KdTrees and all that

Week 9+

Reference : Ivezic et al Chapter 2

References

- Ivezic et al al, Chapter 2

http://andrewd.ces.clemson.edu/courses/cpsc805/references/nearest_search.pdf

https://jakevdp.github.io/blog/2013/04/29/benchmarking-nearest-neighbor-searches-in-python/

(Benchmarking of several python codes for KdTree)

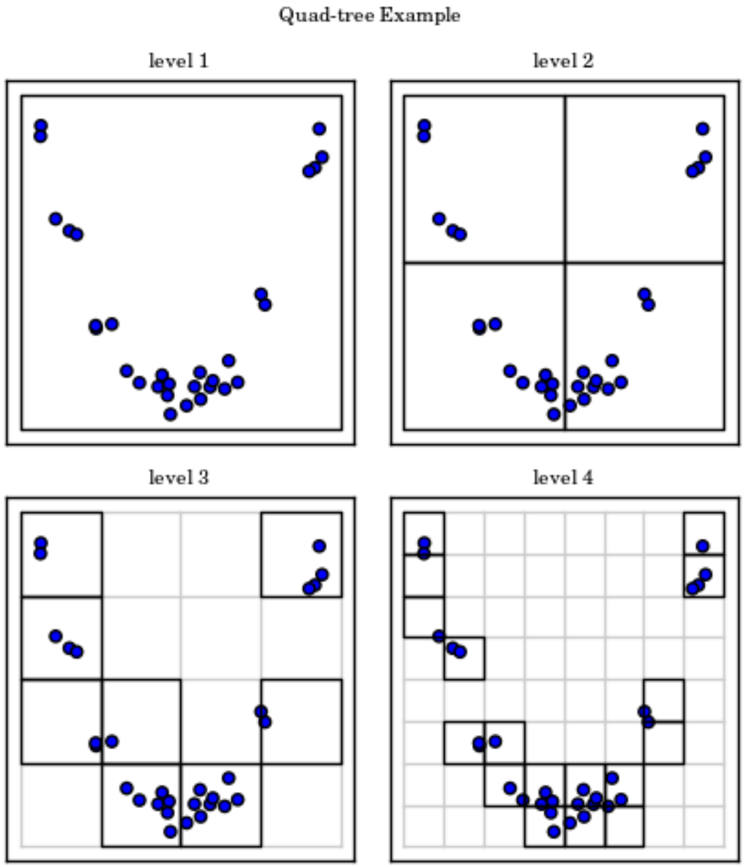
Introduction

- Question : How to accelerate nearest neighbor searches?

Problem : Brute Force search algorithms don't scale well with N $O(N^2)$.

Vectorizing (ala numpy, matlab) speeds up by a factor of 100, but with increased usage of memory.

Quad Trees and Oct Trees



Quad and *Oct* trees work in 2 and 3 dimensions respectively.

A quad tree is a simple data structure in which each tree node has exactly four children, representing its four quadrants. Each node is defined by its four numbers: left, right, top, and bottom quadrants.

By grouping points in this manner, one can progressively constrain the distance between a test point and a group of Points in a tree using the bounding box of each group of points to provide a lower bound on the distance between the query point and any point in the group.

If lower bound > best candidate nearest neighbor distance it can prune the group from the search

Speed up in Quad Tree

- Cost of nearest-neighbour search reduces to O(logN) for a single-query point.
- Building up the tree takes O(N logN) time to construct.

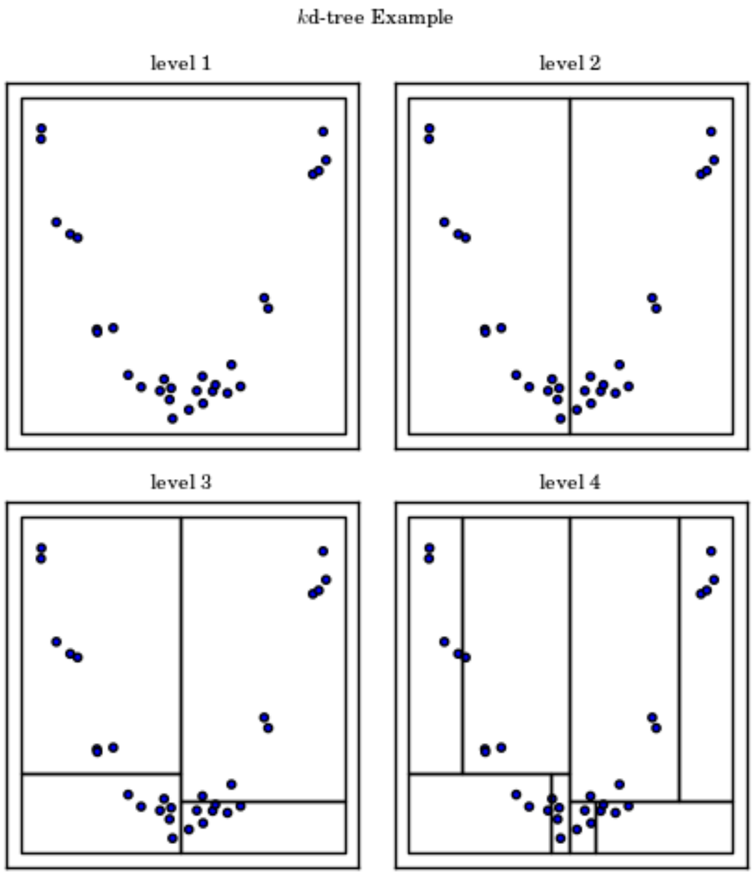
Motivation for Kd Trees

- Generalizing quad-tree and oct-tree to higher dimensions would imply that in D-dimensions, we would build a tree with 2^D children per node.

- Unfortunately size of tree would blow up for large D. For D=10, each node would would require 2^{10} children (1024). To further divide each dimension into four units would require 10^6 nodes. This is called "Curse of Dimensionality"

- For D=100, would require 10^{15} petabytes of storage (Total volume of internet worldwide traffic in 2010)

Introduction to Kd Trees



*K*d tree is a *k*-dimensional generalization of quad tree.

It is implemented as a *binary* tree. Each node has two children.

Top node of a *k* d tree is a *D*-dimensional hyperrectangle which contains the entire dataset.

To create the subnodes, volume is split into two regions along a simple dimension and this procedure is repeated recursively until the lowest node contains a specified number of points.

Kd Trees in Python

- KdTree available in scipy as well as sklearn.
  Fast Kdtree available in scipy is shown below

```
import numpy as np
from scipy.spatial import KDTree
np.random.seed(0)
X=np.random.random((100000,3))
kdt = cKDTree(X) #build the Kdtree
kdt.query(X, k=2) #query for two neighbours
```

The above query takes 949 ms . OTOH a brute force search takes 23 minutes

Problems with Kd Trees

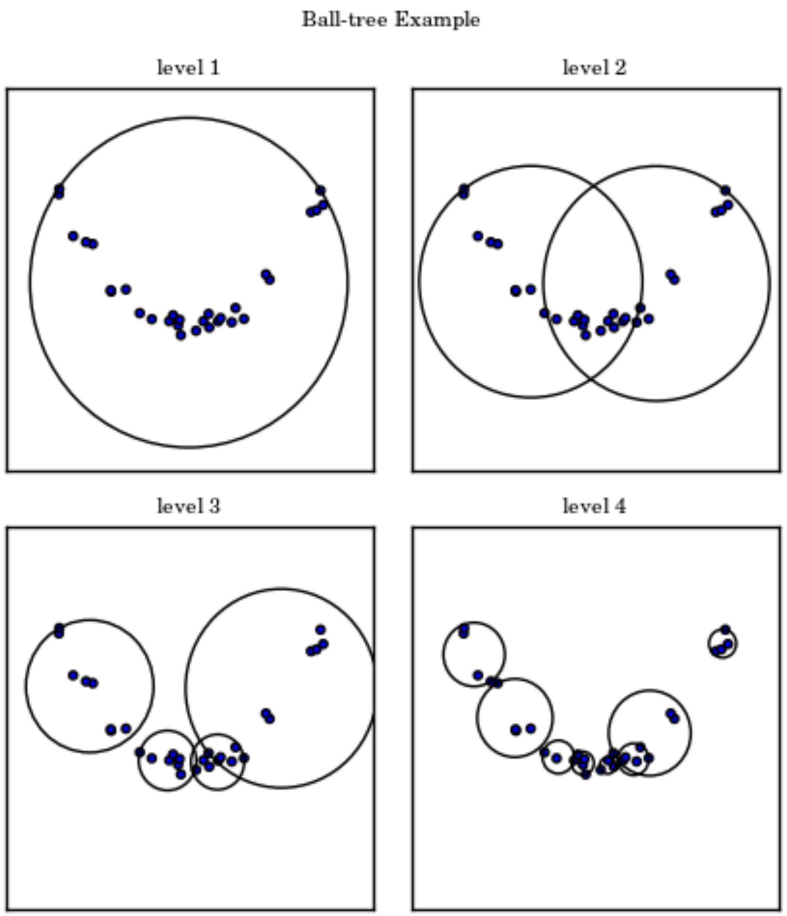
- Kdtree relies on rectlinear splitting of the data space, it is also subject to the curse of dimensionality
- Because the Kd-Tree splits along a single dimension in each level, one must go D levels deep before each dimension has been split.
- For D=100, we must create $2^{100} \sim 10^{30}$ nodes in order to split each dimension once.
- For N points in D dimensions, a kd tree will loose efficiency when $D \gg \log_2 N$

Ball Trees

- If x_1 is far from x_2 and x_2 is near x_3, then x_1 is also far from x_3

Consequence of the Triangle inequality (which also applies to Euclidean distances)

$D(x_1,x_2) + D(x_2,x_3) >= D(x_1,x_3)$



- Instead of building rectlinear nodes in D dimensions, ball-tree construction builds hyper-spherical nodes.

- Each node defined by a centroid c_i and a radius r_i, such that distance $D(y,c_i) <= r_i$ for every point contained in the node. With this Construction, given a point x outside the node One can show that for any point y in the node

$$[D(x, c_i) - r_i] \leq D(x, y) \leq [D(x, c_i) + r_i]$$

Ball Trees in Python

```
import numpy as np
from sklearn.neighbours import BallTree
np.random.seed(0)
X=np.random.random((1000,3))
bt = BallTree(X)  # build the balltree
bt.query(X, k=2)
```

Useful links to scipy/scikit learn documentation

Ball Tree

http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html

KDTree (scipy)

https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.spatial.KDTree.query.html
(Note that scipy has both KDTree and cKDTree)

KDTree (sklearn)
http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html

Matching of Astrophysical Catalogs

```
radec_mags = NP.dstack((magnitudes['ra'],magnitudes['dec']))[0]
kdtree_object = spatial.cKDTree(radec_twomass)
distances, indexes =
kdtree_object.query(radec_mags,distance_upper_bound=match_radius/3600.)
nomatch = NP.isinf(distances)
indexes = indexes[~nomatch]
n_match = NP.sum(~nomatch)
```