
...    print(i)
```

Указанный конец интервала никогда не включается в сгенерированный список; вызов range(10) генерирует десять значений, которые являются подходящими индексами для элементов последовательности длины 10. Можно указать другое начало интервала и другую, даже отрицательную, величину шага.

```
range(5, 10)
  от 5 до 9

range(0, 10, 3)
  0, 3, 6, 9

range(-10, -100, -30)
  -10, -40, -70
```

**Определение функции**

```
>>> def fib(n):    # вывести числа Фибоначчи меньшие (вплоть до) n
...     """Выводит ряд Фибоначчи, ограниченный n."""
...     a, b = 0, 1
...     while b < n:
...         print(b, end=' ')
...         a, b = b, a+b
...
>>> # Теперь вызовем определенную нами функцию:
... fib(2000)
```

Зарезервированное слово def предваряет *определение* функции. За ним должны следовать имя функции и заключённый в скобки список формальных параметров. Выражения, формирующие тело функции, начинаются со следующей строки и должны иметь отступ.

Наиболее полезной формой является задание значений по умолчанию для одного или более параметров. Таким образом создаётся функция, которая может быть вызвана с меньшим количеством параметров, чем в её определении: при этом неуказанные при вызове параметры примут данные в определении функции значения. Например:

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
   while True:
      ok = input(prompt)
      if ok in ('y', 'yeah', 'yes', 'yep'): return True
      if ok in ('n', 'no', 'nop', 'nope'): return False
      retries = retries - 1
      if retries < 0:
```

```
        raise IOError('refusenik user')
    print(complaint)
```

Эта функция может быть вызвана, например, так: `ask_ok('Do you really want to quit?')` или так: `ask_ok('OK to overwrite the file?', 2)`.

Этот пример также знакомит вас с зарезервированным словом `in`. Посредством его можно проверить, содержит ли последовательность определённое значение или нет.

**Входные данные:**

К заданию на лабораторную работу прилагаются файлы, в которых представлены наборы данных из $\sim 10^4$ объектов. Каждый объект описывается двумя признаками ($f_j(x) \in R$) и соответствующим ему классом ($y \in \{0,1\}$).

**Задание:**

Суть лабораторной работы заключается в написании классификатора на основе метода k ближайших соседей. Данные из файла необходимо разбить на две выборки, обучающую и тестовую, согласно общепринятым правилам разбиения. На основе этих данных необходимо обучить разработанный классификатор и протестировать его на обеих выборках. В качестве отчёта требуется представить работающую программу и таблицу с результатами тестирования для каждого из 10 разбиений. Разбиение выборки необходимо выполнять программно, случайным образом, при этом, не нарушая информативности обучающей выборки. Разбивать рекомендуется по следующему правилу: делим выборку на 3 равных части, 2 части используем в качестве обучающей, одну в качестве тестовой. Кроме того, обучающая выборка должна быть сгенерирована таким образом, чтобы минимизировать разницу между количеством представленных в ней объектов разных классов, т.е. $abs\left(\left|\{(x_i, y_i) \in X^l \middle| y_i = -1\}\right| - \left|\{(x_i, y_i) \in X^l | y_i = 1\}\right|\right) \to min$.

**Варианты:**

Выполнение лабораторной работы разбито на несколько пунктов, в каждом из которых есть несколько вариантов, выбор варианта опирается на $N_c$ – индивидуальный номер студента (для студентов очной формы обучения это номер в журнале).

Первый пункт отвечает за выбор типа классификатора. Вариант выбирается по формуле $N_{\text{В}} = (N_c \ mod\ 3) + 1$:

1. Метод k взвешенных ближайших соседей
2. Метод парзеновского окна с фиксированным h
3. Метод парзеновского окна с относительным размером окна

Для первого варианта задания необходимо использовать весовую функцию $w_i$ по формуле $N_w = (N_c mod\ 2) + 1$. Параметр $q$ подбирается методом скользящего контроля.

1. $w_i = q^i, q \in (0,1)$
2. $w_i = \left(\frac{k+1-i}{k}\right)^q, q \in \{2,3,4\}$

В случае 2-го и 3-го вариантов, необходимо использовать функцию ядра $K(z)$ из списка по следующей формуле $N_{\text{я}} = \left((N_{\text{с}} * 6 + 13)\ mod\ 8\ mod\ 3\right) + 1$:

1. *Q –квартическое* $K(x) = (1 - r^2)^2[r \leq 1]$
2. *T – треугольное* $K(x) = (1 - r)[r \leq 1]$
3. *П – прямоугольное* $K(x) = [r \leq 1]$

Кроме того, к лабораторной работе прилагаются 5 файлов с данными для классификации, файл выбирается по следующей формуле

$$N_{\text{ф}} = \left((N_c + 2) mod\ 5\right) + 1$$