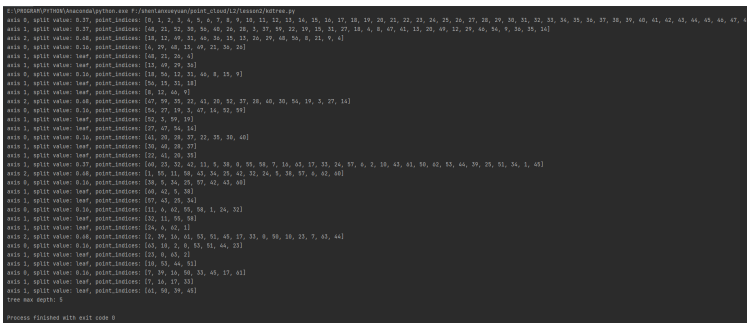
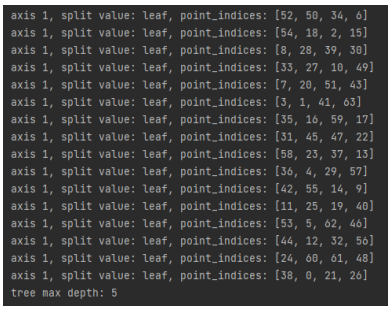
第二章作业

1. Kd-Tree

（1）建树

All Nodes	Leaf Nodes
	

（2）搜索

```
k-NN search:
12 - 0.18
21 - 0.35
25 - 0.35
55 - 0.43
31 - 0.46
13 - 0.48
52 - 0.59
0 - 0.63
In total 64 comparison operations.
[12 21 25 55 31 13 52  0]
[0.18264528 0.35297464 0.3536302  0.42667876 0.46259296 0.47892959
 0.58743535 0.63361626]
Radius search:
12 - 0.18
21 - 0.35
25 - 0.35
55 - 0.43
31 - 0.46
In total 5 neighbors within 0.500000.
There are 32 comparison operations.
```

k-NN search和Radius search的结果相互印证

2. Octree

（1）建树

这里对比了一下暴力分类和位运算的速度差异

暴力分类	位运算

暴力分类	位运算

`# 根据8种情况将父结点种的点放入相应的子节点中：粗暴分类法`
`for point_index in point_indices:`
`  point = db[point_index, :]`
`  if point[0] < center[0] and point[1] < center[1] and point[2] < center[2]:`
`  children_point_index[0].append(point_index)`
`  elif point[0] >= center[0] and point[1] < center[1] and point[2] < center[2]:`
`  children_point_index[1].append(point_index)`
`  elif point[0] < center[0] and point[1] >= center[1] and point[2] < center[2]:`
`  children_point_index[2].append(point_index)`
`  elif point[0] >= center[0] and point[1] >= center[1] and point[2] < center[2]:`
`  children_point_index[3].append(point_index)`
`  elif point[0] < center[0] and point[1] < center[1] and point[2] >= center[2]:`
`  children_point_index[4].append(point_index)`
`  elif point[0] >= center[0] and point[1] < center[1] and point[2] >= center[2]:`
`  children_point_index[5].append(point_index)`
`  elif point[0] < center[0] and point[1] >= center[1] and point[2] >= center[2]:`
`  children_point_index[6].append(point_index)`
`  elif point[0] >= center[0] and point[1] >= center[1] and point[2] >= center[2]:`
`  children_point_index[7].append(point_index)`
	`# 根据8种情况将父结点种的点放入相应的子节点中：位运算法`
	`for point_index in point_indices:`
	`  point_db = db[point_index, :]`
	`  morton_code = 0`
	`  if point_db[0] > center[0]:`
	`  morton_code = morton_code | 1`
	`  if point_db[1] > center[1]:`
	`  morton_code = morton_code | 2`
	`  if point_db[2] > center[2]:`
	`  morton_code = morton_code | 4`
	`  children_point_index[morton_code].append(point_index)`

`tree max depth: 5` `Construction takes 685.169ms`	`tree max depth: 5` `Construction takes 387.963ms`

(2) 搜索

k-NN Search	Radius Search (before optimization)	Radius Search (after optimization)

`k-NN search:`
`48423 - 0.03`
`53889 - 0.04`
`15441 - 0.04`
`51126 - 0.04`
`30379 - 0.04`
`40 - 0.04`
`14175 - 0.05`
`28599 - 0.05`
`In total 18 comparison operations.`
`index: [48423 53889 15441 51126 30379  40 14175 28599]`
`distance: [0.0296488  0.03587816 0.03969907 0.04173917 0.04292096 0.04339923`
` 0.04541451 0.05062956]`
`k-NN search takes 9.973ms`
	`Radius search normal:`
	`55970 - 0.02`
	`5246 - 0.02`
	`18381 - 0.02`
	`24022 - 0.02`
	`14803 - 0.03`
	`10777 - 0.03`
	`26280 - 0.03`
	`16146 - 0.03`
	`42444 - 0.03`
	`46694 - 0.04`
	`53458 - 0.04`
	`26724 - 0.04`
	`1281 - 0.04`
	`In total 13 neighbors within 0.040000.`
	`There are 168 comparison operations.`
	`Search takes 1.994ms`
		`Radius search fast:`
		`55970 - 0.02`
		`5246 - 0.02`
		`18381 - 0.02`
		`24022 - 0.02`
		`14803 - 0.03`
		`10777 - 0.03`
		`26280 - 0.03`
		`16146 - 0.03`
		`42444 - 0.03`
		`46694 - 0.04`
		`53458 - 0.04`
		`26724 - 0.04`
		`1281 - 0.04`
		`In total 13 neighbors within 0.040000.`
		`There are 168 comparison operations.`
		`Search takes 1.994ms`

1. 为了打印出result-set list，首先设置了较小的radius，可以看到Radius Search在优化前后均找到13个相同的点
2. 为了对比优化前后的速度，关闭打印，设置了较大的radius，结果如下图，可以看到有较大的速度提升：

```
Radius search normal:
Search takes 354.054ms

Radius search fast:
Search takes 124.666ms
```

3. Benchmark

将三种算法（kdtree，octree，brute）的速度进行了对比（iteration=10：十次实验取平均值）：

```
db_np.shape (124668, 3)
iteration_num: 10
octree --------------
Octree: build 5115.738, knn 0.499, radius 0.599, brute 9.674
kdtree --------------
Kdtree: build 130.177, knn 4.289, radius 0.299, brute 9.874
```

可以看出octree建树较kdtree慢，但是在knn任务上，octree search比kdtree search快很多，而在radius search任务上，kdtree速度更快；