




# k-NN и приближённый поиск ближайших соседей

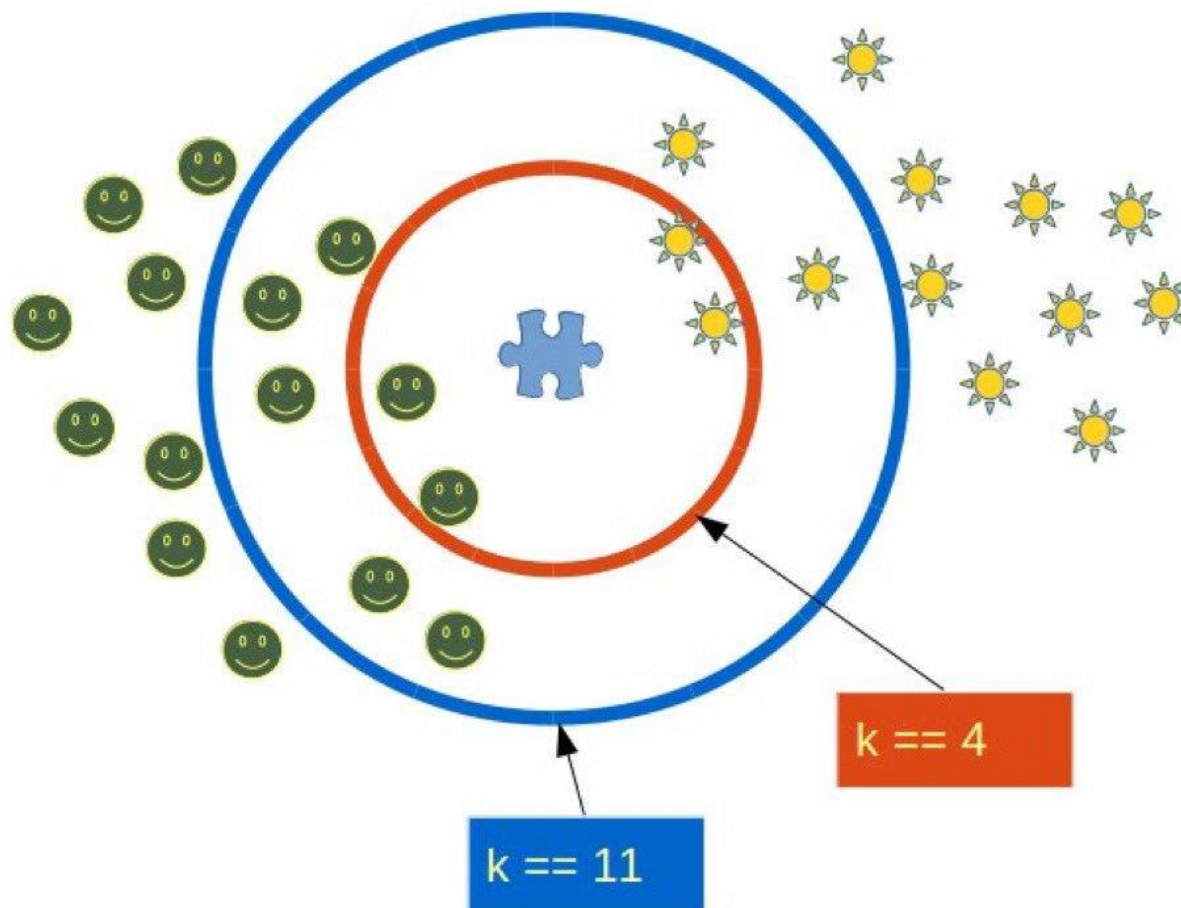
Анищенко Илья

НИУ ВШЭ

27 сентября 2019

О чем будет идти речь?

 ==  or  ==  ?



## Гипотезы компактности и непрерывности

Задачи классификации и регрессии:

$X$  — объекты,  $Y$  — ответы;

$X^\ell = (x_i, y_i)_{i=1}^\ell$  — обучающая выборка;

Гипотеза непрерывности (для регрессии):

*“Близким” объектам соответствуют близкие ответы*

выполнена:



не выполнена:



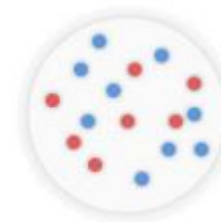
Гипотеза компактности (для классификации):

*“Близкие” объекты гораздо чаще лежат в одном классе, чем в разных*

выполнена:



не выполнена:



## Гипотезы компактности и непрерывности

Понятие “близости” формально:

Для объектов с численными признаками –  
Евклидова метрика

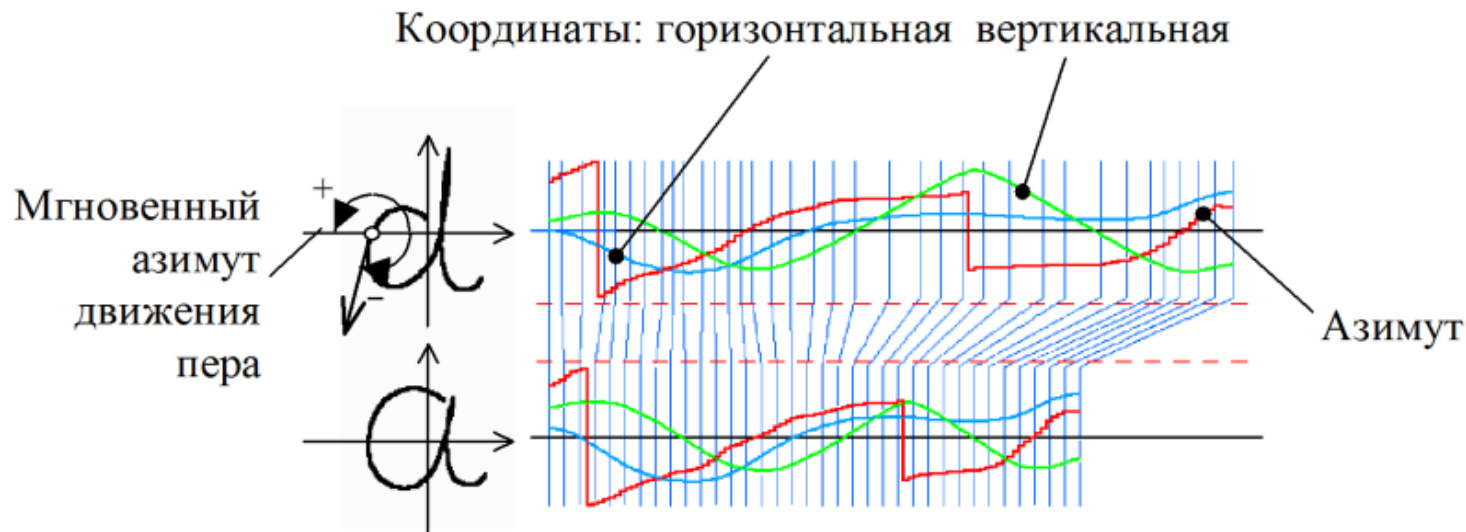
$$\rho(x, x_i) = \left( \sum_{j=1}^n |x^j - x_i^j|^2 \right)^{1/2}$$

$x = (x^1, \dots, x^n)$  — вектор признаков объекта  $x$ ,  
 $x_i = (x_i^1, \dots, x_i^n)$  — вектор признаков объекта  $x_i$ ,

Для объектов типа “строка” — расстояние Левенштейна

СТGGGCTAAAGGTCCTTAGCC..TTTAGAAAAA.GGGCCATTAGGAAATTGC  
СТGGGACTAAA....CCTTAGCCTATTTACAAAAATGGGCCATTAGG...TTGC

Для сигналов — энергия сжатий и растяжений



## Метрический классификатор в общем виде

Для произвольного  $x \in X$  отсортируем объекты из выборки  $x_1, \dots, x_\ell$  по дальности от него

$$\rho(x, x^{(1)}) \leq \rho(x, x^{(2)}) \leq \dots \leq \rho(x, x^{(\ell)}),$$

$x^{(i)}$  —  $i$ -й сосед объекта  $x$  среди  $x_1, \dots, x_\ell$ ;

$y^{(i)}$  — ответ на  $i$ -м соседе объекта  $x$ .

Метрический алгоритм классификации:

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

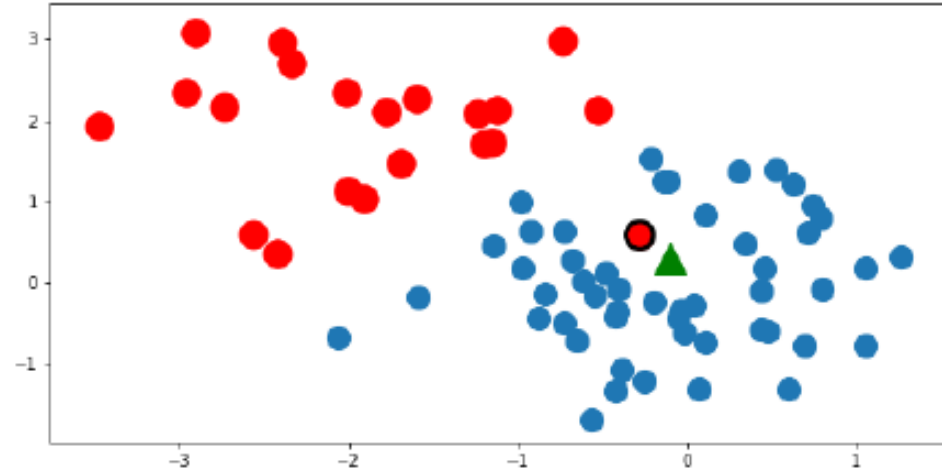
оценка близости объекта  $x$  к классу  $y$

Где  $w(i, x)$  — вес  $i$ -го соседа объекта  $x$ . За вес берем неотрицательную и невозрастающую функцию по  $i$

# Метод ближайшего соседа

Определение класса объекта идет только на основе 1 ближайшего соседа

$$w(i, x) = [i \leq 1]$$



## Плюсы:

- Интерпретация ответа
- Простота реализации (все обучение – запомнить обучающую выборку)

## Недостатки:

- Неустойчивость к выбросам в выборке (шуму)
- Нет настраиваемых параметров
- Хранение всей выборки целиком

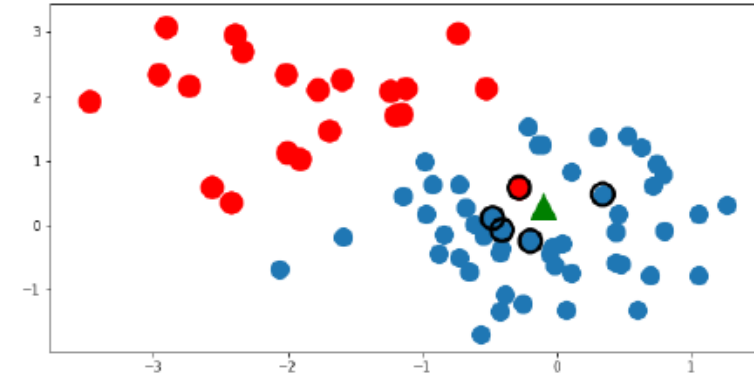
# Метод k-ближайших соседей

Определение класса объекта идет на основе информации от k-ближайших соседей

Ответ – преобладающий класс из k-ближайших соседей

$$w(i, x) = [i \leq k]$$

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



## Плюсы:

- Гиперпараметр k хорошо поддается оптимизации (leave-one-out)
- Такой подход лучше устойчив к шуму, выбросам

## Минусы:

- Неоднозначность результата в некоторых случаях
- Также хранение всей выборки целиком

## Важно отметить:

идея брать  $k = n$  (вся выборка) тоже не очень хороша. Так как результат будет вырождаться в какой-то доминирующий класс на выборке

## Метод k взвешенных ближайших соседей

$$w(i, x) = [i \leq k] w_i,$$

где  $w_i$  — вес, зависящий только от номера соседа;

### Некоторые варианты эвристик:

$w_i = \frac{k+1-i}{k}$  — линейные убывающие веса;

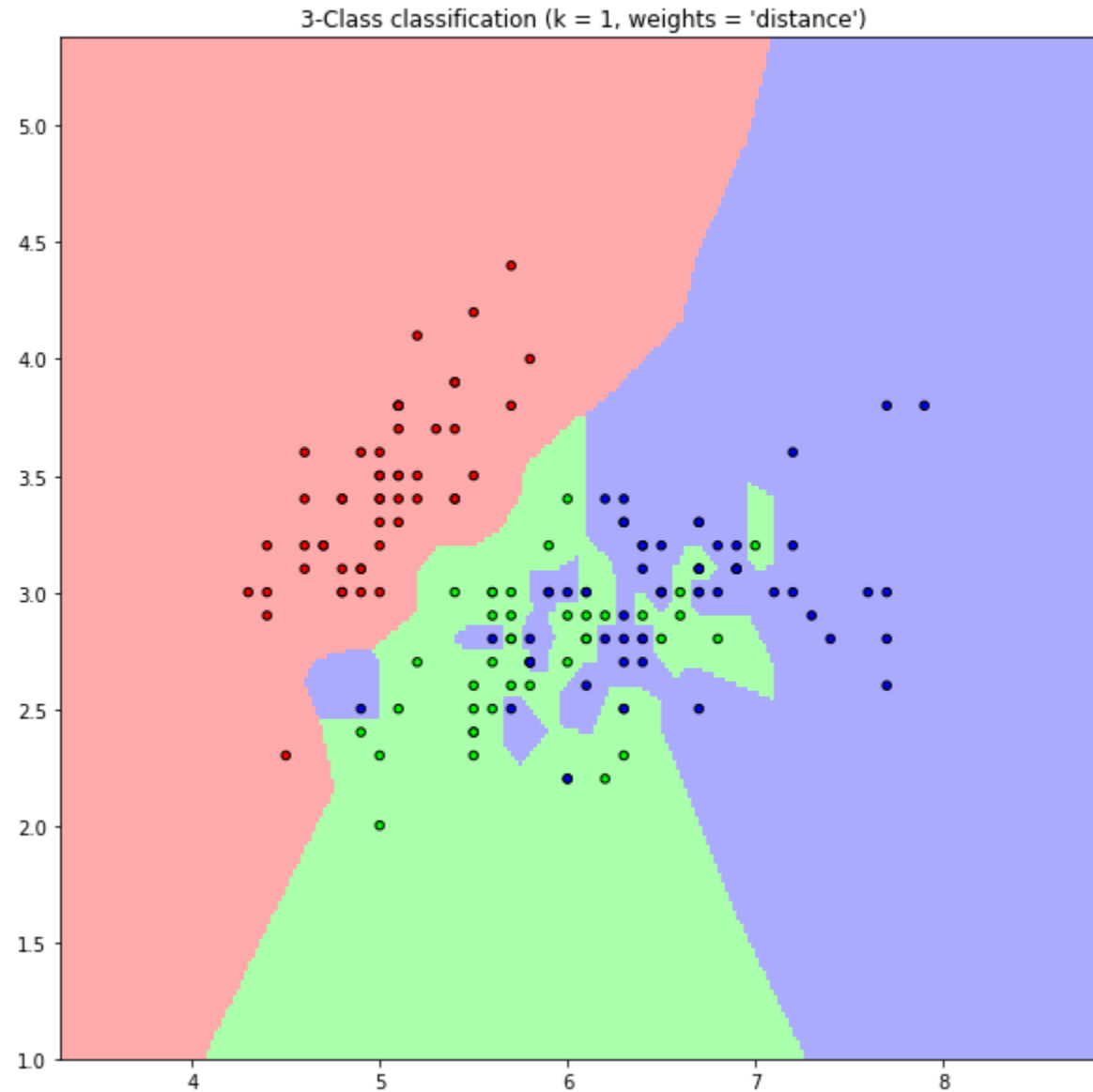
$w_i = q^i$  — экспоненциально убывающие веса,  $0 < q < 1$ ;

### Проблемы такого подхода:

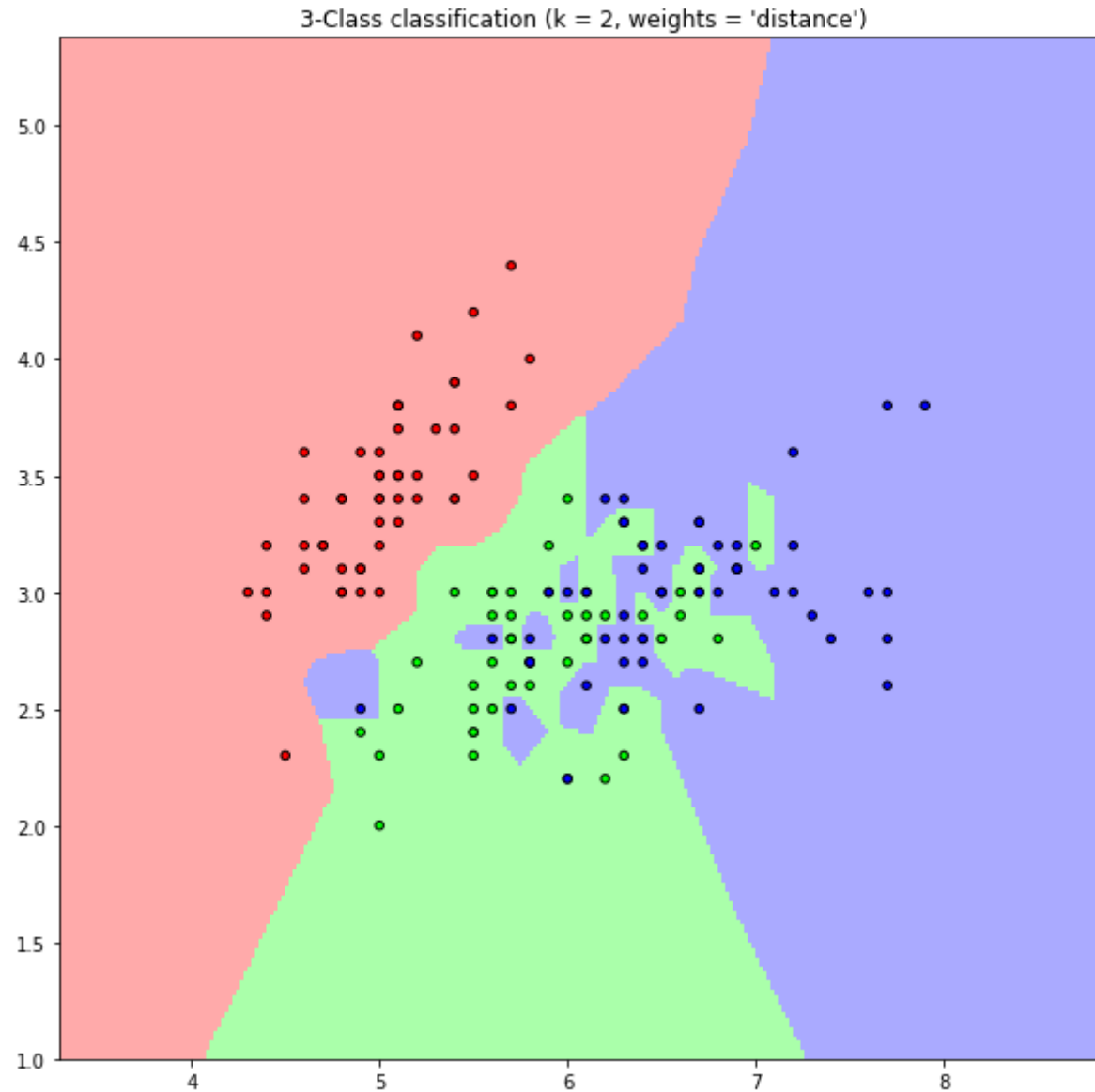
- По каким критериям брать функции для весов?
- Корректна ли зависимость веса от порядкового номера объекта?



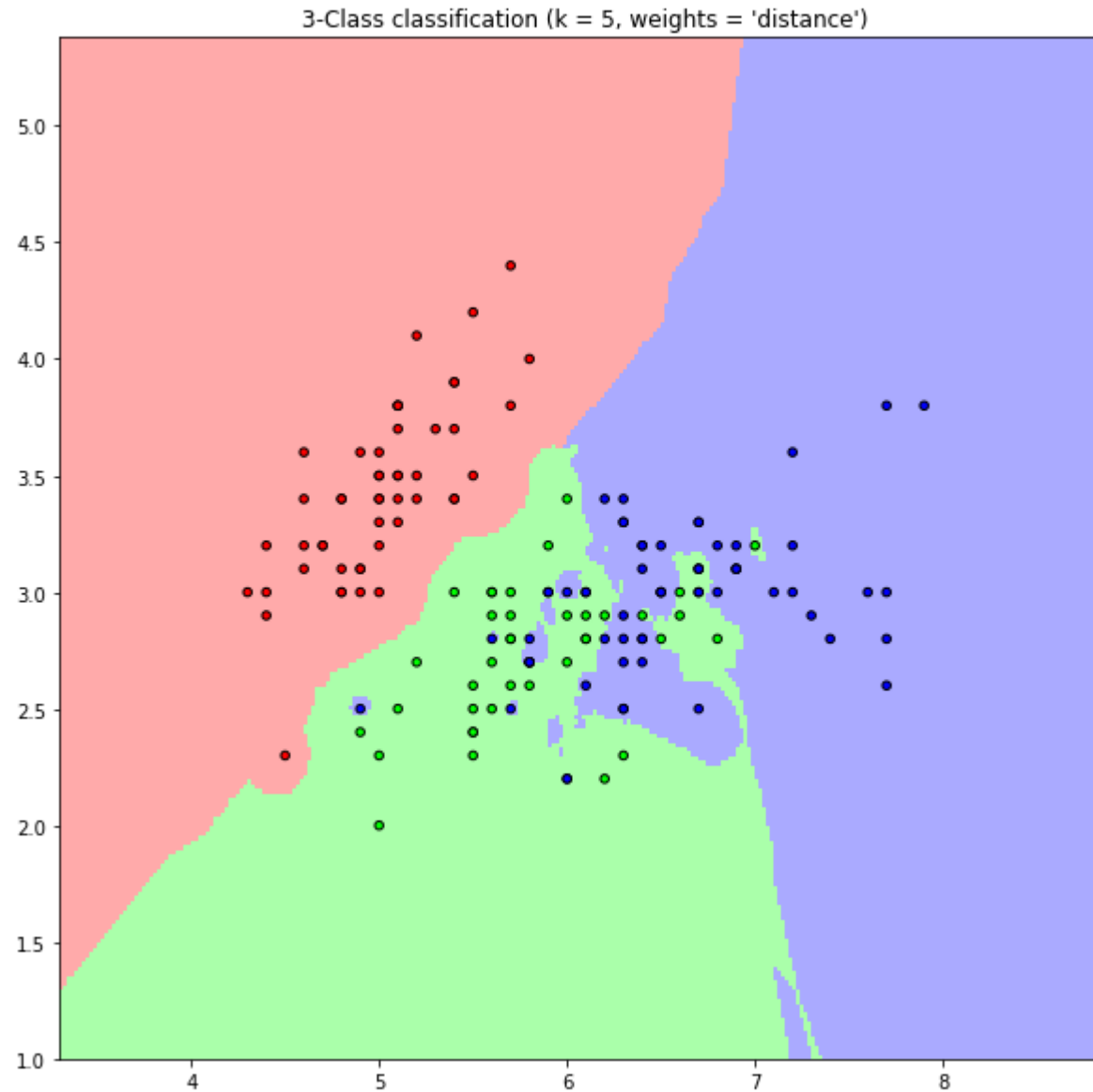
## Влияние гиперпараметра $k$ на классификацию



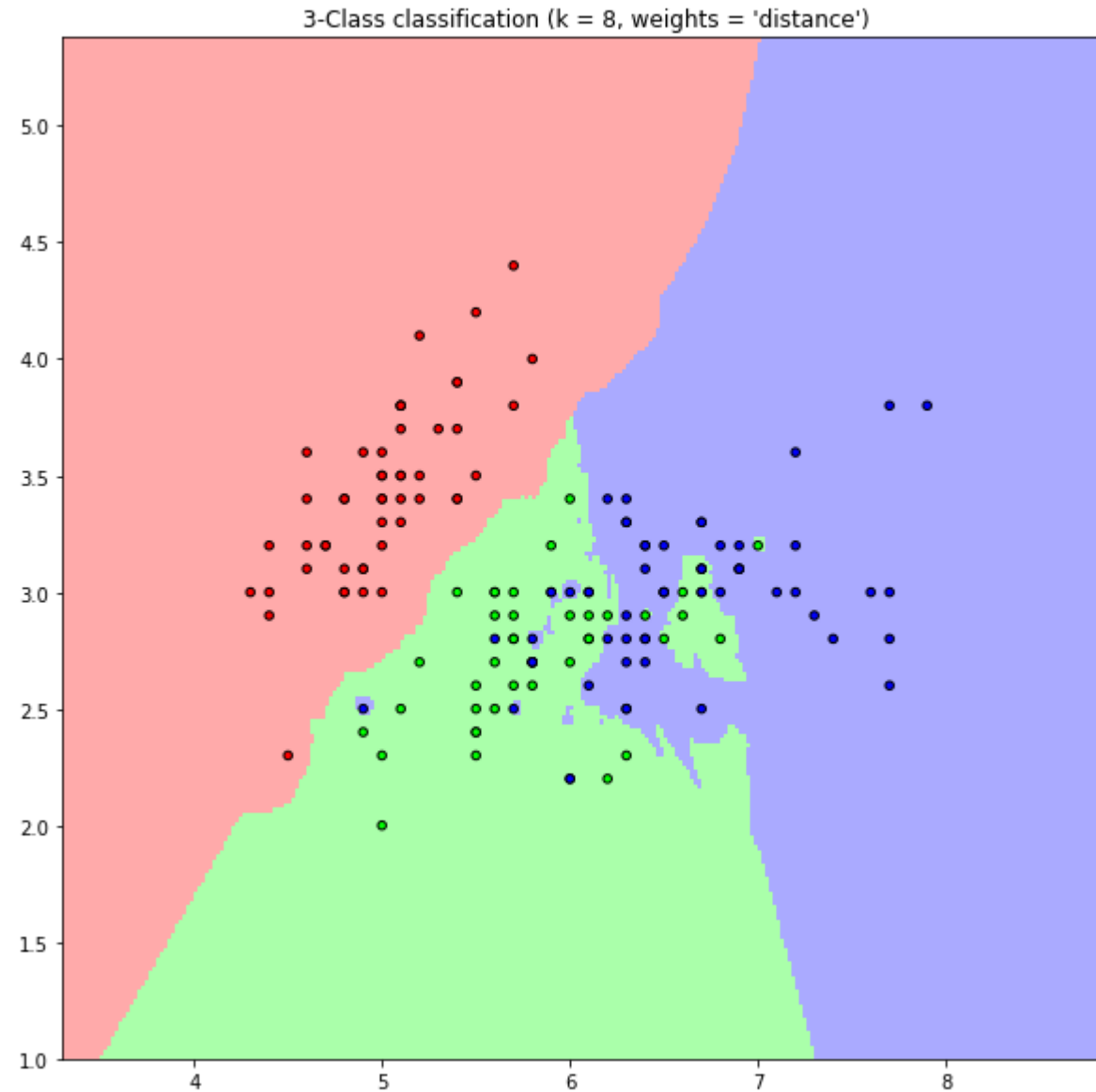
## Влияние гиперпараметра k на классификацию



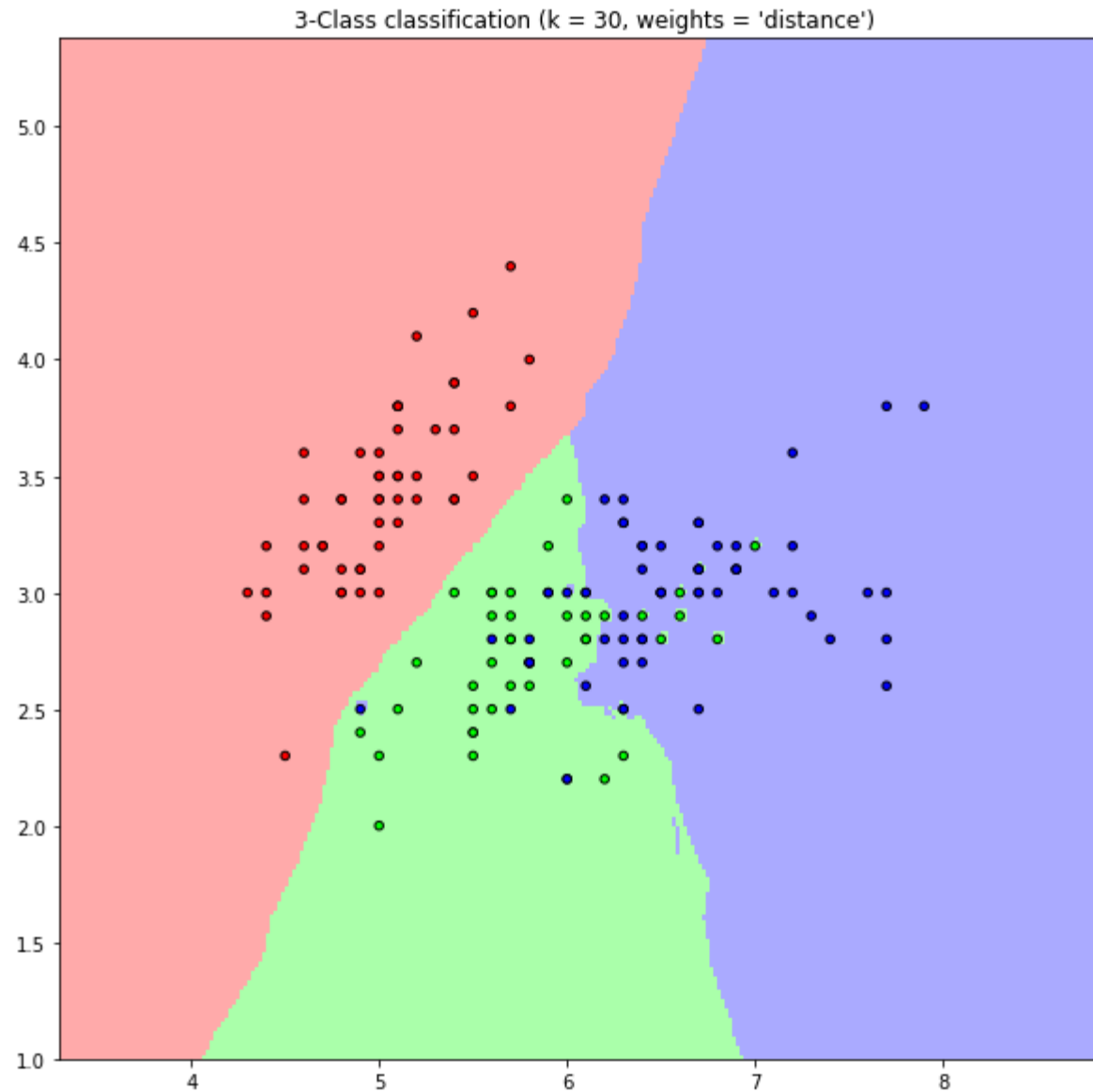
## Влияние гиперпараметра k на классификацию



## Влияние гиперпараметра k на классификацию



## Влияние гиперпараметра k на классификацию



# Применение kNN

## Классификация:

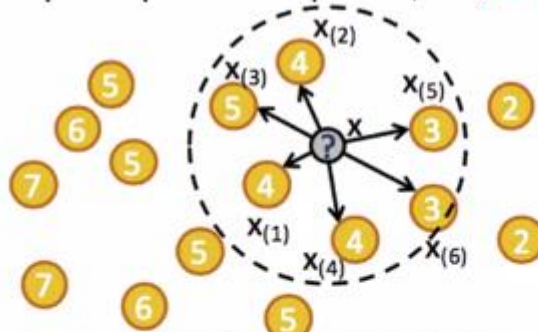
Поиск по картинке



## Регрессия:

Получение информации об объекте от его 6 ближайших соседей

Пример классификации ( $k = 6$ )



$$\begin{aligned} \textcircled{?} = & \frac{4w(x_{(1)}) + 4w(x_{(2)}) + 5w(x_{(3)}) +}{w(x_{(1)}) + w(x_{(2)}) + w(x_{(3)}) +} \\ & + \frac{4w(x_{(4)}) + 3w(x_{(5)}) + 3w(x_{(6)})}{w(x_{(4)}) + w(x_{(5)}) + w(x_{(6)})} \end{aligned}$$

## С какими еще проблемами сталкивается kNN?

Наличие в выборке шумов/выбросов/неинформативных объектов:

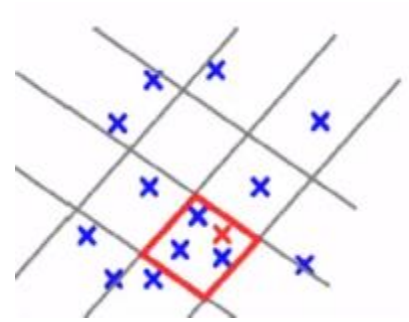
- Решается поиском эталонных объектов, устранением шумов/выбросов

Проблема больших размерностей (большого кол-ва признаков):

- Решается через понижение размерности пр-ва пространства (проецирование данных)
- Или отбор главных признаков из множества всех, и дальнейшая работа только с ними

# Метод приближенного поиска ближайших соседей

Основная идея – работа с индексирующей структурой  
(разбиение пространства поиска)



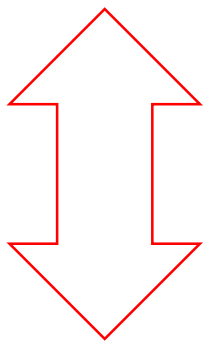
Данная структура работы актуальна для баз с миллиардами данных



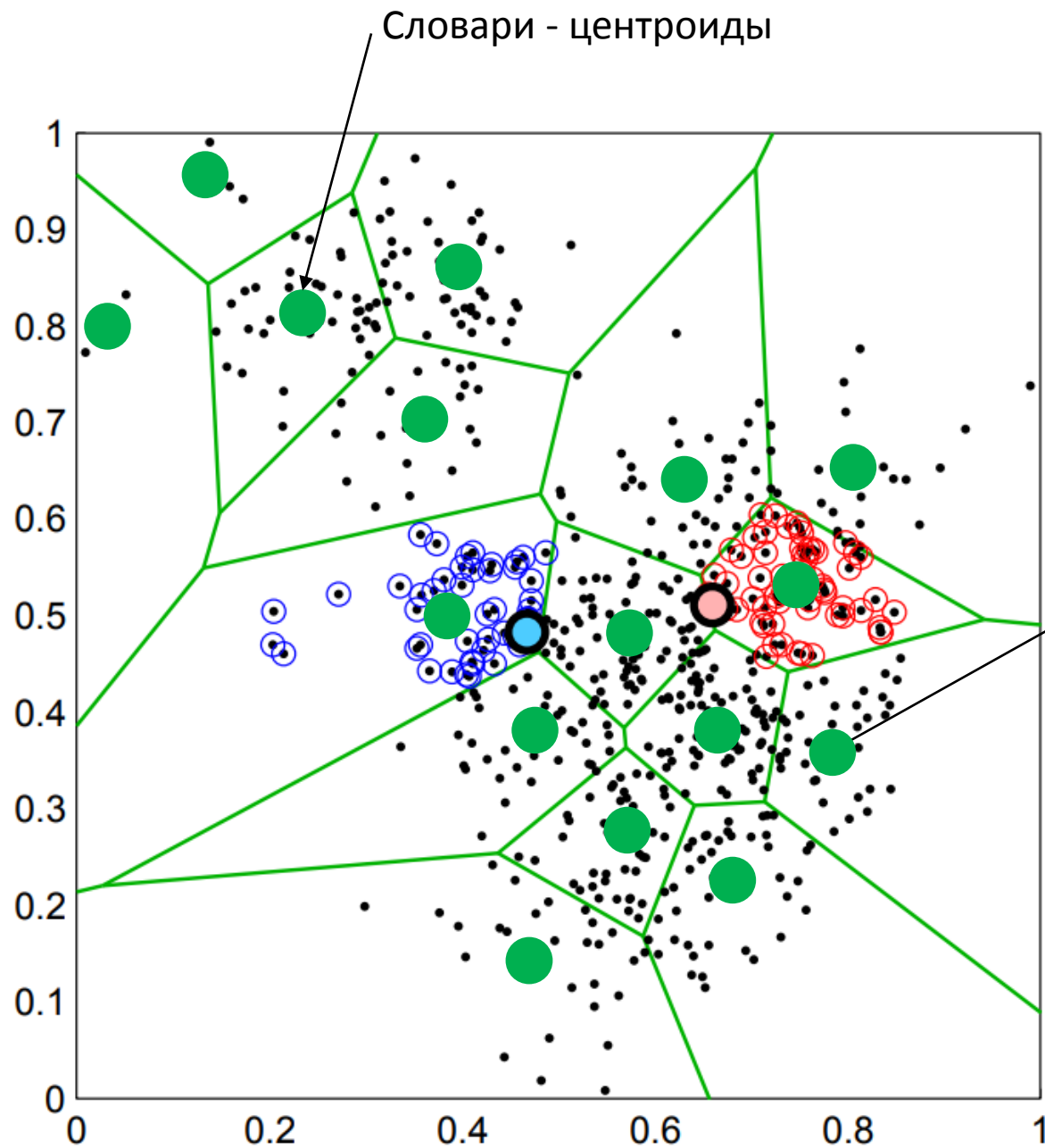
## Inverted index

Необходимо проверять  
несколько ближайших  
регионов

Чем больше регионов,  
тем лучше



Нахождение  
перспективных регионов  
должно быть быстрым



# Inverted multi-index

## Идея:

Заменяем K-means кластеризацию исходных d-мерных векторов на две отдельные кластеризации первых и вторых половин: получаем  $K^2$  регионов

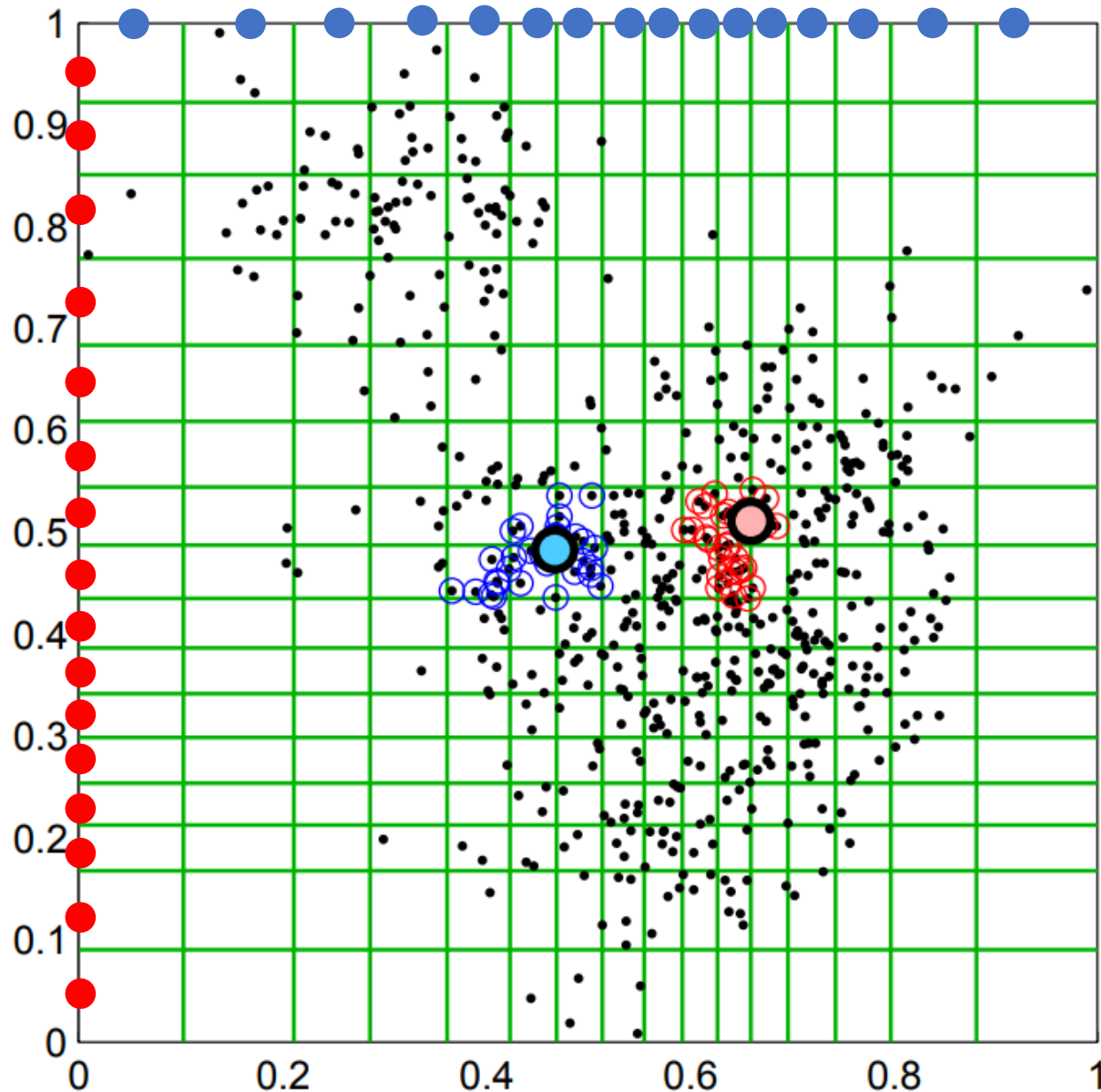
Получаем больше регионов фактически для того же K

## Проблема такого подхода:

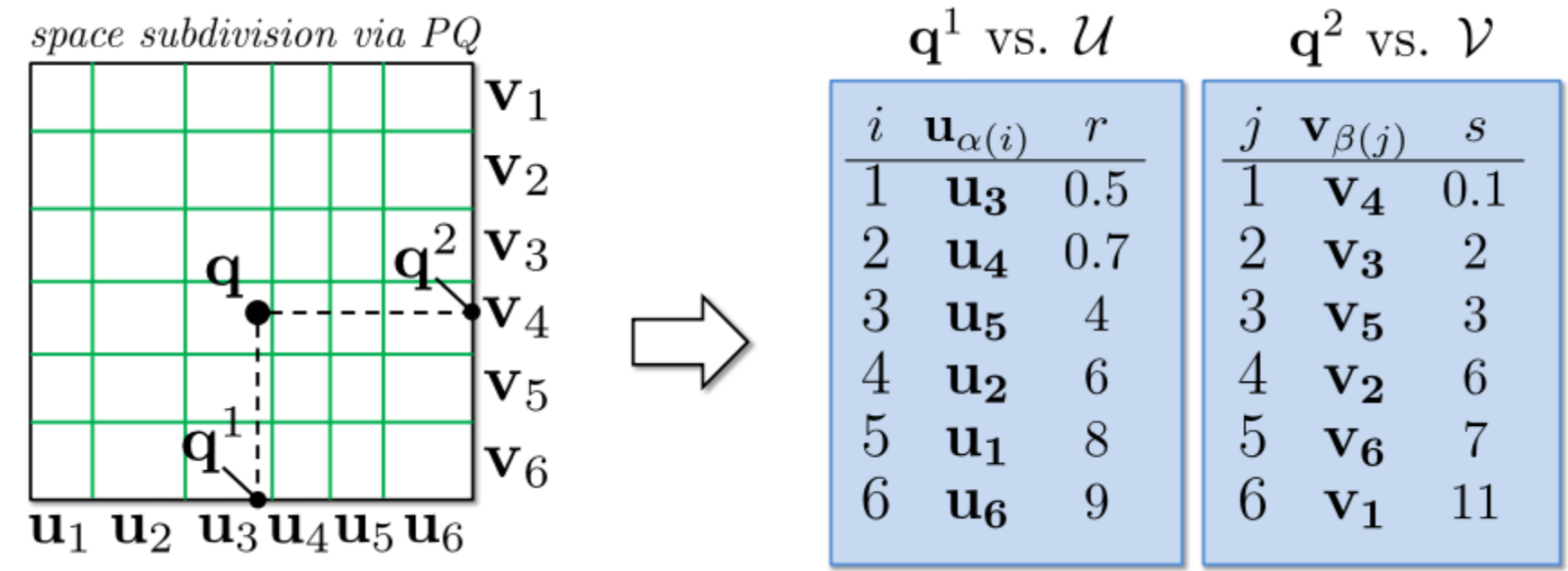
Некоторые регионы могут оказаться пустыми

## Вопрос:

Каким образом выстроить приоритетность обхода регионов?



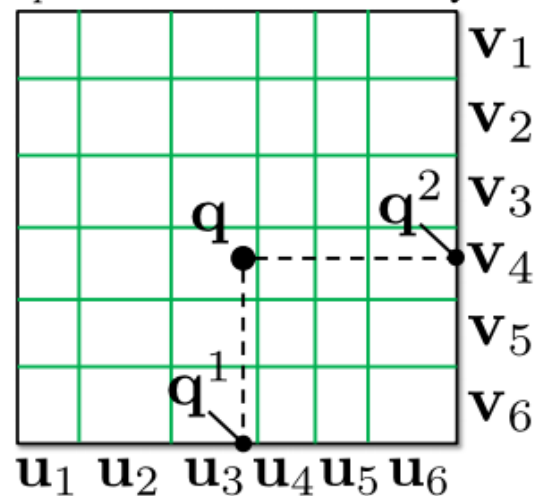
Проблема приоритетности обхода регионов



	Индекс	Мульти-индекс
Число регионов	$K$	$K^2$
Нахождение расстояний до элементов словарей	$O(K)$	$O(K)$

# Проблема приоритетности обхода регионов

space subdivision via PQ



$q^1$  vs.  $\mathcal{U}$

$i$	$u_{\alpha(i)}$	$r$
1	$u_3$	0.5
2	$u_4$	0.7
3	$u_5$	4
4	$u_2$	6
5	$u_1$	8
6	$u_6$	9

$q^2$  vs.  $\mathcal{V}$

$j$	$v_{\beta(j)}$	$s$
1	$v_4$	0.1
2	$v_3$	2
3	$v_5$	3
4	$v_2$	6
5	$v_6$	7
6	$v_1$	11



multi-sequence algorithm

$[u_{\alpha(i)} \ v_{\beta(j)}]$	$(i, j)$	$r(i) + s(j)$
$[u_3 \ v_4]$	(1,1)	0.6 (0.5+0.1)
$[u_4 \ v_4]$	(2,1)	0.8 (0.7+0.1)
$[u_3 \ v_3]$	(1,2)	2.5 (0.5+2)
$[u_4 \ v_3]$	(2,2)	2.7 (0.7+2)
$[u_3 \ v_5]$	(1,3)	3.5 (0.5+3)
$[u_4 \ v_5]$	(2,3)	3.7 (0.7+3)
$[u_5 \ v_4]$	(3,1)	4.1 (4+0.1)
$[u_5 \ v_3]$	(3,2)	6 (4+2)
$[u_3 \ v_2]$	(1,4)	6.5 (0.5+6)
...		

	1	2	3	4	5	6	
1	0.6	0.8	4.1	6.1	8.1	9.1	
2	2.5	2.7	6	8	10	11	
3	3.5	3.7	7	9	11	12	
4	6.5	6.7	10	12	14	15	
5	7.5	7.7	11	13	15	16	
6	11.5	11.7	15	17	19	20	
	$u_3$	$u_4$	$u_5$	$u_2$	$u_1$	$u_6$	

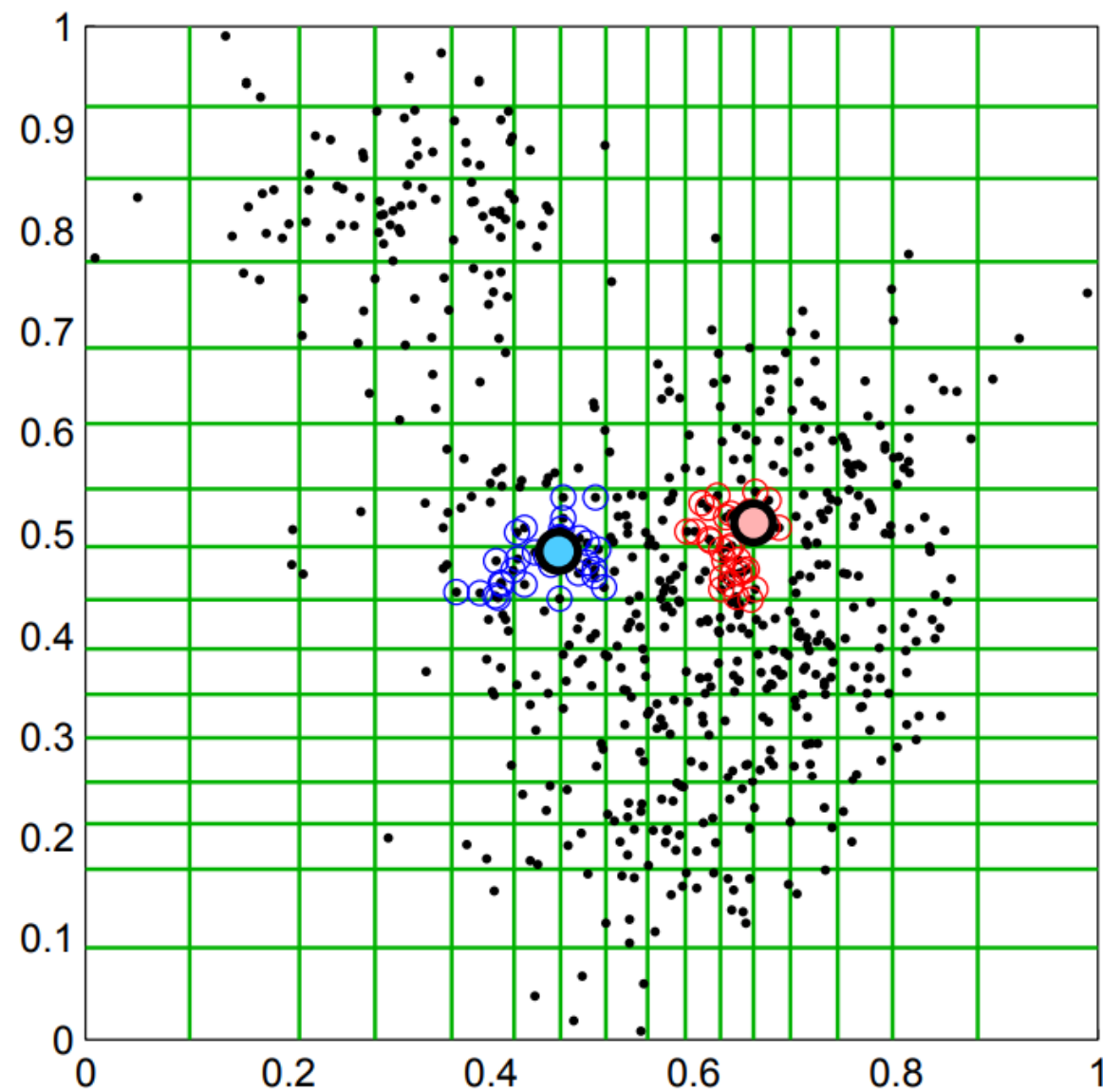
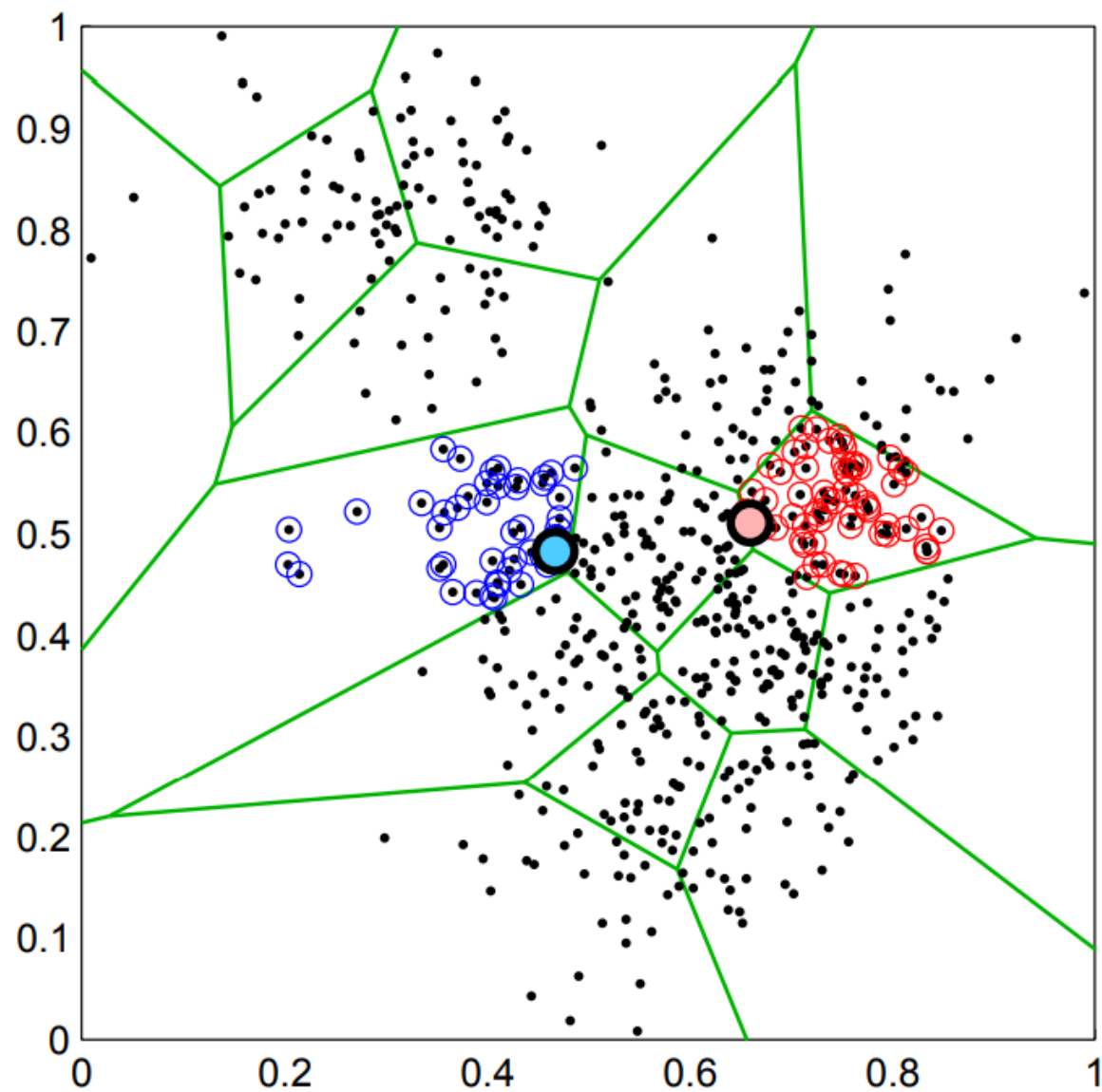
	1	2	3	4	5	6	
1	0.6	0.8	4.1	6.1	8.1	9.1	
2	2.5	2.7	6	8	10	11	
3	3.5	3.7	7	9	11	12	
4	6.5	6.7	10	12	14	15	
5	7.5	7.7	11	13	15	16	
6	11.5	11.7	15	17	19	20	
	$u_3$	$u_4$	$u_5$	$u_2$	$u_1$	$u_6$	

	1	2	3	4	5	6	
1	0.6	0.8	4.1	6.1	8.1	9.1	
2	2.5	2.7	6	8	10	11	
3	3.5	3.7	7	9	11	12	
4	6.5	6.7	10	12	14	15	
5	7.5	7.7	11	13	15	16	
6	11.5	11.7	15	17	19	20	
	$u_3$	$u_4$	$u_5$	$u_2$	$u_1$	$u_6$	

	1	2	3	4	5	6	
1	0.6	0.8	4.1	6.1	8.1	9.1	
2	2.5	2.7	6	8	10	11	
3	3.5	3.7	7	9	11	12	
4	6.5	6.7	10	12	14	15	
5	7.5	7.7	11	13	15	16	
6	11.5	11.7	15	17	19	20	
	$u_3$	$u_4$	$u_5$	$u_2$	$u_1$	$u_6$	

	1	2	3	4	5	6	
1	0.6	0.8	4.1	6.1	8.1	9.1	$v_4$
2	2.5	2.7	6	8	10	11	$v_3$
3	3.5	3.7	7	9	11	12	$v_5$
4	6.5	6.7	10	12	14	15	$v_2$
5	7.5	7.7	11	13	15	16	$v_6$
6	11.5	11.7	15	17	19	20	$v_1$
	$u_3$	$u_4$	$u_5$	$u_2$	$u_1$	$u_6$	

## Наглядное сравнение



## Сложность нахождения очередного региона

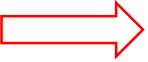
После нахождения  $m$  ближайших регионов очередь с приоритетами содержит не более чем  $\lceil 0.5 + \sqrt{2m + 0.25} \rceil$  элементов

После нахождения  $m$  ближайших регионов сложность нахождения очередного региона составит  $O(\log m)$

Также дополнительной памяти потребуется:  $\text{sizeof}(\text{int}) * K^2$  bytes. Так как в большом массиве данных регион объектов будет задаваться только точкой старта конкретного. Эти точки старта и хранятся отдельно.

## Почему для многомерных векторов идёт разбиение на две части?

При большем разбиении частей возможны следующие проблемы:

- Потребуется больше дополнительной памяти  $\text{sizeof}(\text{int}) * K^2$  bytes   $\text{sizeof}(\text{int}) * K^4$  bytes
- Рост числа пустых регионов (будет уходить больше времени на накопление нужного числа кандидатов)

## Данные, с которыми велись эксперименты в статье

Миллиард 128-мерных векторов

Отдельное множество из 10.000 запросов, для которых известны истинные ближайшие соседи

Производилось сравнение индекса и мульти-индекса:

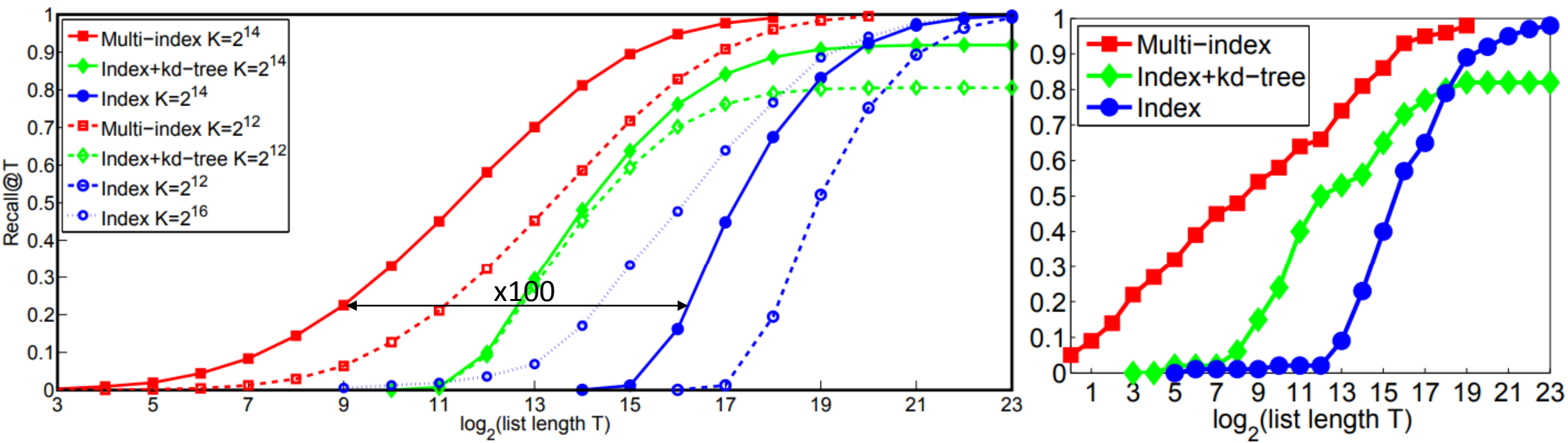
- Устанавливали кол-во кандидатов  $T$
- Для каждого запроса:
  - Формировали  $T$  кандидатов
  - Проверяли, содержится ли истинный ближайший сосед в списке кандидатов

Для обоих методов сравнивались доли запросов, для которых истинный ближайший сосед содержался в списке из  $T$  кандидатов



# Что вышло по результатам

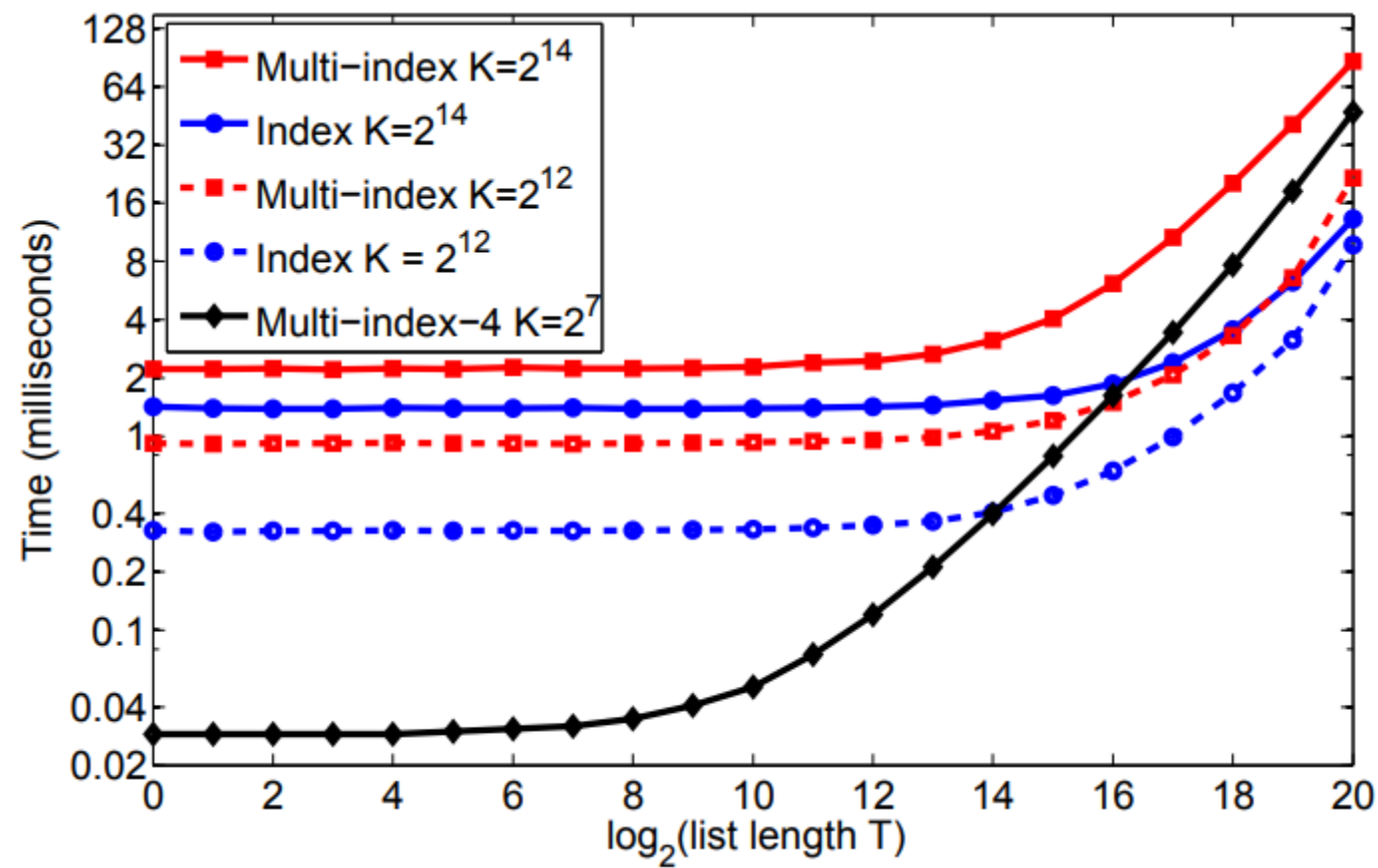
На сколько быстро метод находит ближайших соседей?





Что вышло по результатам

Время нахождения кандидатов



## Мульти-индекс с переранжированием

Схема “Multi-ADC” – кодировать исходные векторы некоторым кол-вом бит

Схема “Multi-D-ADC” – кодировать смещения исходных векторов относительно центров регионов, которым они принадлежат

Как получились итоговые оценки:

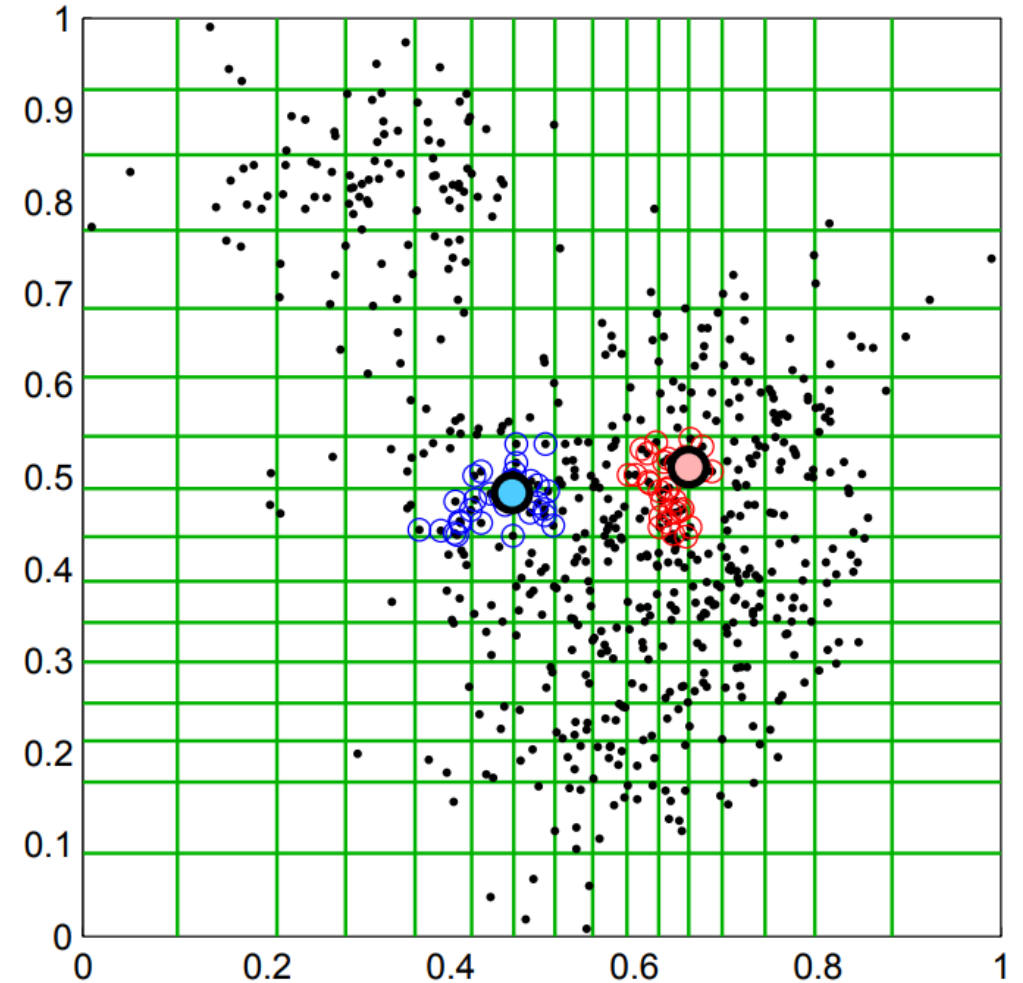
- Для запроса формировали список из  $T$  кандидатов
- Реконструировали закодированных кандидатов и переранжировали их согласно расстоянию до запроса
- Проверяли, есть ли истинный ближайший сосед в топе размера  $T^*$

## Как-то так

System	Number of cells	List len. $T$	R@1	R@10	R@100	Time(ms)	Memory(Gb)
<b>BIGANN, 1 billion SIFTs, 8 bytes per vector</b>							
IVFADC [13]	$2^{13}$	8 million	0.112 <sub>(0.088)</sub>	0.343 <sub>(0.372)</sub>	0.728 <sub>(0.733)</sub>	155 <sub>(74)</sub>	12
IVFADC [13]	$2^{16}$	600000	0.124	0.414	0.772	25	12
Multi-D-ADC	$2^{14} \times 2^{14}$	10000	0.153	0.473	0.707	2	13
Multi-D-ADC	$2^{14} \times 2^{14}$	30000	0.161	0.506	0.813	4	13
Multi-D-ADC	$2^{14} \times 2^{14}$	100000	0.162	0.515	0.854	11	13
<b>BIGANN, 1 billion SIFTs, 16 bytes per vector</b>							
IVFADC+R [13]	$2^{13}$	8 million	(0.262)	(0.701)	(0.962)	(116*)	20
IVFADC [13]	$2^{16}$	600000	0.311	0.750	0.923	28	20
Multi-D-ADC	$2^{14} \times 2^{14}$	10000	0.303	0.672	0.742	2	21
Multi-D-ADC	$2^{14} \times 2^{14}$	30000	0.325	0.762	0.883	5	21
Multi-D-ADC	$2^{14} \times 2^{14}$	100000	0.332	0.799	0.959	16	21

## Заключение по методу

- Хорошая структура данных для индексации векторов высокой размерности
- Существенное улучшение в точности и скорости поиска в больших массивах данных
- Метод внедрен в поисковые системы



## Общее заключение

- Метрические методы классификации (kNN в частности) реально применять для создания рекомендательных систем или для систем принятия решений
- Уже существует много вариаций этого метода, которые отличаются набором гиперпараметров и весовыми функциями.
- Сам поиск ближайших соседей можно ускорить, используя методы приближенного поиска, такие как inverted multi-index или KD-tree

## Вопросы?

- 1) К каким результатам может привести значение гиперпараметра  $k = 1$ ,  $m$ (всей выборке) в алгоритме поиска  $k$  ближайших соседей?
- 2)Оптимальный способ подбора гиперпараметра  $k$  для  $kNN$ .
- 3) в чем заключается метод inverted index? В чем его выигрыш перед  $kNN$ .
- 4) В чем выигрыш метода inverted multi index перед inverted index.

## Ссылки на источники

- <http://www.machinelearning.ru/wiki/> - материалы по kNN, гипотезам и методу метрической классификации
- <http://sites.skoltech.ru/app/data/uploads/sites/25/2014/12/TPAMI14.pdf> - статья по inverted multi-index
- [https://scikit-learn.org/stable/auto\\_examples/neighbors/plot\\_classification.html#sphx-glr-download-auto-examples-neighbors-plot-classification-py](https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-download-auto-examples-neighbors-plot-classification-py) – графическая реализация kNN