Методы приближенного поиска ближайших соседей

Антюх Михаил

Задача

- **Имеем:** множество элементов, расположенных в метрическом пространстве и функция близости (метрика), которая задает данное метрическое пространство.
- **Хотим:** найти элементы близкие к заданному.

Рассматриваемые методы

- K-мерное дерево (k-d tree)
- Инвертированный индекс (inverted index)

K-мерное дерево

- **K-мерное дерево (*k-d tree)*** - *это* структура данных, которая позволяет разбить K-мерное пространство на части, посредством сечения этого самого пространства гиперплоскостями (**K > 3**), плоскостями (**K = 3**), прямыми (**K = 2**) ну, и в случае одномерного пространства - точкой.
- **bounding box** - параллелепипед, который описывает исходное множество объектов (сам объект), стороны bounding box-а параллельны осям координат.

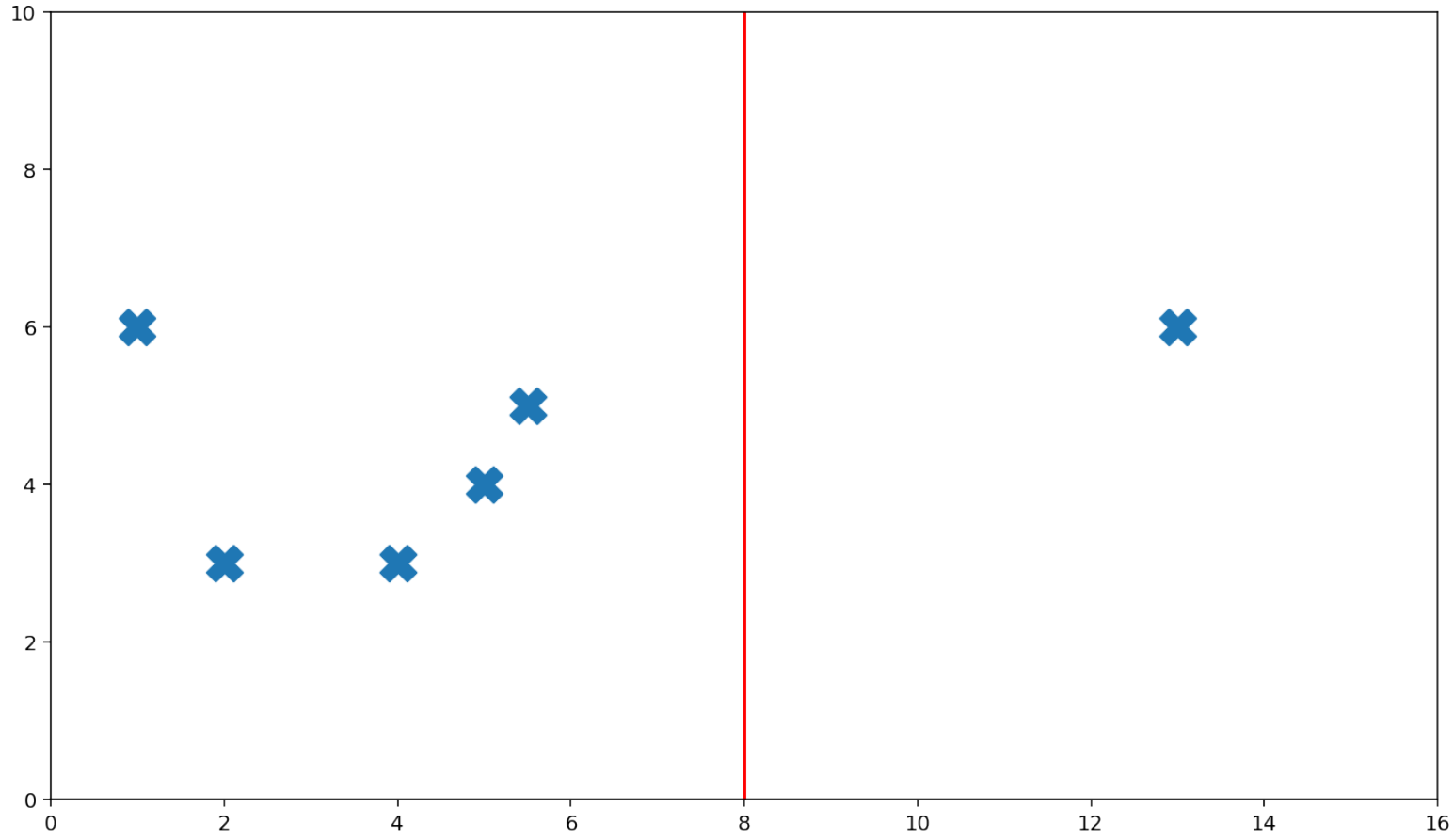
Построение к-мерного дерева

- Все элементы помещаются в один bounding box, описывающий элементы исходного множества.
- bounding box разбивается плоскостью на две части (добавляются два новых узла).
- Таким же путем каждая из полученных частей разбивается на две и т.д.
- Процесс завершается при достижение определенного условия (например при достижение определенного числа элементов в узле дерева), либо при достижении определенной глубины дерева

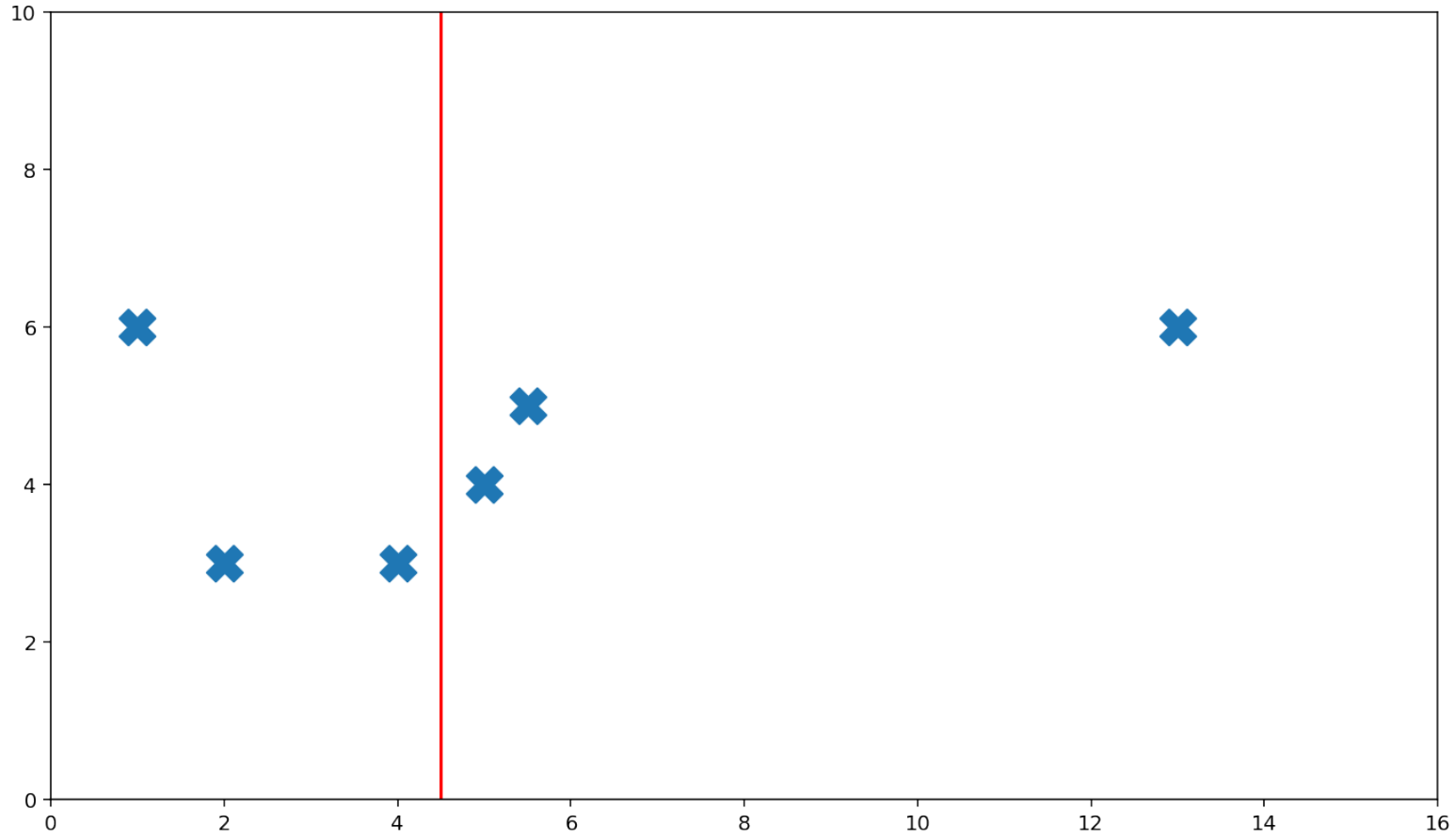
Способы разделения

- Провести секущую плоскость через середину наибольшей стороны bounding box-а.
- По медиане, то есть так чтобы в обеих частях было примерно одинаковое количество элементов.
- Использовать *SAH (Surface Area Heuristic)* функцию оценки.

Разделение по наибольшей стороне



Разделение по медиане



Разделение с помощью SAH

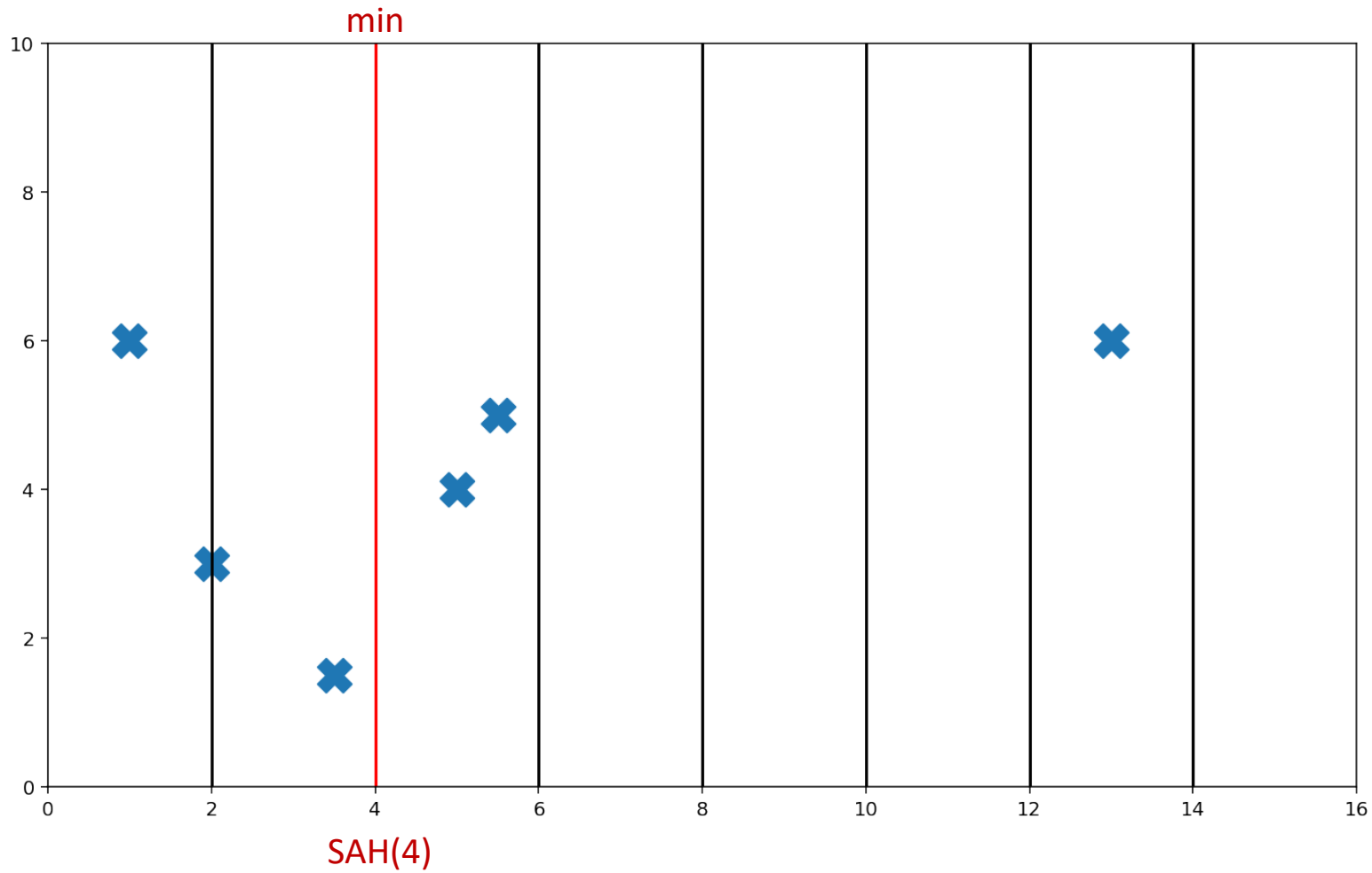
- Основной bounding box делится N плоскостями на N + 1 часть.
- Для плоскостей вычисляют значение функции SAH.
- Выбирается плоскость с минимальным значением SAH.

$$\text{SAH(x)} = C_t + C_i \times \frac{S_l(x) \times N_l(x) + S_r(x) \times N_r(x)}{S_{all}(x)}$$

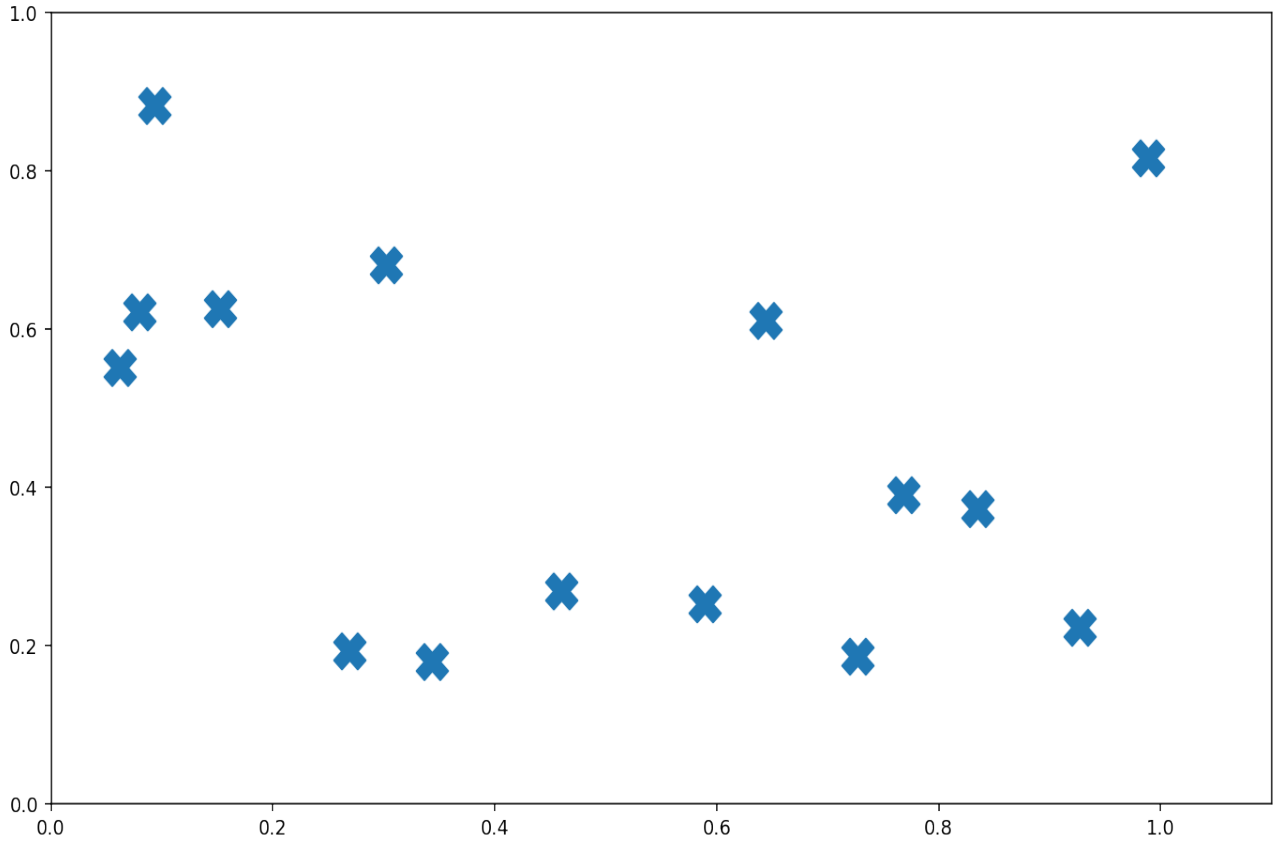
Где:

- C_t – стоимость построения плоскости
- C_i – стоимость пересечения *элемента плоскостью*
- S_l, S_r – площади поверхности слева и справа от плоскости
- N_l, N_r –количество элементов слева и справа от плоскости
- S_{all} – площадь всей поверхности

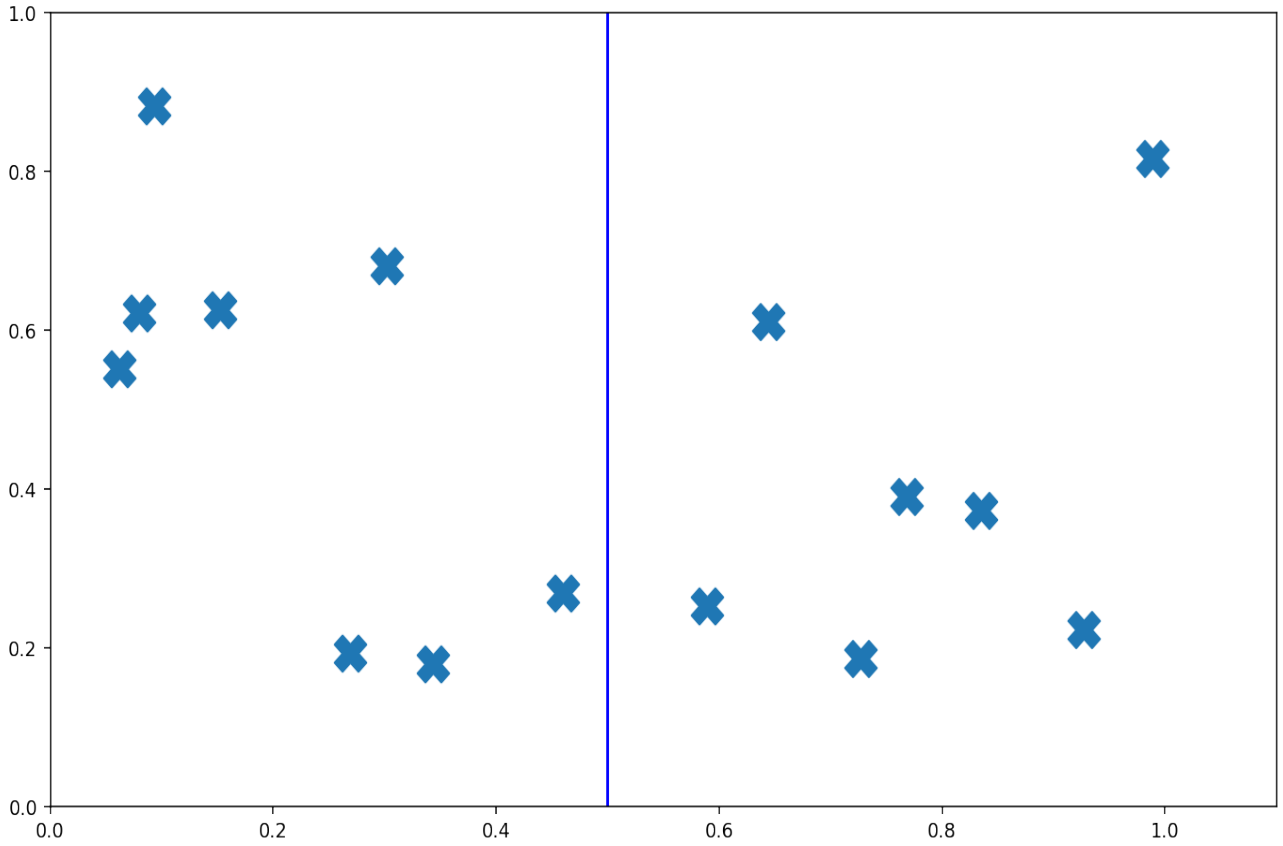
Разбиение с помощью SAH

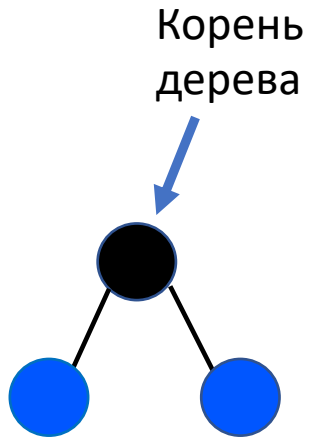


Пример построения

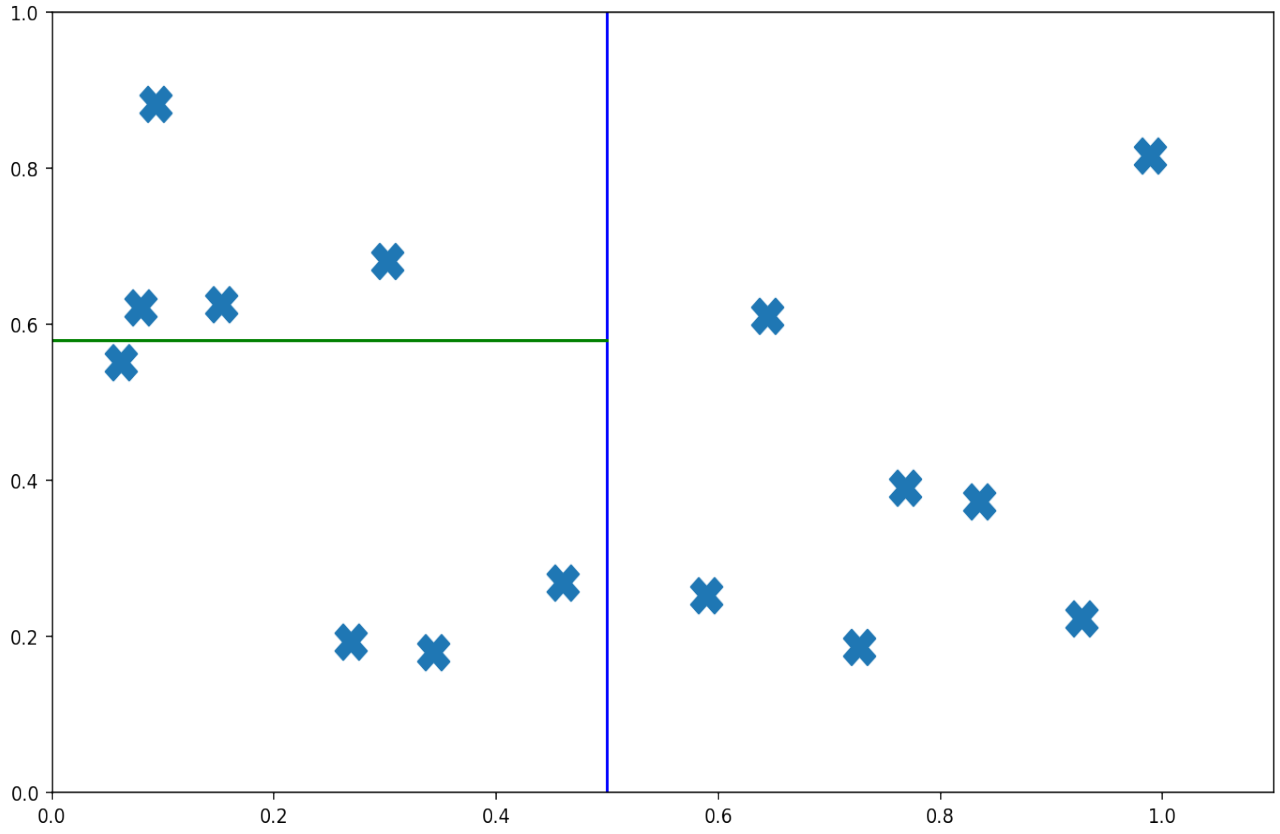


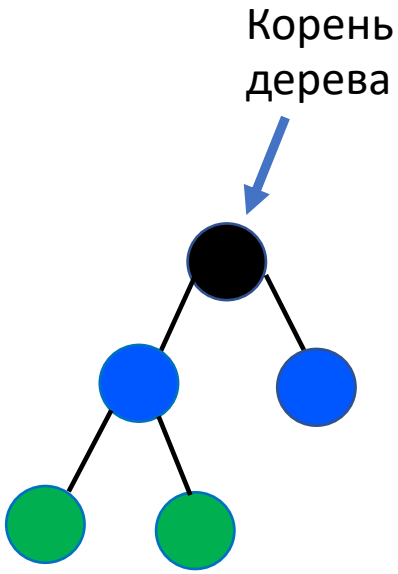
Пример построения



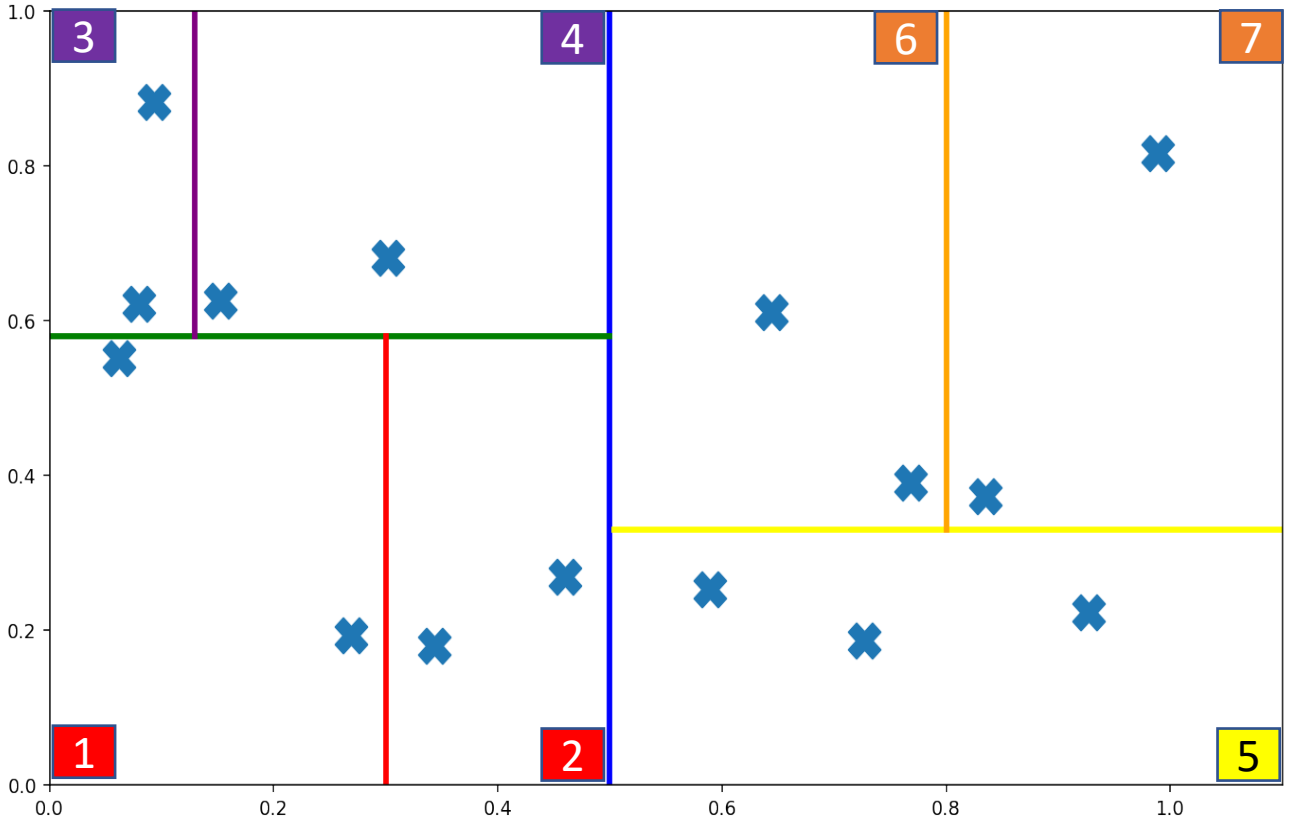


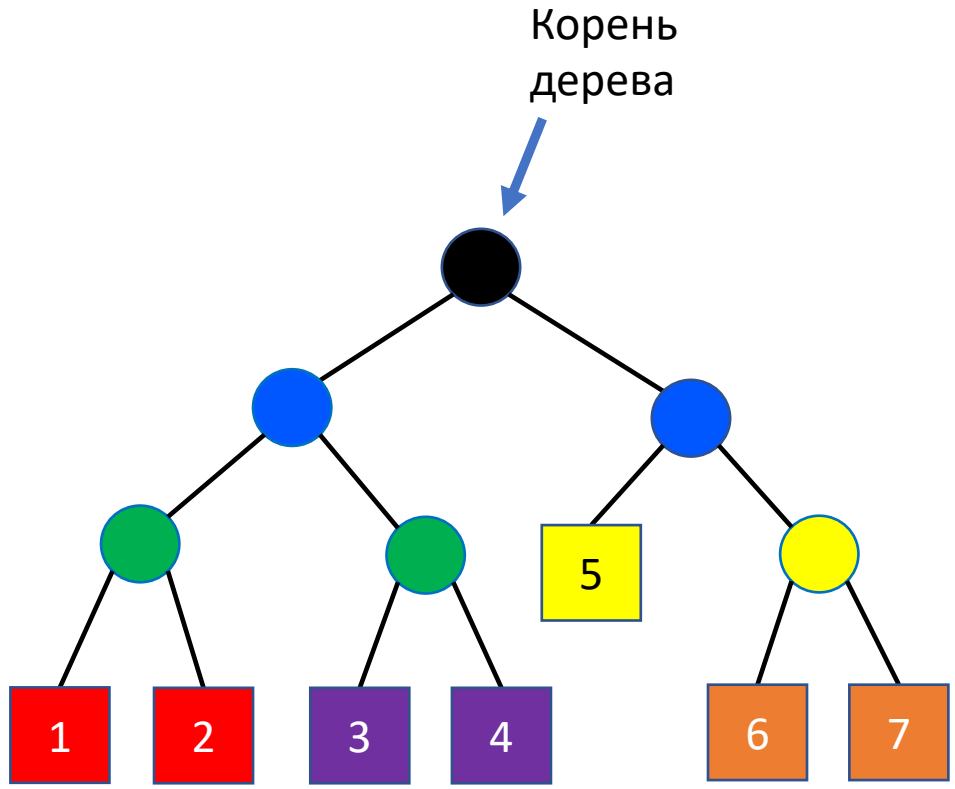
Пример построения





Пример построения



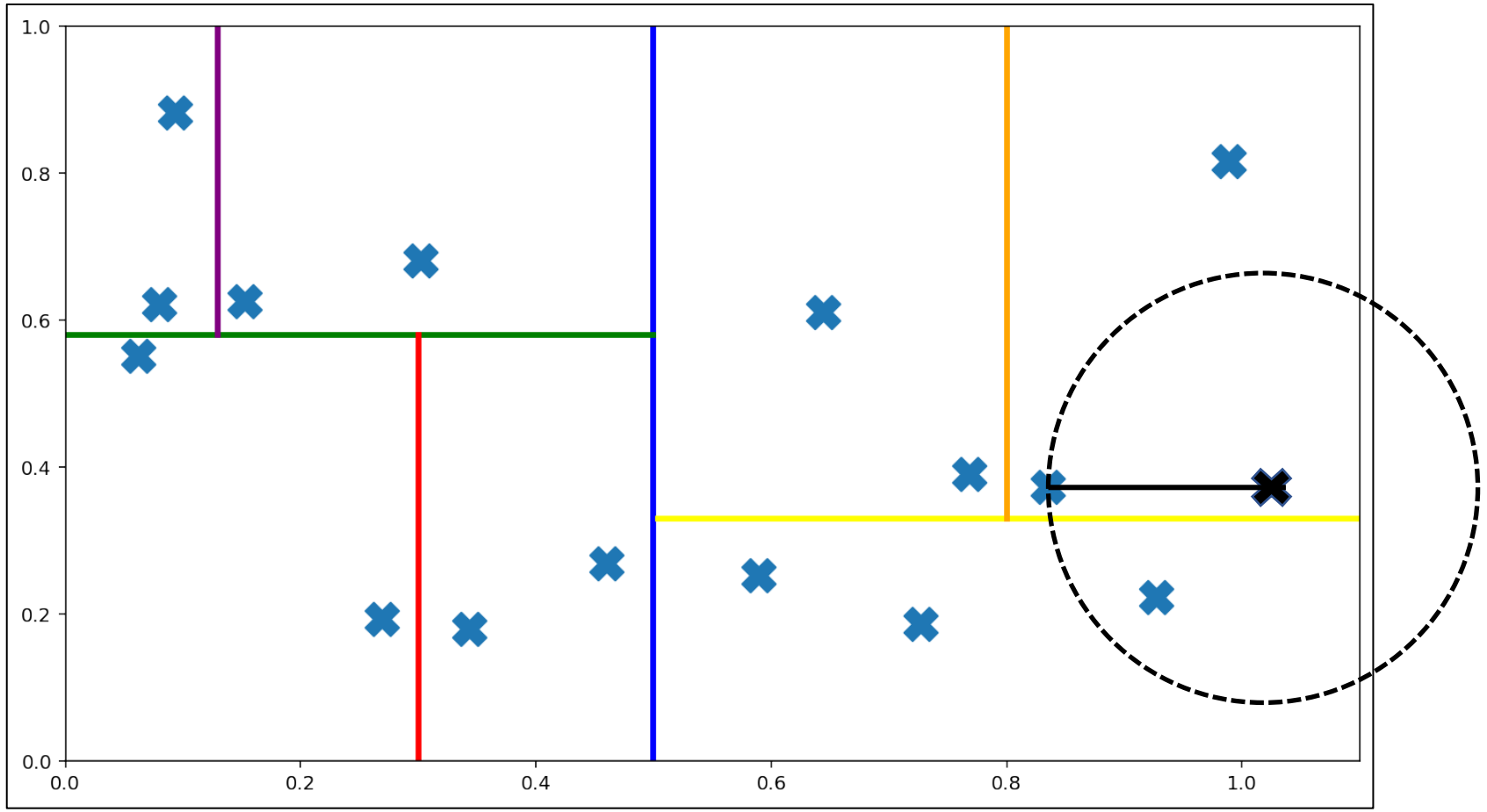


Поиск ближайшего соседа

- **Шаг 1.** Находим лист в котором лежит заданный элемент.
- **Шаг 2.** Полным перебором в данном листе ищем ближайший элемент к заданному и обозначаем расстояние за min_dist.
- **Шаг 3.** Строим вокруг заданного элемента сферу с радиусом длины min_dist.
- **Шаг 4.** Запускаем обход дерева с корня и проверяем пересекает ли сфера левый или правый узел, при достижение листа запускаем перебор элементов и если нашелся более близкий чем min_dist, то обновляем min_dist и радиус сферы.
- **Шаг 5.** Продолжаем обход с обновленным радиусом.

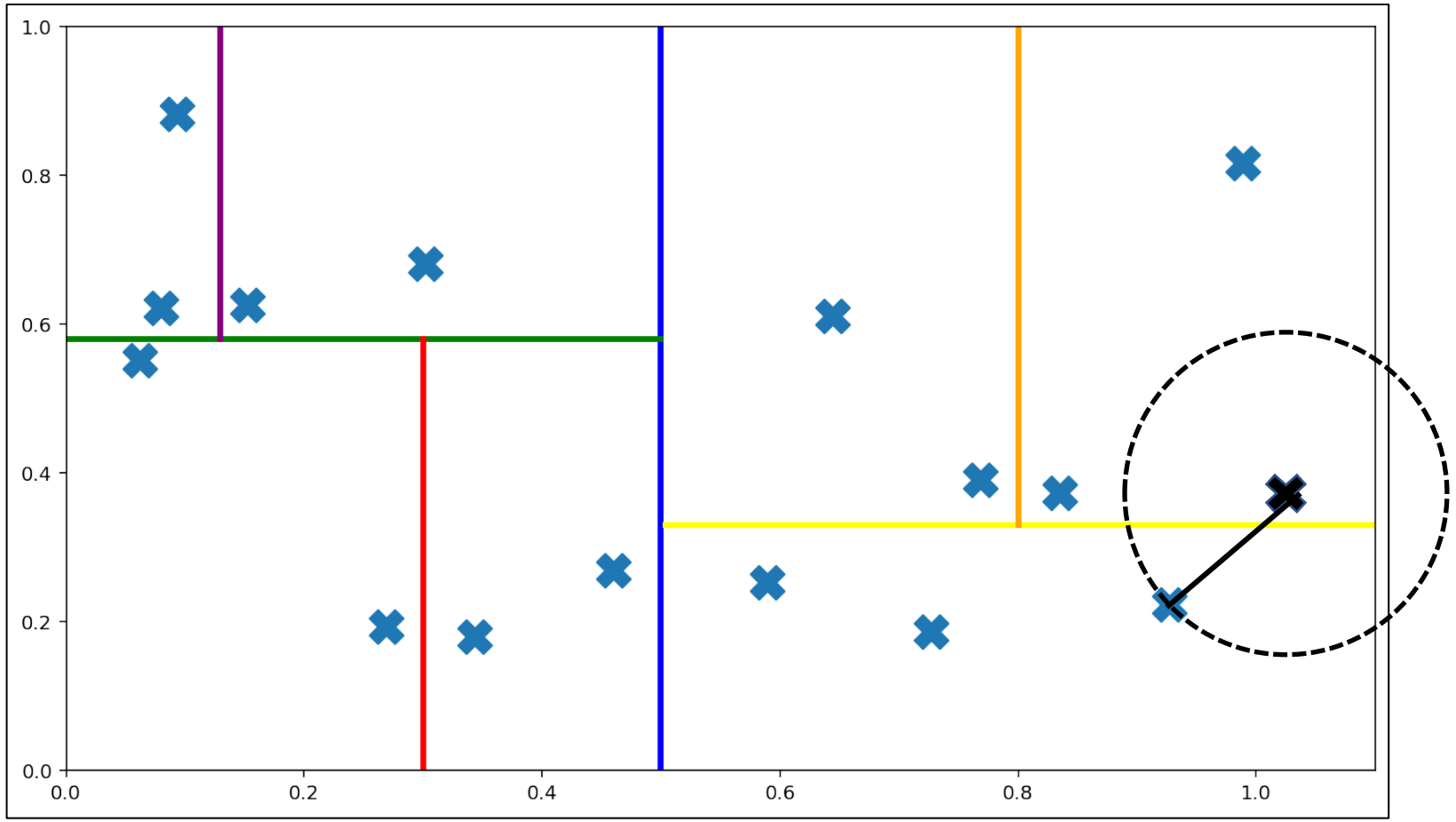
Поиск ближайшего соседа

- Находим лист в котором лежит заданный элемент.
- Полным перебором в данном листе ищем ближайший элемент к заданному.
- Строим вокруг заданного элемента сферу.

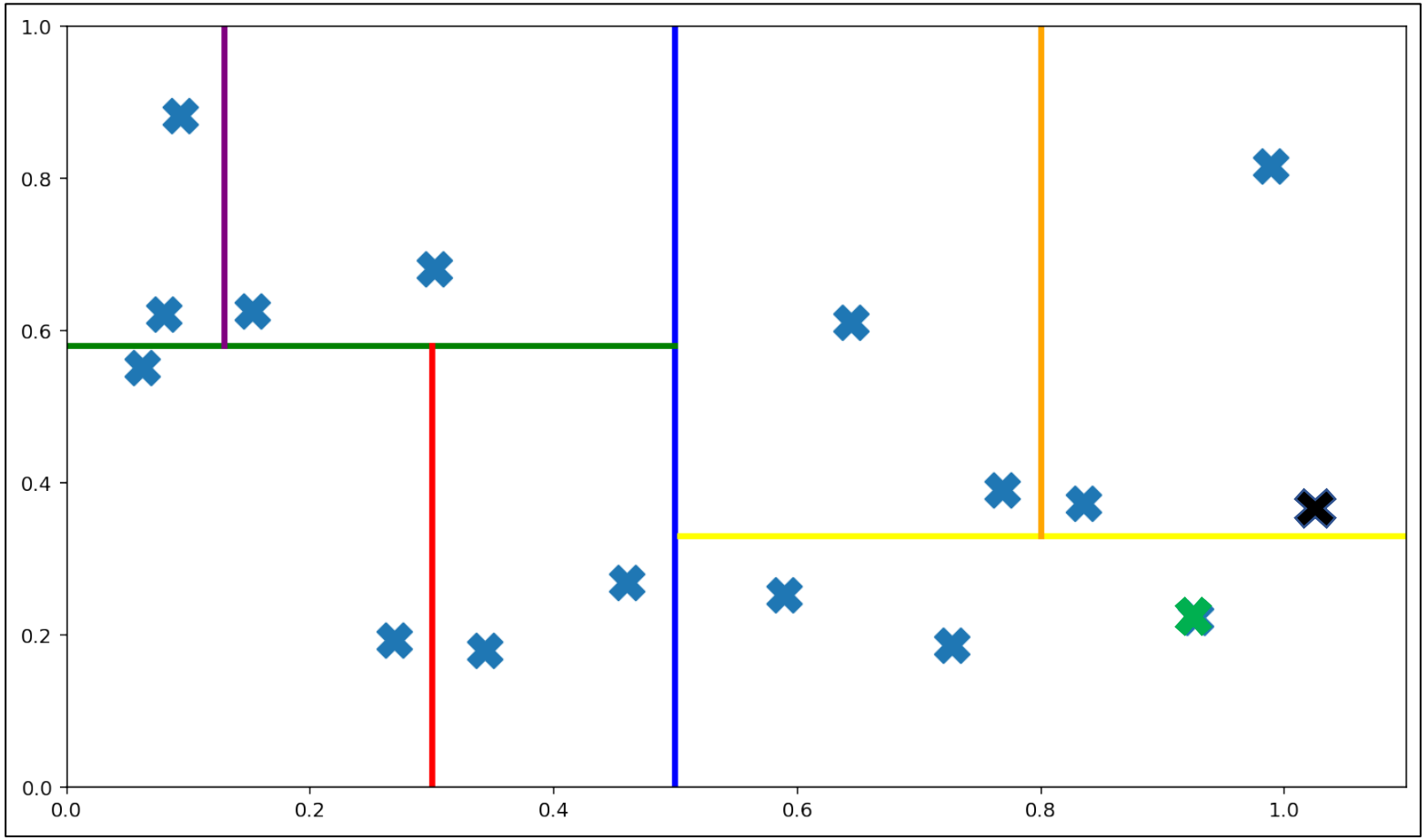


Поиск ближайшего соседа

- Запускаем обход дерева с корня.
- При достижение листа запускаем перебор элементов.
- Обновляем радиус сферы.



Поиск ближайшего соседа



Время работы и память

- Построение: $O(kn + n \log n)$
- Высота: $O(\log n)$
- Поиск ближайшего соседа:
  - в среднем за $O(\log n)$
  - в худшем случае за $O(n)$
- Память $O(n)$

Плюсы и минусы к-мерного дерева

Плюсы:

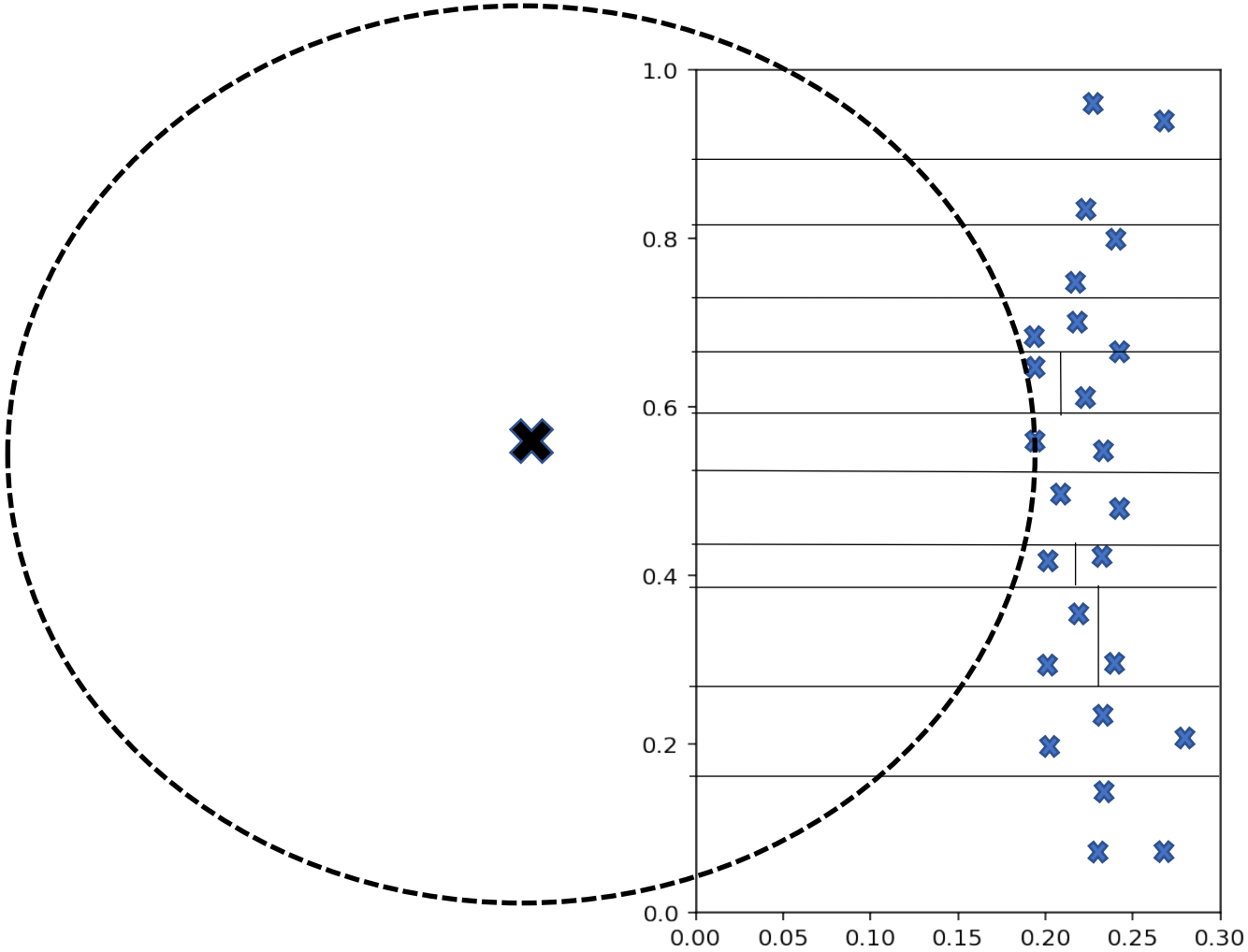
- Быстрый поиск (в среднем).
- Небольшие затраты памяти.

Минусы:

- Не всегда сбалансировано.
- Возможны ситуации с плохой асимптотикой.
- Трудоемкое построение.
- Не эффективен при большой размерности пространства.

Плохая ситуация

Заданный элемент находится слишком далеко от bounding box, в результате чего сфера пересекает слишком много областей



Инвертированный индекс

- **Инвертированный индекс** - это индекс хранящий сопоставление содержимого, такого как слова, числа, вектора, с его местоположениями.
- **Мотивация**:

1. запросы к инвертированному индексу позволяют избежать оценки расстояний между запросом и каждой точкой в наборе данных и, таким образом, обеспечивают существенное ускорение по сравнению с полным поиском
2. Эффективен в пространствах высокой размерности

Примеры использования

- В поисковых системах.
- В компьютерном зрении для поиска сходства (классификация изображений, редактирование изображений и т. д.).
- В биоинформатике в процессе определения коротких последовательностей нуклеиновой кислоты.

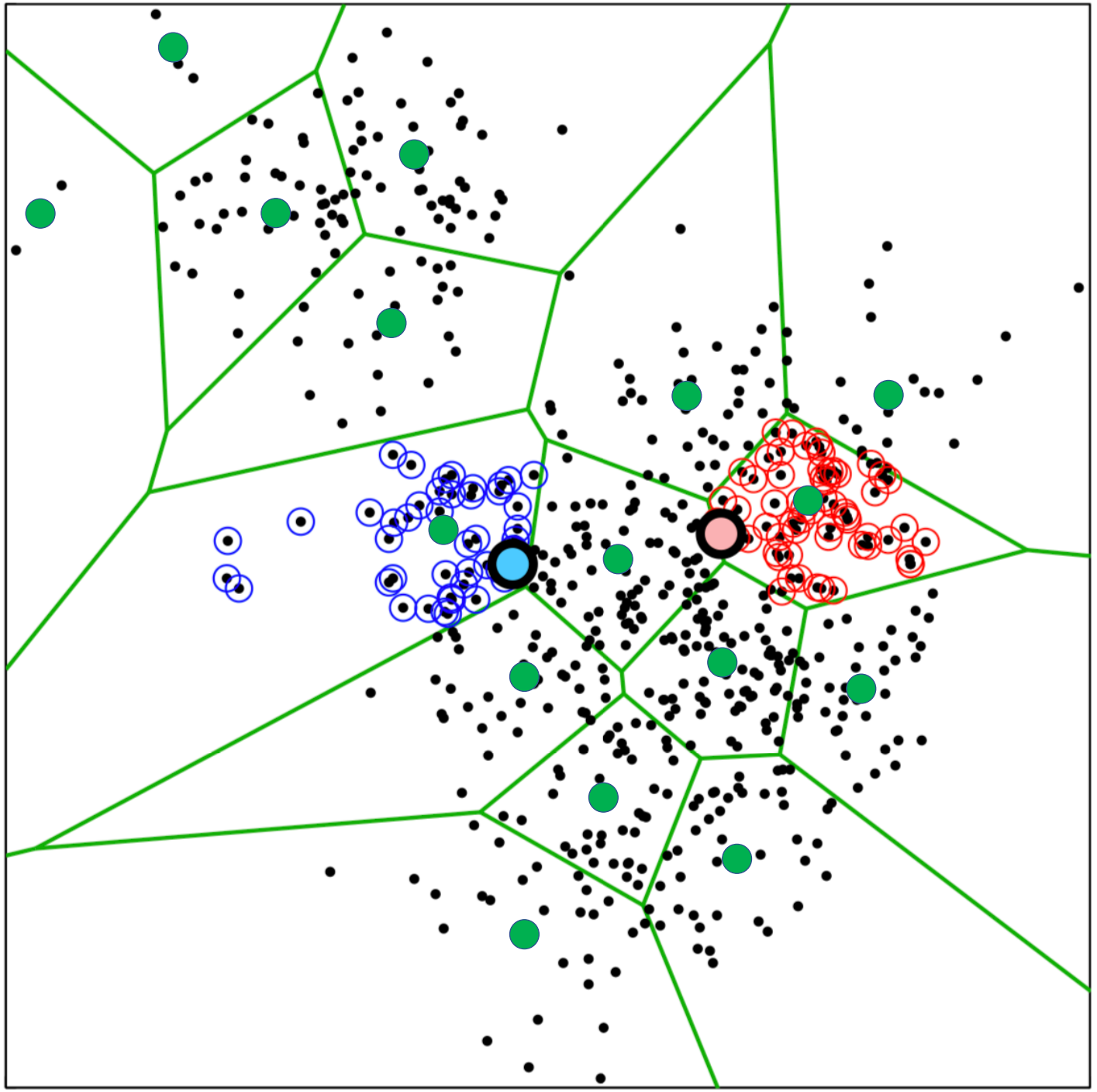
Поиск k-ближайших соседей

- **Шаг 1.** Запускаем на множестве d-мерных векторов алгоритм кластеризации (k-means).
- **Шаг 2.** Вычисляем расстояние от запроса до каждого центра кластера.
- **Шаг 3.** Делаем полный перебор внутри ближайшего кластера.

- центр кластера

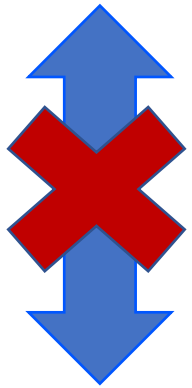
- запрос

- запрос



Противоречие

- Необходимо проверять несколько ближайших кластеров.
- Чем больше кластеров, тем лучше.



- Нахождение лучших кластеров должно быть быстрым.

Инвертированный мульти-индекс

Идея:

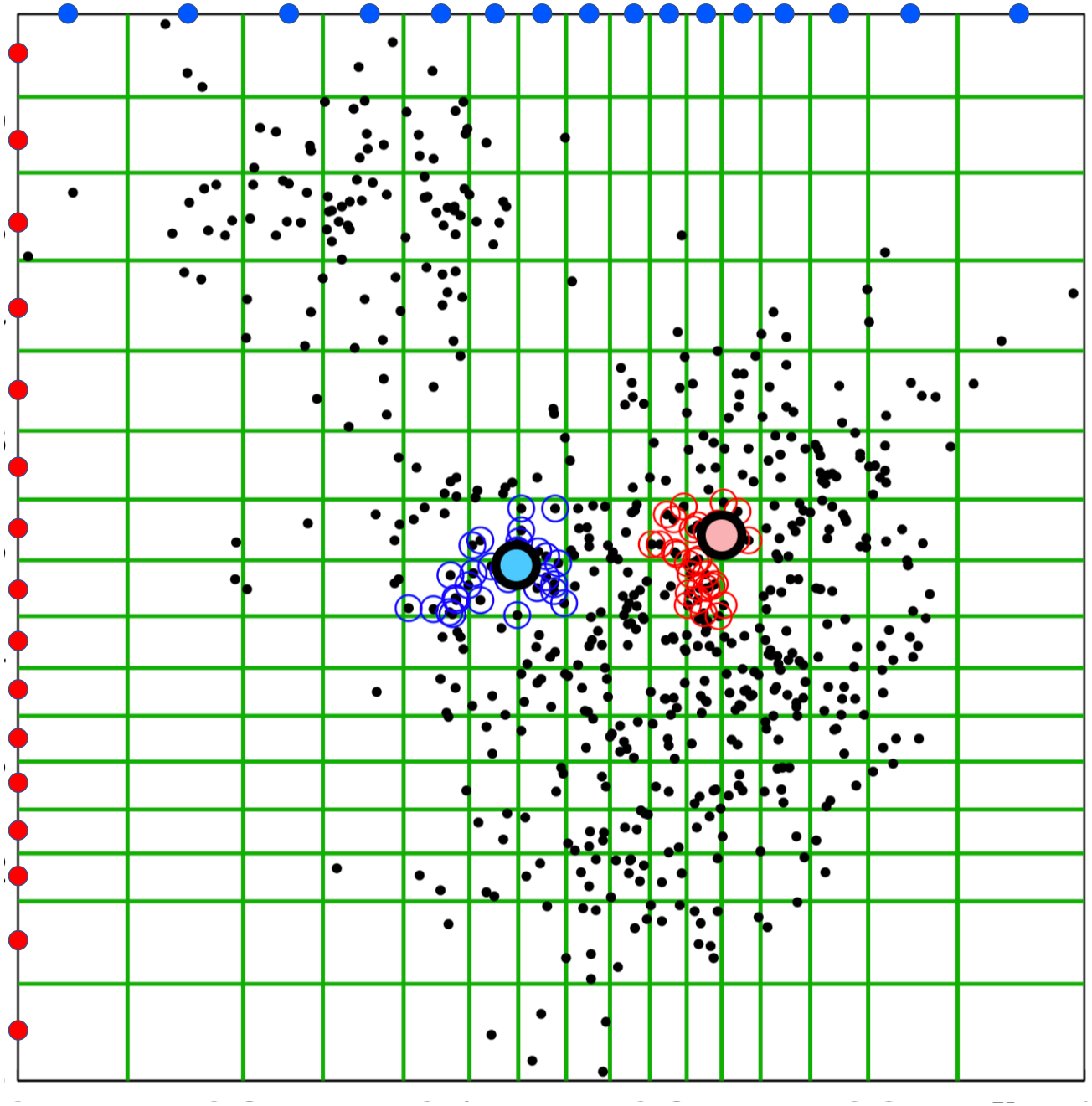
Заменить k-means кластеризацию исходных d-мерных векторов на две независимые кластеризации первых и вторых половин.

Получаем:

гораздо больше кластеров для того же K.

Недостаток:

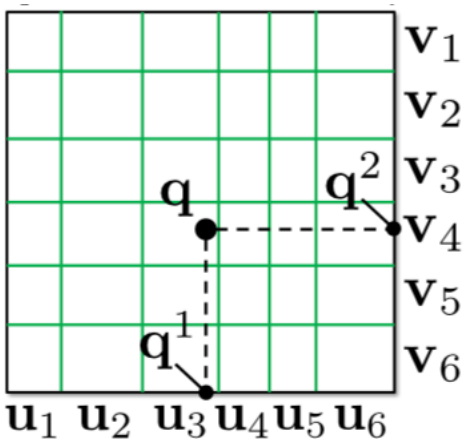
Кластеры могут быть пустыми.



Поиск с инвертированным мульти-индексом

- **Шаг 1**. Разбиваем запрос на две части и для каждой части считаем расстояние для соответствующих центроидов.
- **Шаг 2**. Считаем общее расстояние по двум частям.
- **Шаг 3.** Составляем очередь обхода.

Поиск с инвертированным мульти-индексом



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

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

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

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

$[\mathbf{u}_{\alpha(i)}\ \mathbf{v}_{\beta(j)}]$	(i,j)	$r(i)+s(j)$
$[\mathbf{u_3}\ \mathbf{v_4}]$	(1,1)	0.6 (0.5+0.1)
$[\mathbf{u_4}\ \mathbf{v_4}]$	(2,1)	0.8 (0.7+0.1)
$[\mathbf{u_3}\ \mathbf{v_3}]$	(1,2)	2.5 (0.5+2)
$[\mathbf{u_4}\ \mathbf{v_3}]$	(2,2)	2.7 (0.7+2)
$[\mathbf{u_3}\ \mathbf{v_5}]$	(1,3)	3.5 (0.5+3)
$[\mathbf{u_4}\ \mathbf{v_5}]$	(2,3)	3.7 (0.7+3)
$[\mathbf{u_5}\ \mathbf{v_4}]$	(3,1)	4.1 (4+0.1)
$[\mathbf{u_5}\ \mathbf{v_3}]$	(3,2)	6 (4+2)
$[\mathbf{u_3}\ \mathbf{v_2}]$	(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	$\mathbf{v}_4$
2	2.5	2.7	6	8	10	11	$\mathbf{v}_3$
3	3.5	3.7	7	9	11	12	$\mathbf{v}_5$
4	6.5	6.7	10	12	14	15	$\mathbf{v}_2$
5	7.5	7.7	11	13	15	16	$\mathbf{v}_6$
6	11.5	11.7	15	17	19	20	$\mathbf{v}_1$
	$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$	$\mathbf{u}_2$	$\mathbf{u}_1$	$\mathbf{u}_6$	

	1	2	3	4	5	6	
1	**0.6**	0.8	4.1	6.1	8.1	9.1	$\mathbf{v}_4$
2	2.5	2.7	6	8	10	11	$\mathbf{v}_3$
3	3.5	3.7	7	9	11	12	$\mathbf{v}_5$
4	6.5	6.7	10	12	14	15	$\mathbf{v}_2$
5	7.5	7.7	11	13	15	16	$\mathbf{v}_6$
6	11.5	11.7	15	17	19	20	$\mathbf{v}_1$
	$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$	$\mathbf{u}_2$	$\mathbf{u}_1$	$\mathbf{u}_6$	

	1	2	3	4	5	6	
1	0.6	**0.8**	4.1	6.1	8.1	9.1	$\mathbf{v}_4$
2	2.5	2.7	6	8	10	11	$\mathbf{v}_3$
3	3.5	3.7	7	9	11	12	$\mathbf{v}_5$
4	6.5	6.7	10	12	14	15	$\mathbf{v}_2$
5	7.5	7.7	11	13	15	16	$\mathbf{v}_6$
6	11.5	11.7	15	17	19	20	$\mathbf{v}_1$
	$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$	$\mathbf{u}_2$	$\mathbf{u}_1$	$\mathbf{u}_6$	

	1	2	3	4	5	6	
1	0.6	0.8	4.1	6.1	8.1	9.1	$\mathbf{v}_4$
2	**2.5**	2.7	6	8	10	11	$\mathbf{v}_3$
3	3.5	3.7	7	9	11	12	$\mathbf{v}_5$
4	6.5	6.7	10	12	14	15	$\mathbf{v}_2$
5	7.5	7.7	11	13	15	16	$\mathbf{v}_6$
6	11.5	11.7	15	17	19	20	$\mathbf{v}_1$
	$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$	$\mathbf{u}_2$	$\mathbf{u}_1$	$\mathbf{u}_6$	

	1	2	3	4	5	6	
1	0.6	0.8	4.1	6.1	8.1	9.1	$\mathbf{v}_4$
2	2.5	**2.7**	6	8	10	11	$\mathbf{v}_3$
3	3.5	3.7	7	9	11	12	$\mathbf{v}_5$
4	6.5	6.7	10	12	14	15	$\mathbf{v}_2$
5	7.5	7.7	11	13	15	16	$\mathbf{v}_6$
6	11.5	11.7	15	17	19	20	$\mathbf{v}_1$
	$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$	$\mathbf{u}_2$	$\mathbf{u}_1$	$\mathbf{u}_6$	

	1	2	3	4	5	6	
1	0.6	0.8	4.1	6.1	8.1	9.1	$\mathbf{v}_4$
2	2.5	2.7	6	8	10	11	$\mathbf{v}_3$
3	**3.5**	3.7	7	9	11	12	$\mathbf{v}_5$
4	6.5	6.7	10	12	14	15	$\mathbf{v}_2$
5	7.5	7.7	11	13	15	16	$\mathbf{v}_6$
6	11.5	11.7	15	17	19	20	$\mathbf{v}_1$
	$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$	$\mathbf{u}_2$	$\mathbf{u}_1$	$\mathbf{u}_6$	

Плюсы и минусы мульти-индекса

Плюсы:

- Эффективен для больших размерностей.
- Намного точней, чем базовая версия алгоритма.

Минусы:

- Кластеры могут быть пустыми.
- Работает чуть медленнее, чем базовая версия.

Список источников

К-мерное дерево:

- https://en.wikipedia.org/wiki/K-d_tree
  https://habr.com/ru/post/312882/

Инвертированный индекс (+ мульти):

- https://en.wikipedia.org/wiki/Inverted_index
- https://cache-ash04.cdn.yandex.net/download.yandex.ru/company/cvpr2012.pdf
- https://www.youtube.com/watch?v=UUm4MOyVTnE

Список вопросов

- Опишите два любых способа деления узла при построение kd-tree
- Как найти ближайшего соседа в kd-tree к заданной точке?
- В чем причина неточности поиска k-ближайших с помощью инвертированного индекса?