Locality sensitive hashing for approximate NN search

Limitations of KD-trees

Motivating alternative approaches to approximate NN search

- KD-trees are cool, but
  - Non-trivial to implement efficiently
  - Problems with high-dimensional data

KD-trees in high dimensions

- Unlikely to have any data points close to query point
- Once "nearby" point is found, the search radius is likely to intersect many hypercubes in at least one dim.
- Not many nodes can be pruned
- Can show under some conditions that you visit at least 2^d nodes

Moving away from exact NN search

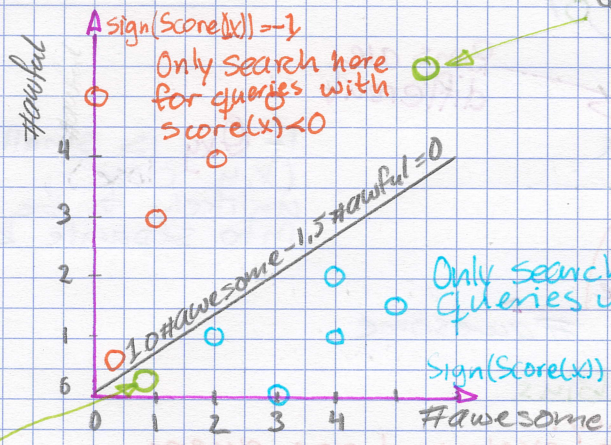
- Approximate neighbor finding.
  - Don't find exact neighbor, but that's okay for many applications
    - Out of millions of articles, do we need the closest article or just one that's pretty similar? Do we even fully trust our measure of similarity???
- Focus on methods that provide good probabilistic guarantees on approximation

LSH as an alternative to KD-trees

Simple "binning" of data into 2 bins
Using bins for NN search

2D Data	Sign(Score)	Bin index
$x_1 = [0,5]$	-1	0
$x_2 = [1,3]$	-1	0
$x_3 = [3,0]$	1	1
...	...	...

candidate neighbors if Score(x) < 0



Bin	0	1	HASH
List containing indices of datapoints	{1, 2, 4, 7}	{3, 5, 6, 8, ...}	TABLE

Provides approximate NN

Nearest neighbor to query point found? NO → search for NN amongst this set

Using random lines to partition points

A practical implementation

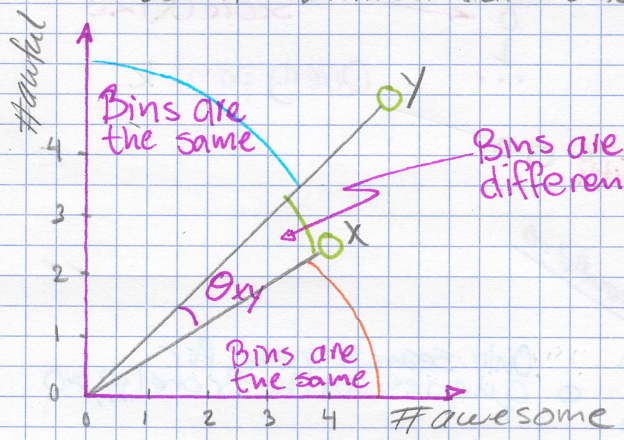
Three potential issues with simple approach

1. Challenging to find good line
2. Poor quality solution:
   - Points close together get split into separate bins
3. Large computational cost:
   - Bins might contain many points, so still searching over large set for each NN query

How to define the line?

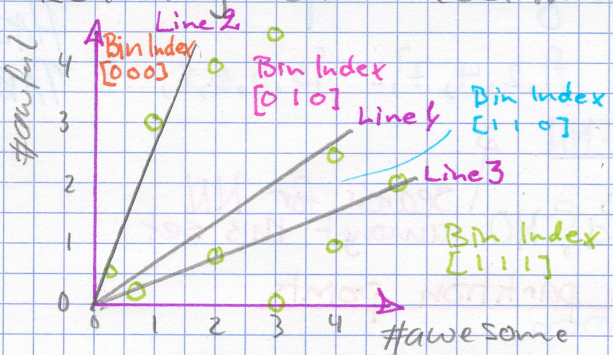
Crazy idea:
Define line randomly!

How bad can a random line be?
Goal: If x, y are close (according to cosine similarity), want binned values to be the same



If θ_{xy} is small (x, y close), unlikely to be placed into separate bins

Defining more bins
Improving efficiency:
Reducing # points examined per query
Reducing search cost through more bins



Using score for NN search

2D Data	Sign ($Score_1$)	Bin 1 index	Sign ($Score_2$)	Bin 2 index	Sign ($Score_3$)	Bin 3 index
$x_1 = [0, 5]$	-1	0	-1	0	-1	0
$x_2 = [1, 3]$	-1	0	-1	0	-1	0
$x_3 = [3, 0]$	1	1	1	1	1	1

Bin	[000] =0	[001] =1	[010] =2	[011] =3	[100] =4	[101] =5	[110] =6	[111] =7
Data Indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

↳ search for NN amongst this set

Searching neighboring bins

Improving search quality by searching neighboring bins

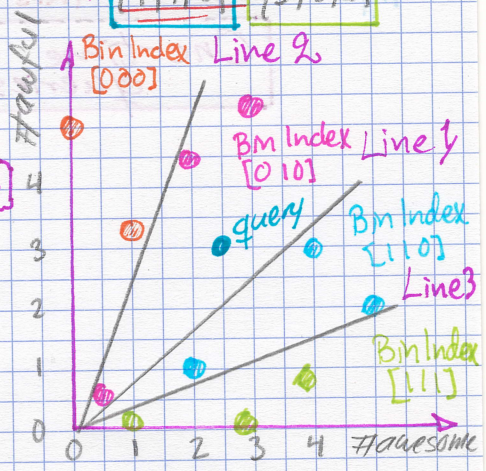
Further bin (flip 2 bits)

Bin	[000] =0	[001] =1	[010] =2	[011] =3	[100] =4	[101] =5	[110] =6	[111] =7
Data indices:	{1,2}		{4,8,11}				{7,9,10}	{3,5,6}

Next closest bins (flip 1 bit)

Query point, here but is NN?

Not necessarily



Even worse than before... Each line can split points. Sacrificing accuracy for speed

Quality of retrieved NN can only improve with searching more bins

Algorithm:
Continue searching until computational budget is reached or quality of NN good enough

LSH recap

kd-tree competitor data structure

- Draw h random lines
- Compute "score" for each point under each line and translate to binary index
- Use h-bit binary vector per data points as bin index
- Create hash table

- For each query point x, search bin(x), then neighboring bins until time limit

LSH in higher dimensions

Cost of binning points in d-dim

$$\text{Score}_i(x) = v_1^{(i)} \#\text{awesome} + v_2^{(i)} \#\text{awful} + v_3^{(i)} \#\text{great}$$

i^{th} hyperplane

Per data point, need d multiplies to determine bin index per plane

In high-dim, (and some applications) this is often a sparse matt.

One-time cost offset if many queries of fixed dataset