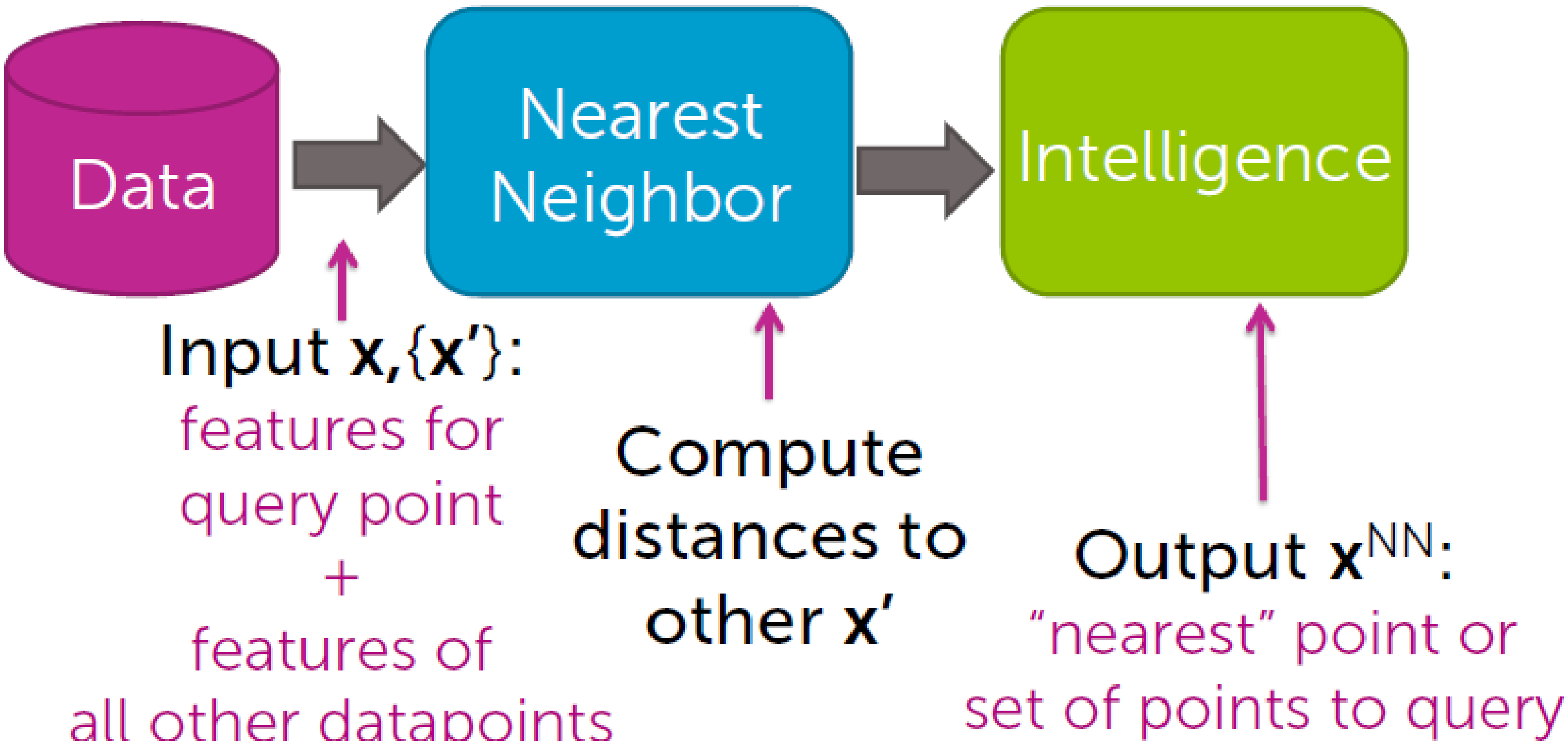
DATA SCIENCE WITH MACHINE LEARNING: RETRIEVAL

This lecture is
based on course by E. Fox and C. Guestrin, Univ of Washington

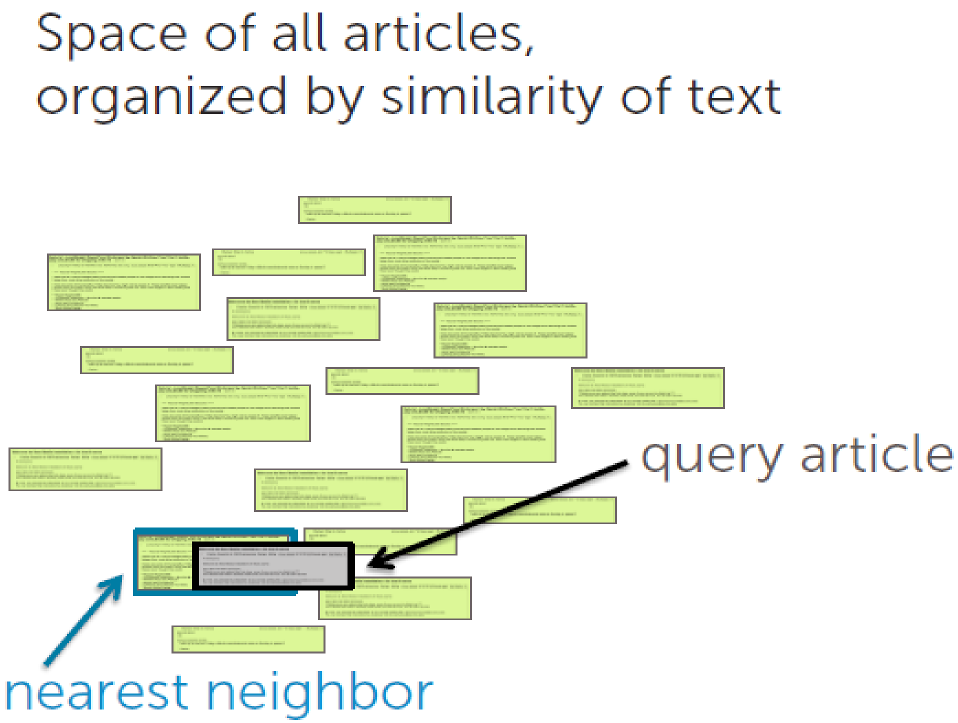
What is retrieval?

Search for related items



What is retrieval?

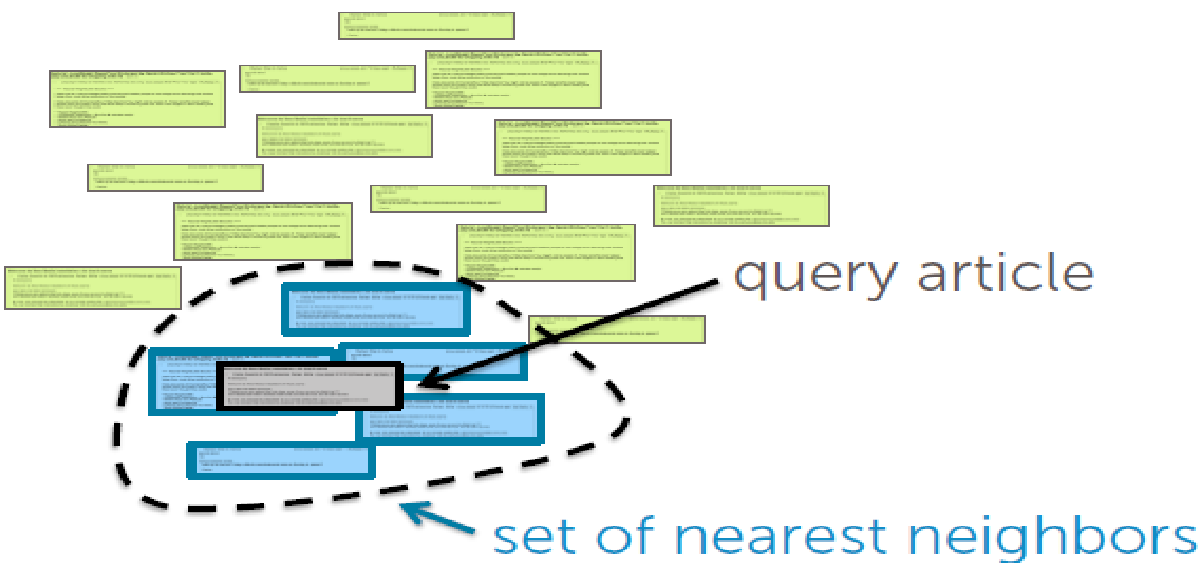
Retrieve "nearest neighbor" article



What is retrieval?

Or set of nearest neighbors

Space of all articles,
organized by similarity of text

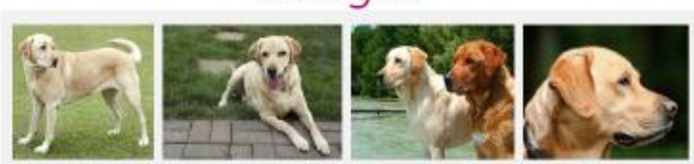


Retrieval applications

Just about everything…

Products

Images



Streaming content:
- Songs
- Movies
- TV shows
- …

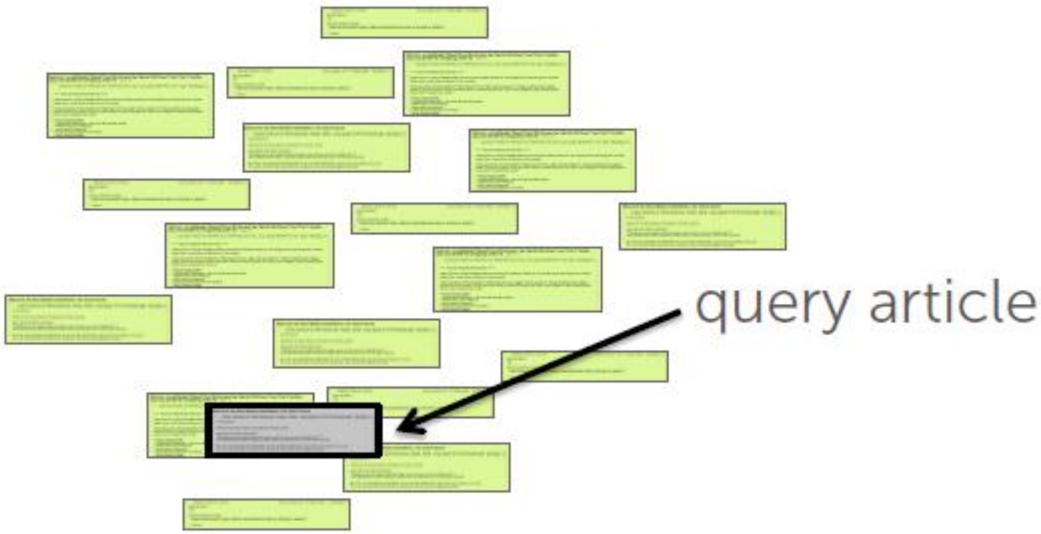
News articles

Social networks
(people you might want to connect with)

Retrieval
as
k-nearest neighbor search

1-NN search for retrieval

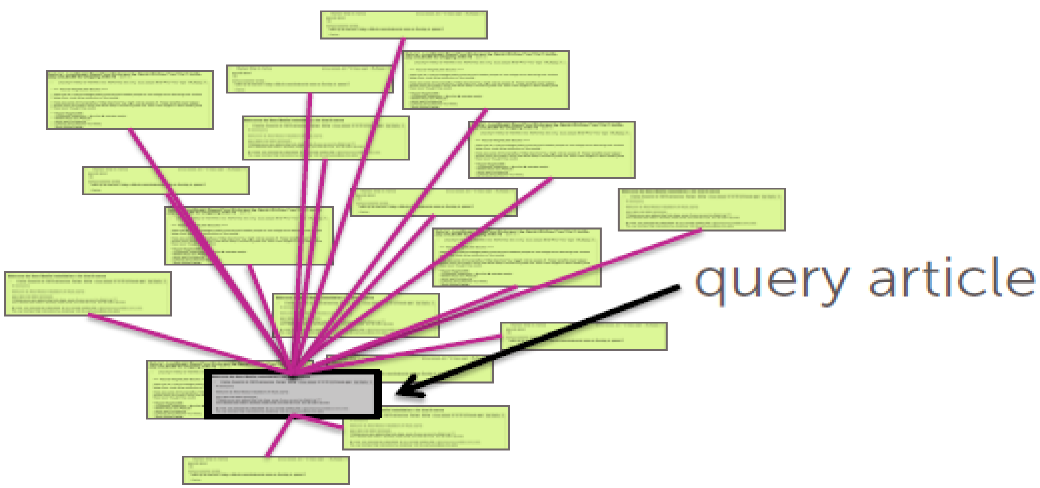
Space of all articles,
organized by similarity of text



1-NN search for retrieval

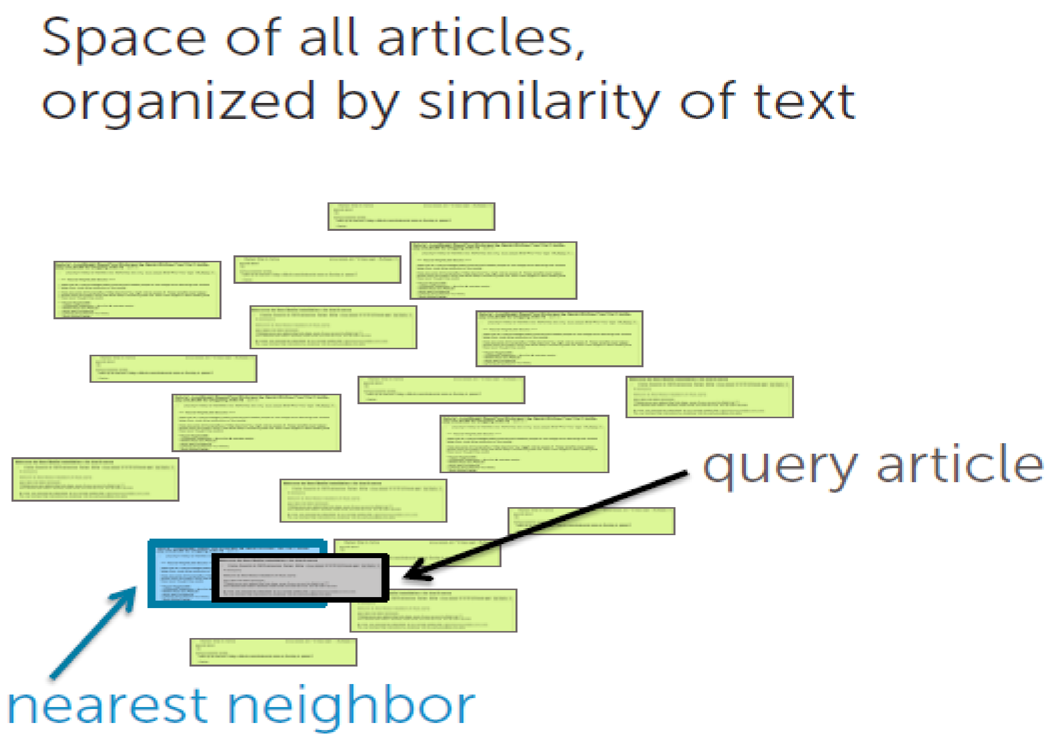
Compute distances to all docs

Space of all articles,
organized by similarity of text



1-NN search for retrieval

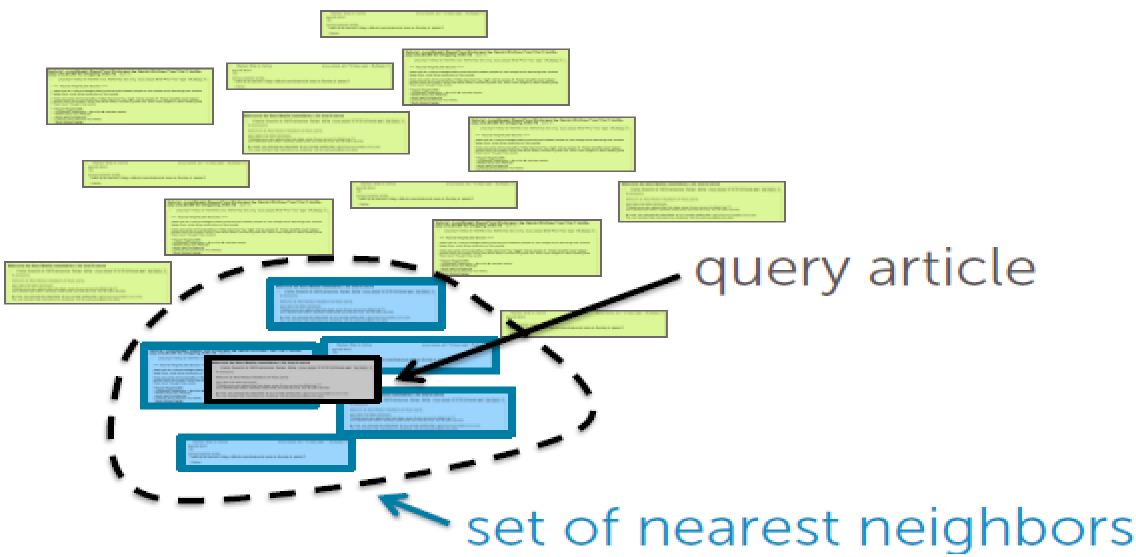
Retrieve "nearest neighbor"



1-NN search for retrieval

Or set of nearest neighbors

Space of all articles,
organized by similarity of text



1-NN algorithm

1 – Nearest neighbor

- **Input:** Query article : $\mathbf{x}_q$

  Corpus of documents (N docs)

  : $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
- **Output:** *Most* similar article $\leftarrow x^{NN}$

Formally: $x^{NN} = \min_{x_i} \text{distance}(x_q, x_i)$

1-NN algorithm

closest document

Initialize **Dist2NN** = ∞, [closest document] = ∅

For i=1,2,…,N

    Compute: δ = **distance(** [document i] x_i , [query document] x_q **)**

    If δ < **Dist2NN**

        set [closest document] = [document i] x_i

        set **Dist2NN** = δ

Return most similar document [closest document] $\leftarrow x^{NN}$

query document

document i
from corpus

closest document in
corpus to query article

k-NN algorithm

- **Input:** Query article  : $\mathbf{x}_q$

  Corpus of documents

   : $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$

- **Output:** *List of k* similar articles

Formally:

$$X^{NN} = \{X^{NN_1}, \ldots, X^{NN_k}\}$$

For all x_i not in X^{NN}, $\text{distance}(x_i, x_q) \geq \max_{x^{NN_j},\, j=1,\ldots,k} \text{distance}(x^{NN_j}, x_q)$

k-NN algorithm

sort first **k documents** by distance to query doc

Initialize **Dist2kNN** = sort($\delta_1,\ldots,\delta_k$) ← list of sorted distances

= ...rt(,... $_1$) $_k$ ← list of sorted docs

For i=k+1,…,N

query doc

Compute: δ = **distance(** $_i$ **,** $_q$ **)**

If δ < **Dist2kNN**[k] ← distance to kth NN (furthest NN in set)

find j such that δ > **Dist2kNN**[j-1] but δ < **Dist2kNN**[j]

remove furthest house and shift queue:

1:k] = -1]

inserting new article

Dist2kNN[j+1:k] = **Dist2kNN**[j:k-1]

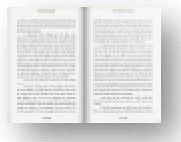
set **Dist2kNN**[j] = δ and [j] = $_i$

closest k docs to query doc

Return k most similar articles

Critical elements of NN search

Item (e.g., doc) representation

$\mathbf{x}_q \leftarrow$ 

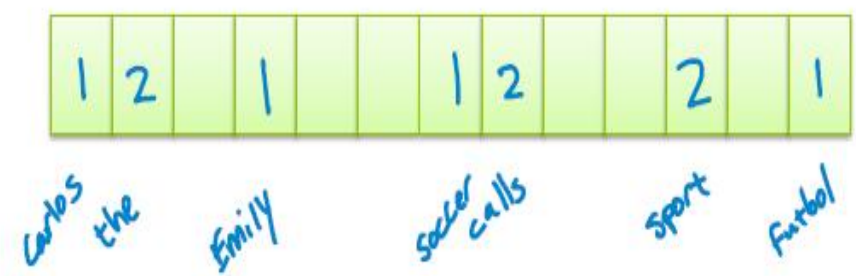
Measure of distance between items:

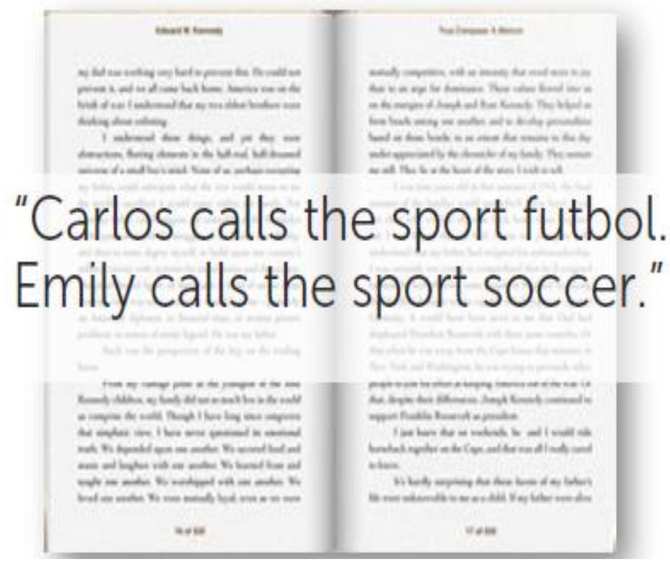
$\delta = \text{distance}(\mathbf{x}_i , \mathbf{x}_q)$

Document representation

Bag of words model

- Ignore order of words
- Count # of instances of each word in vocabulary





Document representation

Issues with word counts – Rare words

Common words in doc: “the”, “player”, “field”, “goal”

Dominate rare words like: “futbol”, “Messi”

Document representation

TF-IDF document representation

Emphasizes important words

- Appears frequently in document (common locally)

  Term frequency = word counts

- Appears rarely in corpus (rare globally)

  Inverse doc freq. = $\log \frac{\#\ \text{docs}}{1 + \#\ \text{docs using word}}$

Document representation

TF-IDF document representation

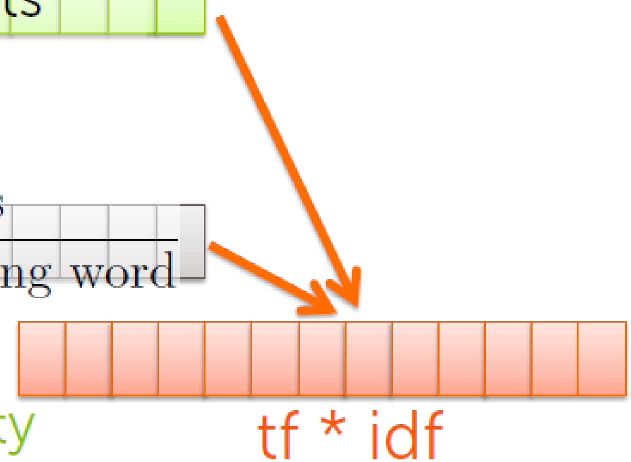
Emphasizes important words

- Appears frequently in document (common locally)

  Term frequency = word counts

- Appears rarely in corpus (rare globally)

  Inverse doc freq. = $\log \frac{\# \text{ docs}}{1 + \# \text{ docs using word}}$



Trade off: local frequency vs. global rarity

Distance metrics:

Distance metrics:
Defining notion of "closest"

In 1D, just Euclidean distance:

$$\text{distance}(x_i, x_q) = |x_i - x_q|$$

In multiple dimensions:

- can define many interesting distance functions
- most straightforwardly, might want to weight different dimensions differently

Distance metrics:

Weighting different features

Reasons:

– Some features are more relevant than others

 # bedrooms
 # bathrooms
sq.ft. living
sq.ft. lot
floors
year built
year renovated
waterfront

Distance metrics:

Weighting different features

Reasons:

– Some features are more relevant than others

title
abstract
main body
conclusion

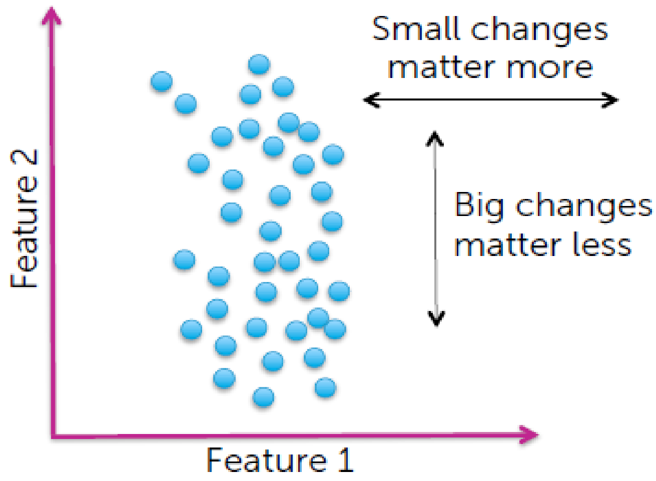
Distance metrics:

Weighting different features

Reasons:

– Some features are more relevant than others

– Some features vary more than others



Specify weights as a function of feature spread

For feature j:

$$\frac{1}{\max_i(\mathbf{x}_i[j]) - \min_i(\mathbf{x}_i[j])}$$

Distance metrics:

Scaled Euclidean distance

Formally, this is achieved via

$$\text{distance}(\mathbf{x}_i, \mathbf{x}_q) = \sqrt{a_1(\mathbf{x}_i[1]-\mathbf{x}_q[1])^2 + \ldots + a_d(\mathbf{x}_i[d]-\mathbf{x}_q[d])^2}$$

weight on each feature
(defining relative importance)

Distance metrics:

Effect of binary weights

distance($\mathbf{x}_i$, $\mathbf{x}_q$) =

$$\sqrt{a_1(\mathbf{x}_i[1]-\mathbf{x}_q[1])^2 + \ldots + a_d(\mathbf{x}_i[d]-\mathbf{x}_q[d])^2}$$

Setting weights as 0 or 1
is equivalent to
feature selection

Feature engineering/
selection is
important, but **hard**

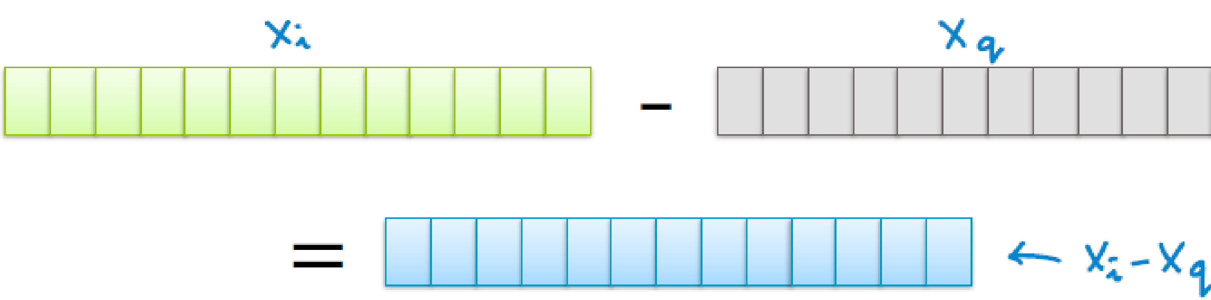
Distance metrics:

(non-scaled) Euclidean distance

Defined in terms of inner product

$$\text{distance}(\mathbf{x}_i, \mathbf{x}_q) = \sqrt{(\mathbf{x}_i - \mathbf{x}_q)^T(\mathbf{x}_i - \mathbf{x}_q)}$$

$$\sqrt{(\mathbf{x}_i[1]-\mathbf{x}_q[1])^2 + \dots + (\mathbf{x}_i[d]-\mathbf{x}_q[d])^2}$$



Distance metrics:

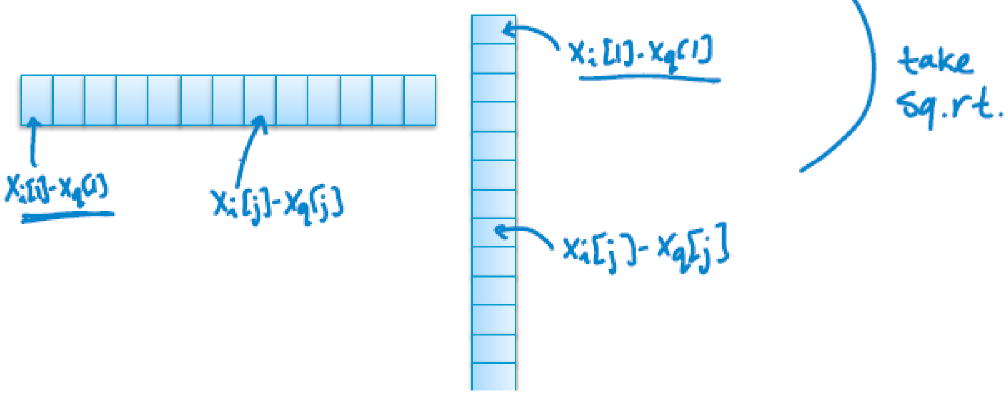
(non-scaled) Euclidean distance

Defined in terms of inner product

$$\text{distance}(\mathbf{x}_i, \mathbf{x}_q) = \sqrt{(\mathbf{x}_i - \mathbf{x}_q)^T(\mathbf{x}_i - \mathbf{x}_q)}$$

$$\sqrt{(\mathbf{x}_i[1]-\mathbf{x}_q[1])^2 + \ldots + (\mathbf{x}_i[d]-\mathbf{x}_q[d])^2}$$

distance^2 =



Distance metrics:

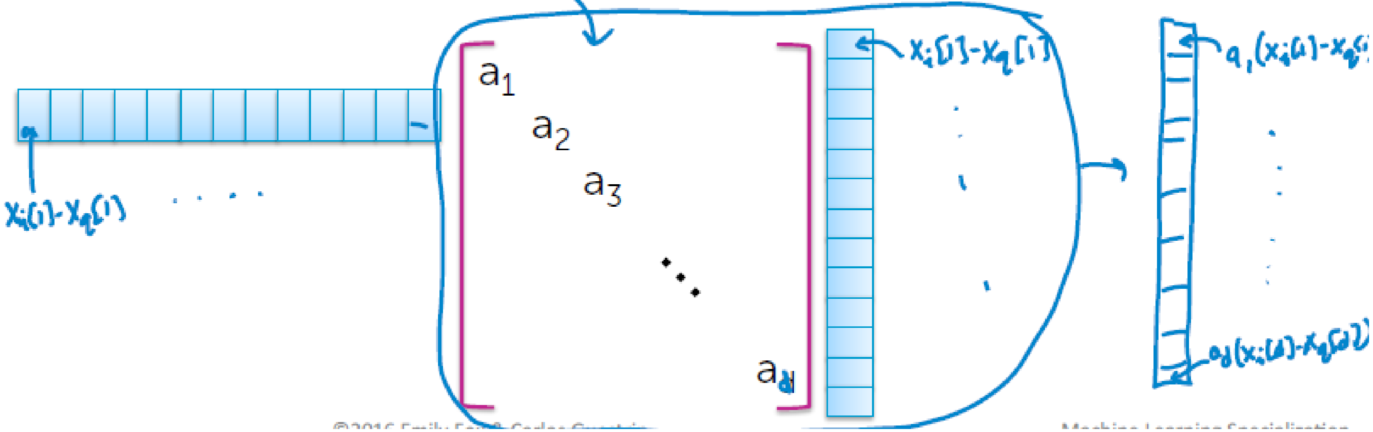
Scaled Euclidean distance

Defined in terms of inner product

$$\text{distance}(\mathbf{x}_i, \mathbf{x}_q) = \sqrt{(\mathbf{x}_i - \mathbf{x}_q)^T \mathbf{A} (\mathbf{x}_i - \mathbf{x}_q)}$$

$$\sqrt{a_1(\mathbf{x}_i[1]-\mathbf{x}_q[1])^2 + \ldots + a_d(\mathbf{x}_i[d]-\mathbf{x}_q[d])^2}$$

$\text{distance}^2 =$



Distance metrics:

Another natural inner product measure

x_q

1	0	0	0	5	3	0	0	1	0	0	0	0

Similarity

$$=\mathbf{x}_i^T \mathbf{x}_q$$

$$=\sum_{j=1}^{d} \mathbf{x}_i[j]\, \mathbf{x}_q[j]$$

$$= 13$$

3	0	0	0	2	0	0	1	0	1	0	0	0

x_i

Distance metrics:

Another natural inner product measure

1	0	0	0	5	3	0	0	1	0	0	0	0

Similarity → = 0

0	0	1	0	0	0	9	0	0	6	0	4	0

Distance metrics

Cosine similarity – normalize

$$\textbf{Similarity} = \frac{\sum_{j=1}^{d} \mathbf{x}_i[j]\,\mathbf{x}_q[j]}{\sqrt{\sum_{j=1}^{d} (\mathbf{x}_i[j])^2}\sqrt{\sum_{j=1}^{d} (\mathbf{x}_q[j])^2}}$$

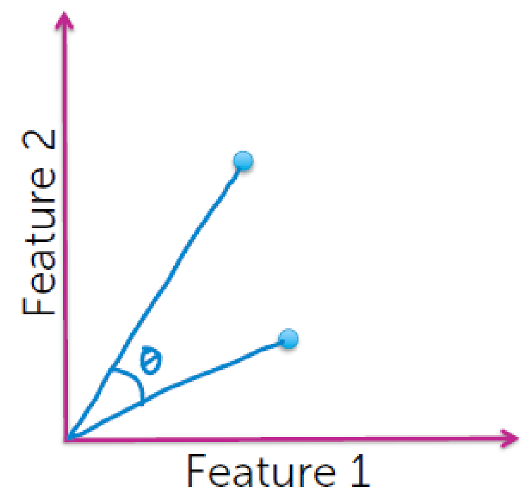
$a^T b = \|a\|\,\|b\| \cos(\theta)$

$$\frac{\mathbf{x}_i^T \mathbf{x}_q}{\|\mathbf{x}_i\|\,\|\mathbf{x}_q\|} = \cos(\theta)$$

$$= \left(\frac{x_i}{\|x_i\|}\right)^T \left(\frac{x_q}{\|x_q\|}\right)$$

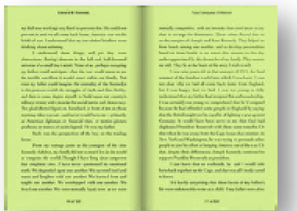
first normalize

- Not a proper distance metric
- Efficient to compute for sparse vecs



Distance metrics

Normalize



1	0	0	0	5	3	0	0	1	0	0	0	0

← x_i

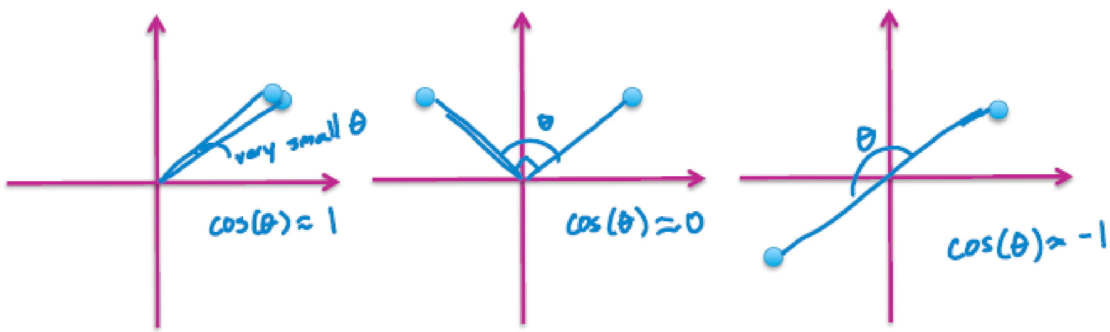
$$\sqrt{(1^2 + 5^2 + 3^2 + 1^2)}$$

← $\|x_i\| = \sum_{j=1}^{d} x_i[j]^2$

1/6	0	0	0	5/6	3/6	0	0	1/6	0	0	0	0

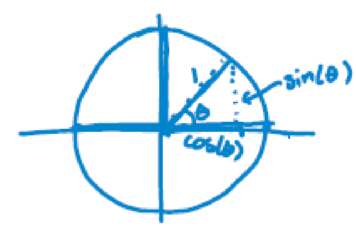
Distance metrics

Cosine similarity



In general, -1 < similarity < 1

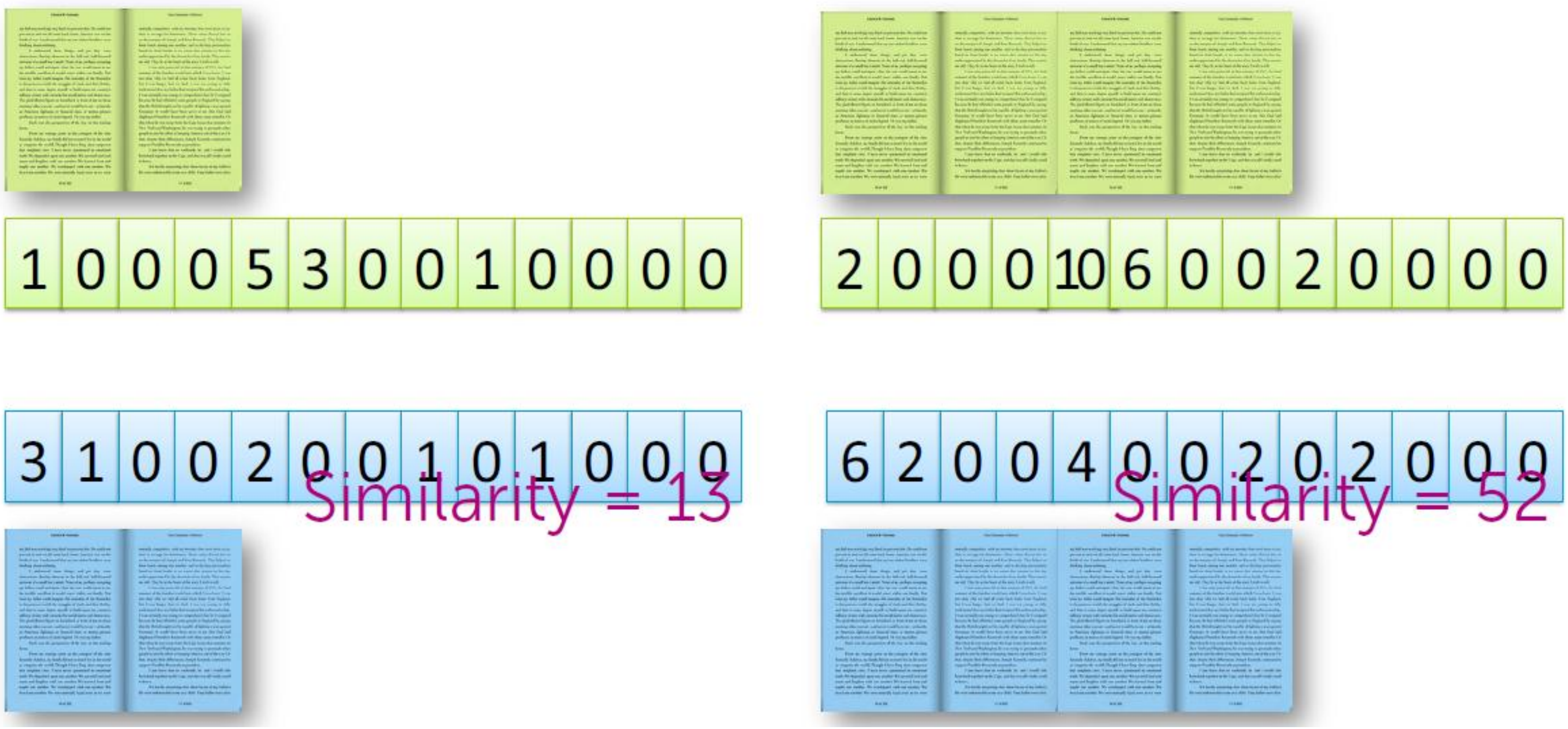
For positive features (like tf-idf)
0 < similarity < 1 } our focus



Define **distance = 1-similarity**

Distance metrics

To normalize or not?



Distance metrics

In the normalized case

1/6	0	0	0	5/6	3/6	0	0	1/6	0	0	0	0
3/4	1/4	0	0	1/2	0	0	1/4	0	1/4	0	0	0

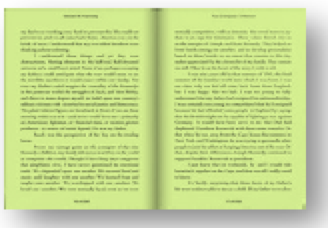
Similarity = 13/24

1/6	0	0	0	5/6	3/6	0	0	1/6	0	0	0	0
3/4	1/4	0	0	1/2	0	0	1/4	0	1/4	0	0	0

Similarity = 13/24

Distance metrics

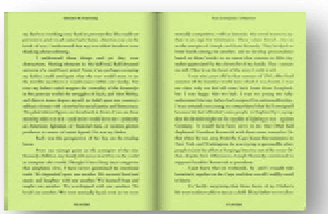
But not always desired...



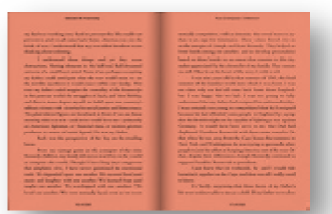
long document

short tweet

Normalizing can make dissimilar objects appear more similar



long document



long document

Common compromise:
Just cap maximum word counts

Distance metrics

Other distance metrics

- Mahalanobis
- rank-based
- correlation-based
- Manhattan
- Jaccard
- Hamming
- ...

Combining distance metrics

Example of document features:

1. Text of document
   – Distance metric: Cosine similarity
2. # of reads of doc
   – Distance metric: Euclidean distance

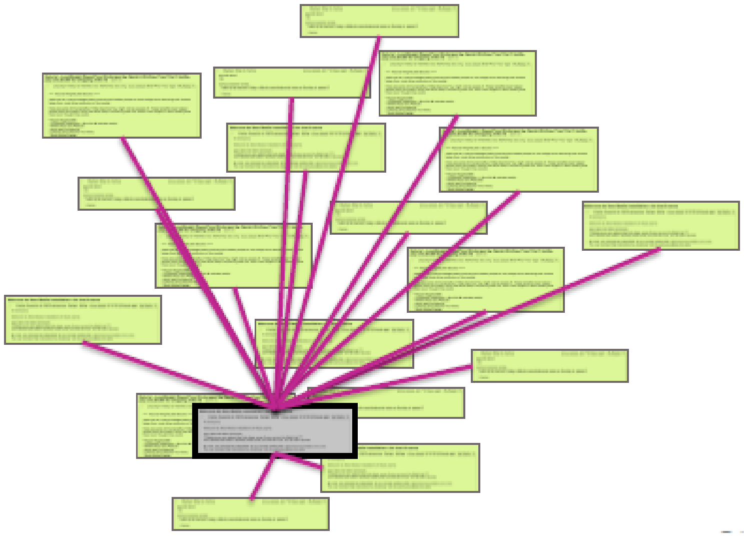
Add together with user-specified weights

Scaling up k-NN search by storing data in a KD-tree

Complexity of brute-force search

Given a query point, scan through each point

- O(*N*) distance computations per 1-NN query!
- O(*N*log*k*) per *k*-NN query!

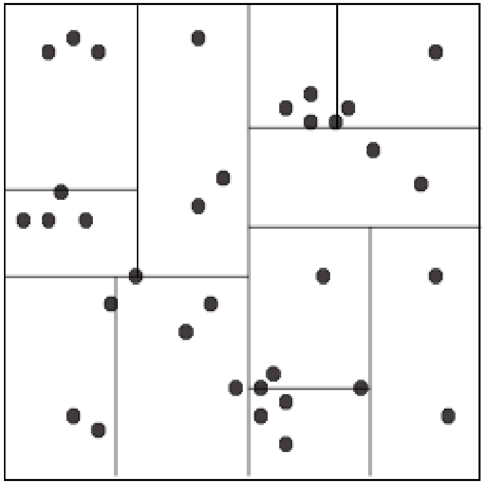


What if *N* is huge???
(and many queries)

KD-trees

Structured organization of documents

- Recursively partitions points into axis aligned boxes.



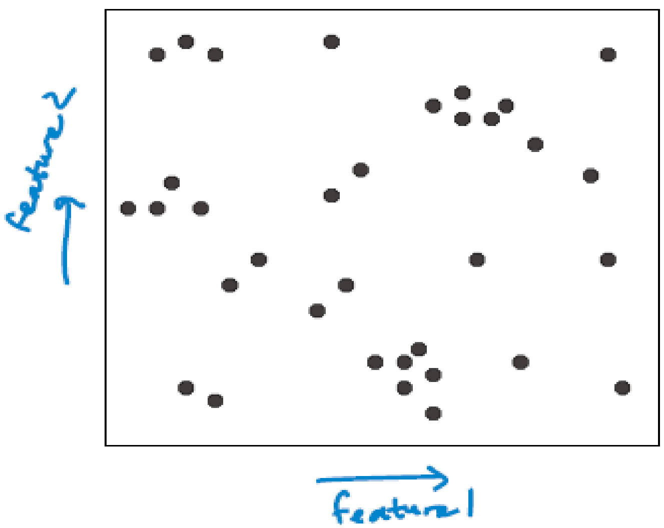
Enables more efficient pruning of search space

Works "well" in "low-medium" dimensions

- We'll get back to this...

KD-trees

KD-tree construction



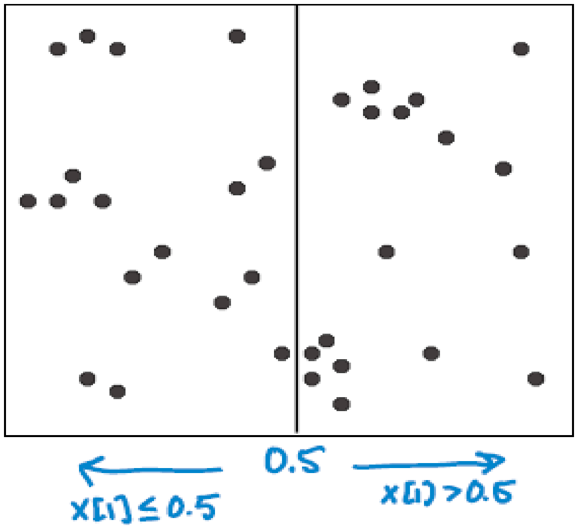
Start with a list of d-dimensional points.

Pt	**x**[1]	**x**[2]
1	0.00	0.00
2	1.00	4.31
3	0.13	2.85
...	...	...

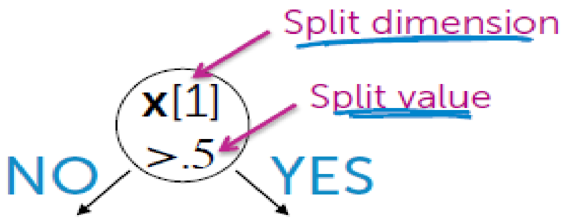
obs. indices | feat. 1 (word 1) | feat. 2 (word 2)

KD-trees

KD-tree construction



Split points into 2 groups



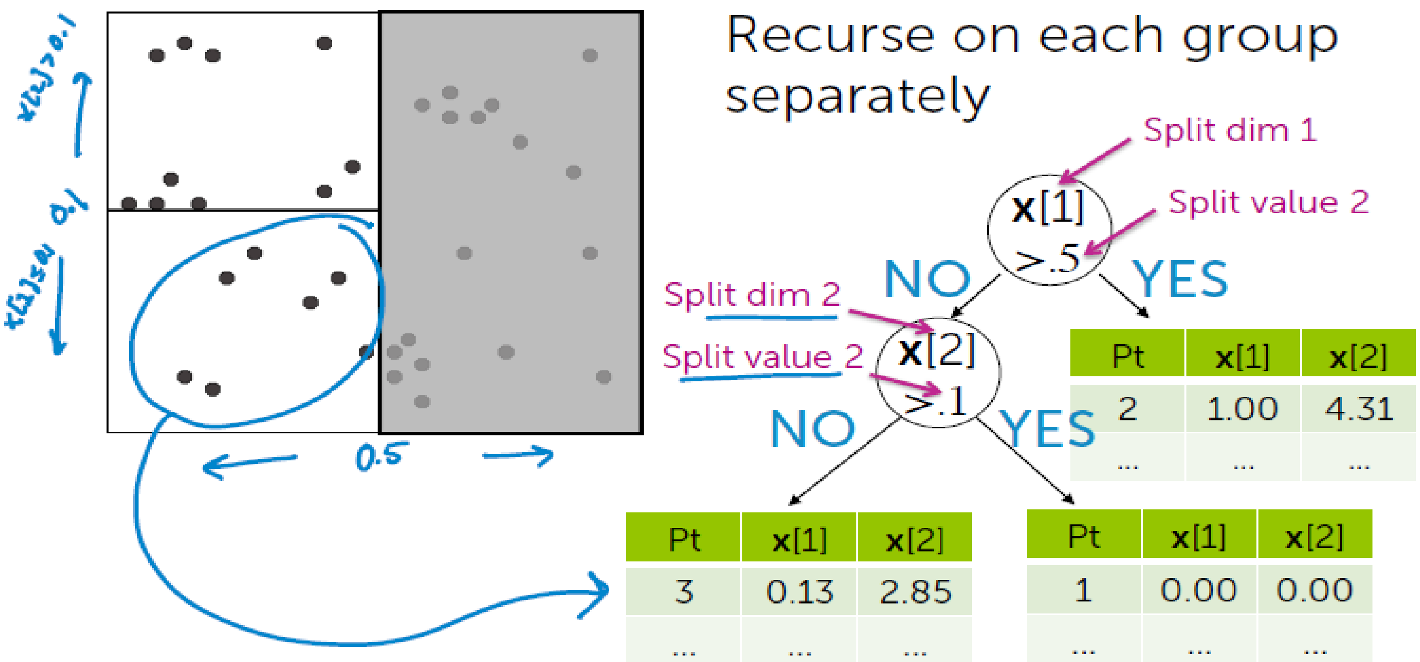
Pt	x[1]	x[2]
1	0.00	0.00
3	0.13	2.85
...	...	...

Pt	x[1]	x[2]
2	1.00	4.31
...	...	...

KD-trees

KD-tree construction

Recurse on each group separately



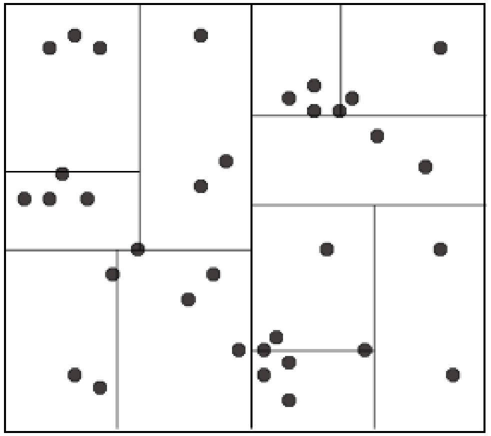
Pt	x[1]	x[2]
2	1.00	4.31
…	…	…

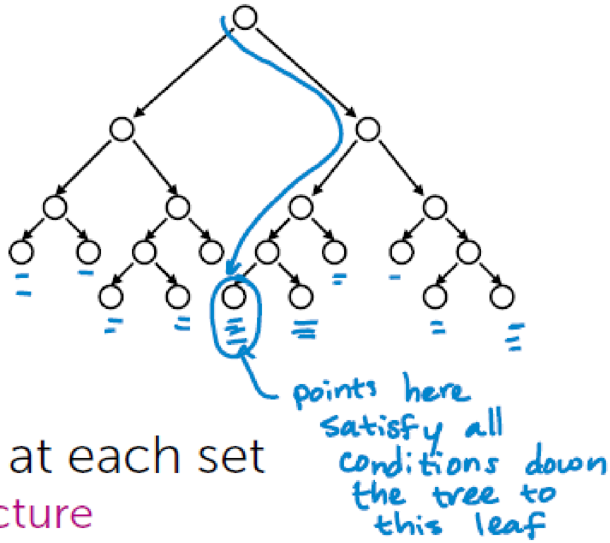
Pt	x[1]	x[2]
3	0.13	2.85
…	…	…

Pt	x[1]	x[2]
1	0.00	0.00
…	…	…

KD-trees

KD-tree construction





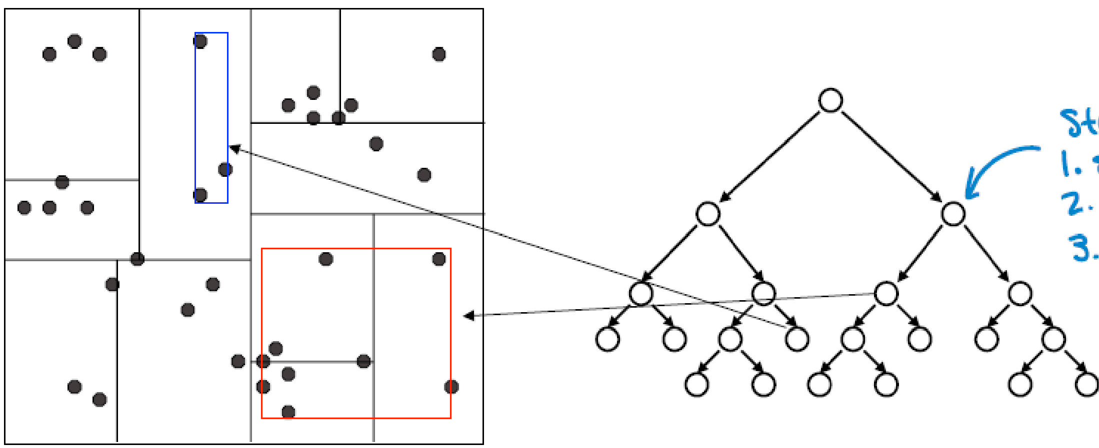
Continue splitting points at each set

- Creates a binary tree structure

Each leaf node contains a list of points

KD-trees

KD-tree construction



Store:
1. split dim?
2. split value
3. bounding box that is as small as possible while containing pts

Keep one additional piece of info at each node:

#3 – The (tight) bounds of points at or below node

KD-trees

KD-tree construction choices

Use heuristics to make splitting decisions:

- Which dimension do we split along?

  widest (or alternate)

- Which value do we split at?

  median (or center point of box, ignoring data in box)

- When do we stop?

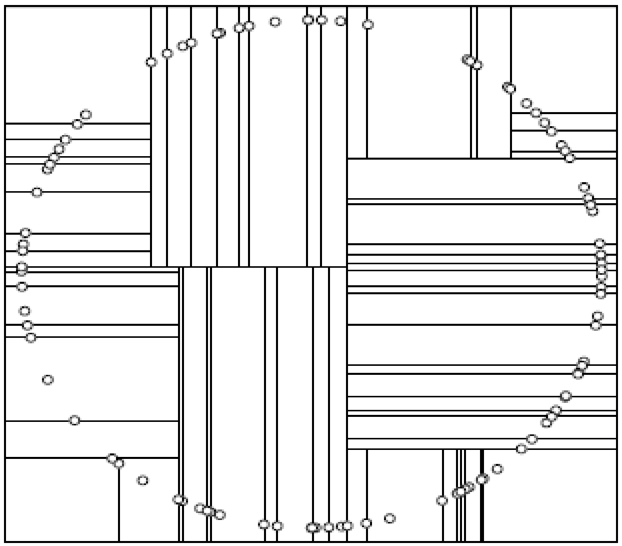
  fewer than m pts left

  <u>or</u>

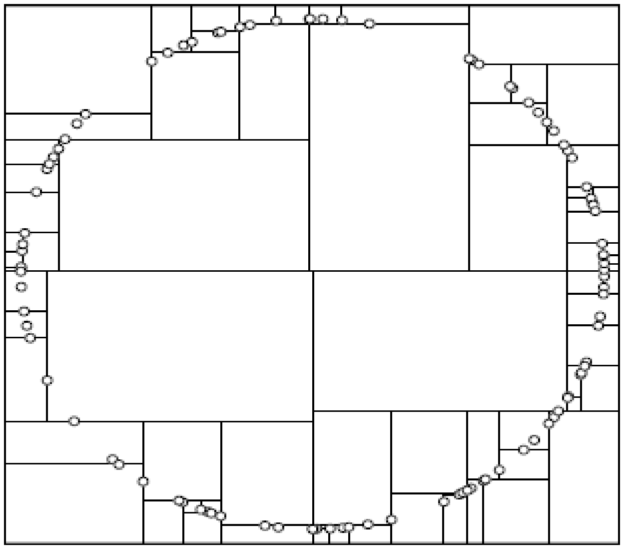
  box hits minimum width

KD-trees

Many heuristics…

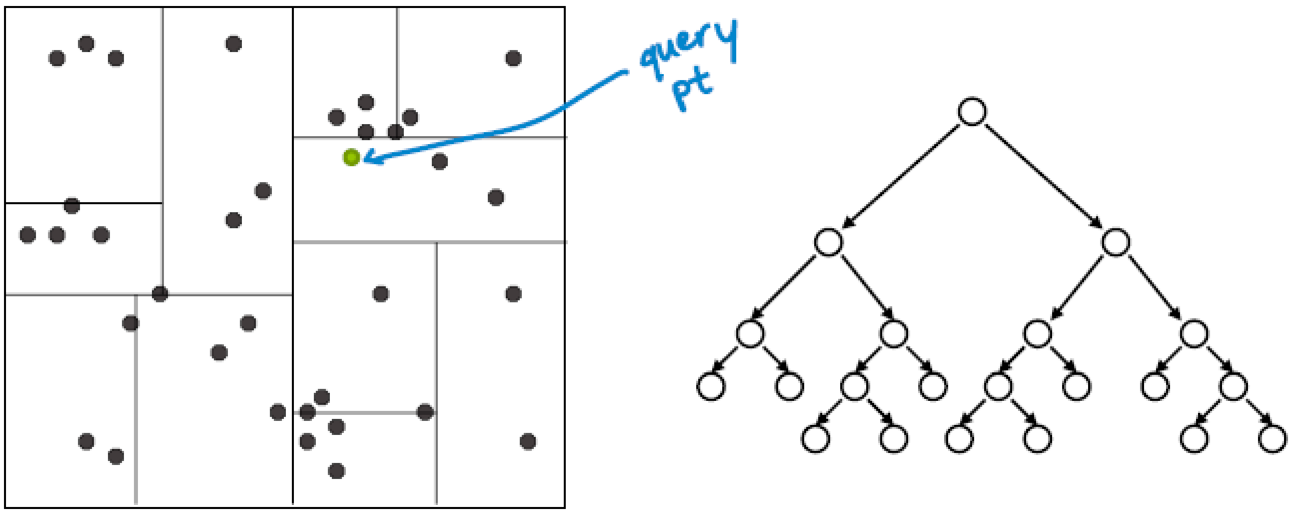


median heuristic



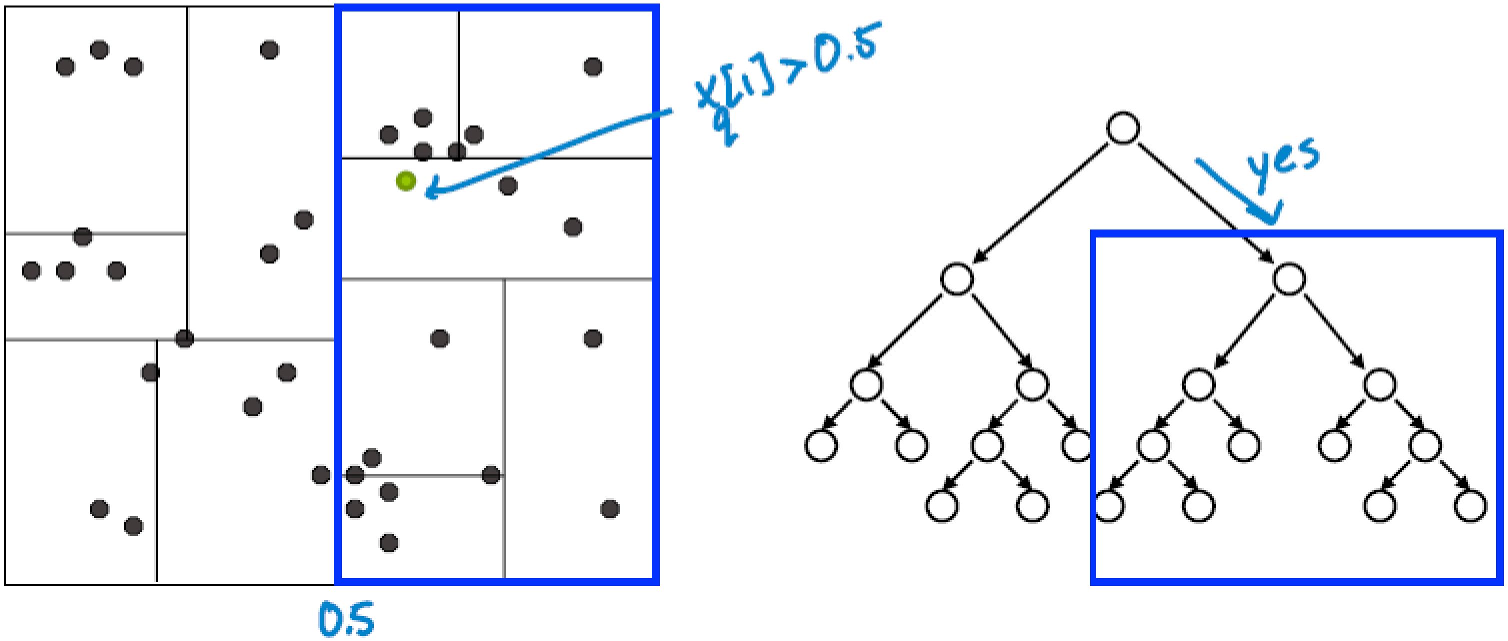
center-of-range heuristic

Nearest neighbor with KD-trees



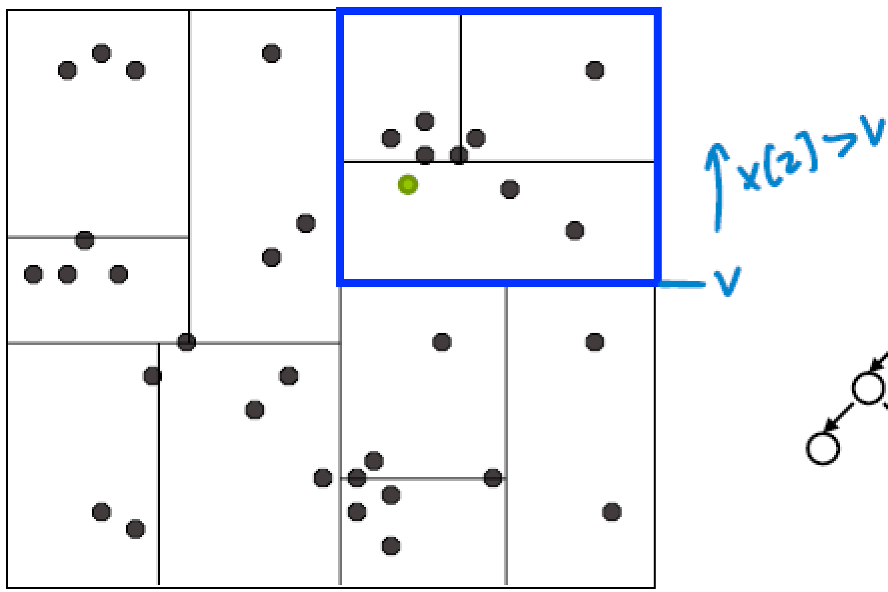
Traverse tree looking for nearest neighbor to query point

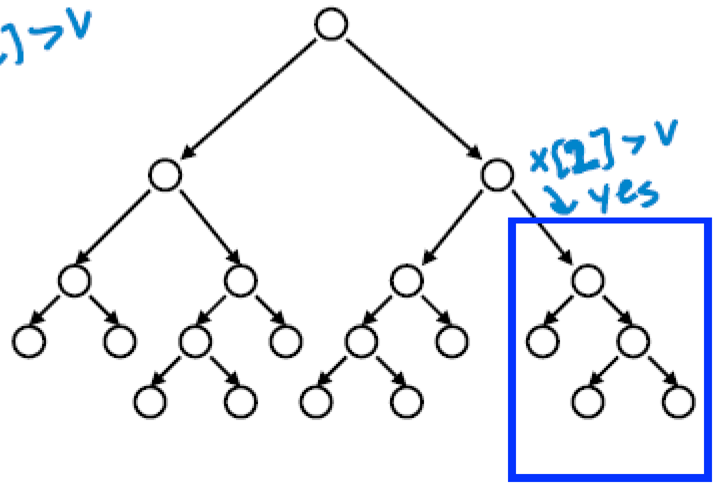
Nearest neighbor with KD-trees



1. Start by exploring leaf node containing query point

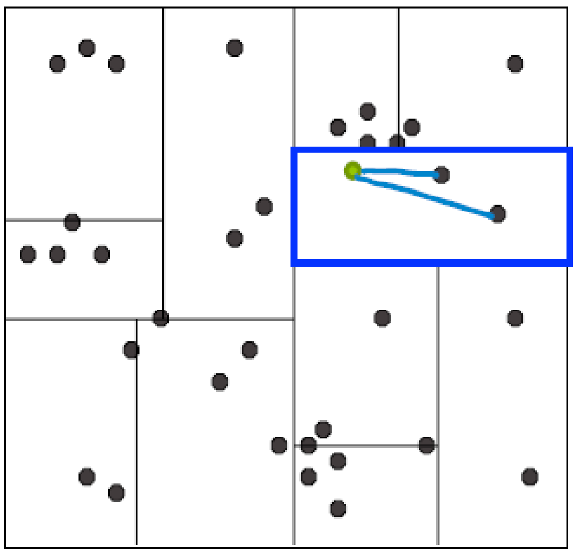
Nearest neighbor with KD-trees

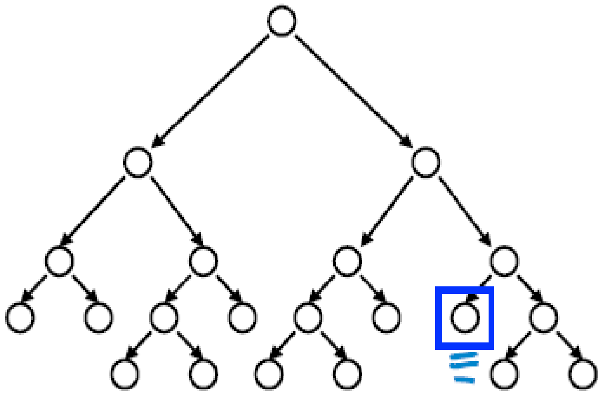




1. Start by exploring leaf node containing query point

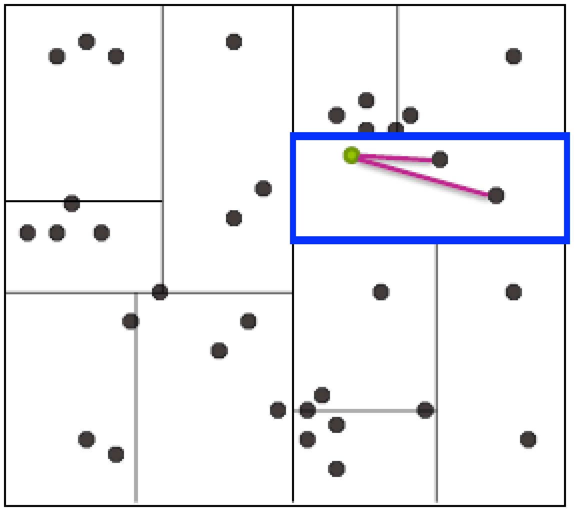
Nearest neighbor with KD-trees

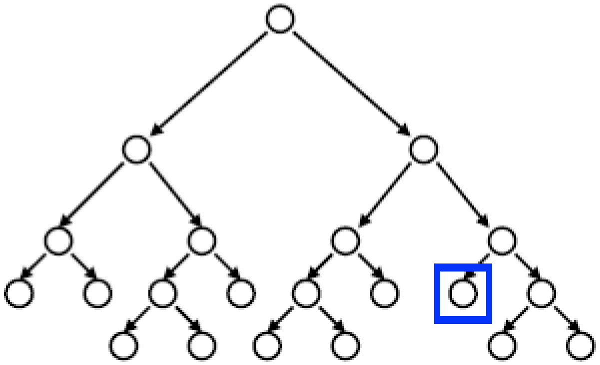




1. Start by exploring leaf node containing query point

Nearest neighbor with KD-trees

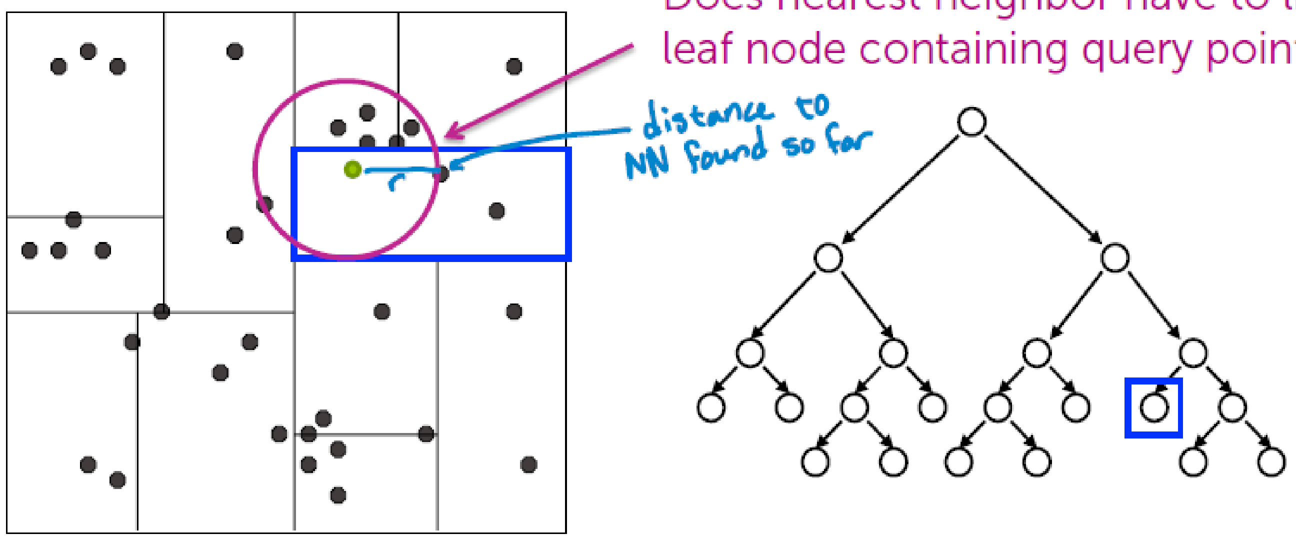




1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node

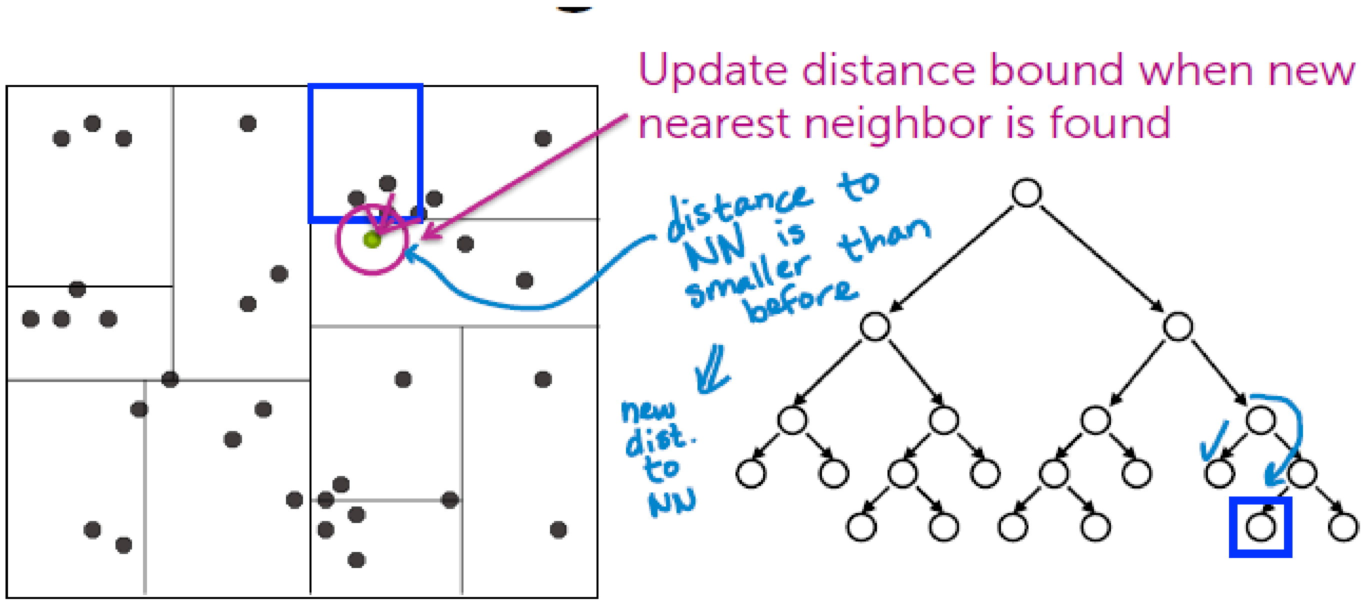
Nearest neighbor with KD-trees

Does nearest neighbor have to live at leaf node containing query point?



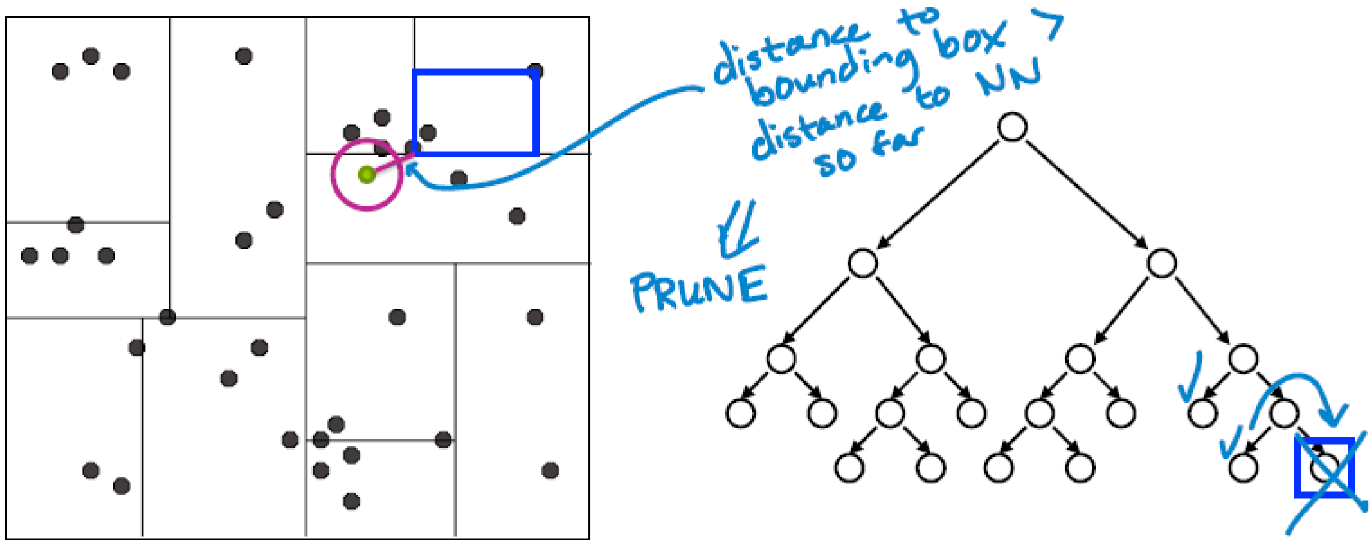
1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node

Nearest neighbor with KD-trees



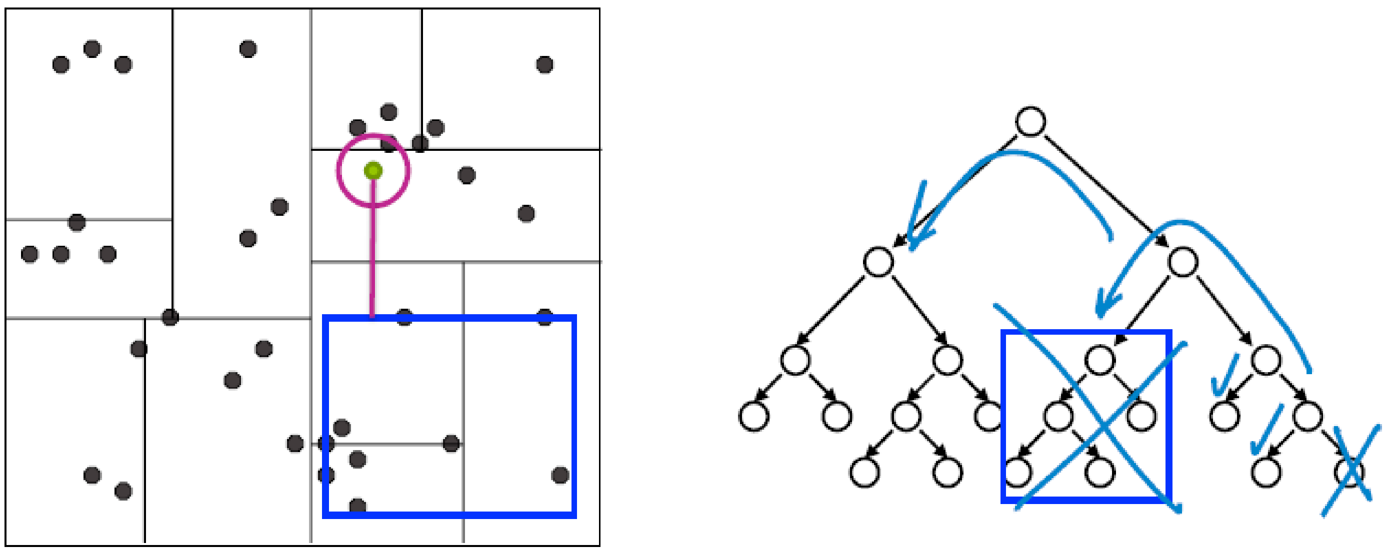
1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node
3. Backtrack and try other branch at each node visited

Nearest neighbor with KD-trees



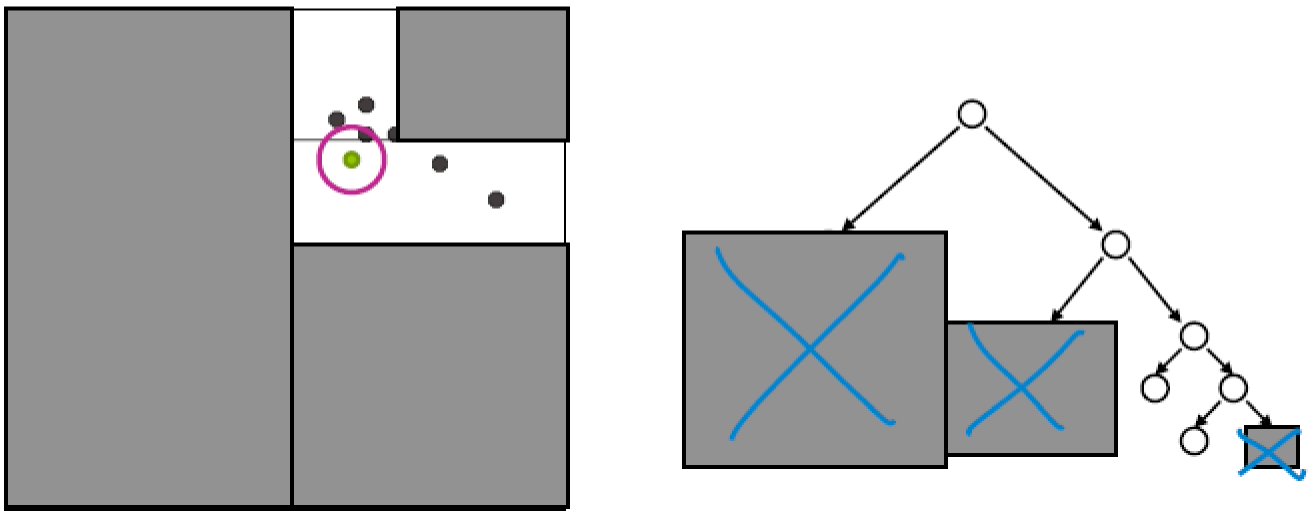
Use distance bound and bounding box of each node to prune parts of tree that cannot include nearest neighbor

Nearest neighbor with KD-trees



Use distance bound and bounding box of each node to prune parts of tree that cannot include nearest neighbor

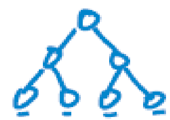
Nearest neighbor with KD-trees



Use distance bound and bounding box of each node to prune parts of tree that cannot include nearest neighbor

Nearest neighbor with KD-trees

Complexity



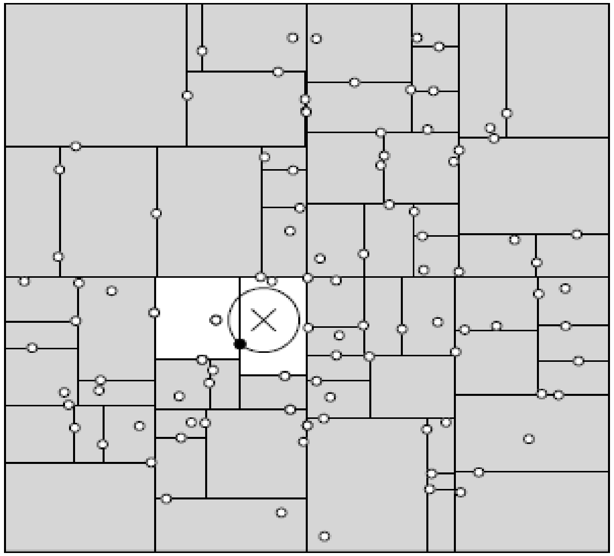
For (nearly) balanced, binary trees...

- Construction
  - Size: 2N−1 nodes if 1 datapt at each leaf → O(N)
  - Depth: O(log N)
  - Median + send points left right: O(N) at every level of the tree
  - Construction time: O(N log N)
- 1-NN query
  - Traverse down tree to starting point: O(log N)
  - Maximum backtrack and traverse: O(N) in worst case
  - Complexity range: O(log N) → O(N)

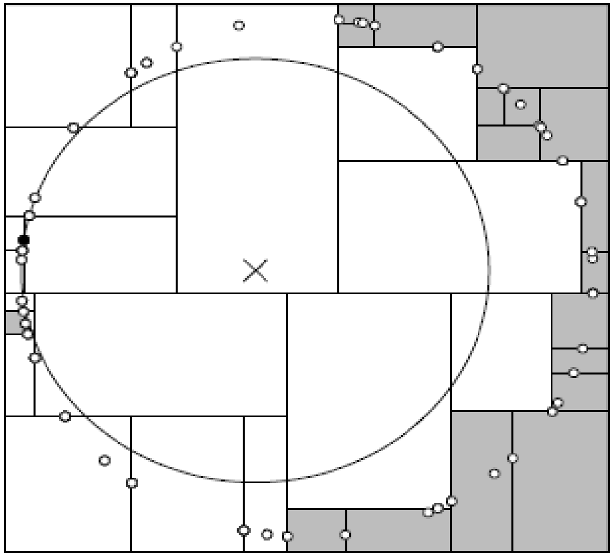
Under some assumptions on distribution of points, we get O(logN) but exponential in d

Nearest neighbor with KD-trees

Complexity



pruned many
(closer to O(log N))



pruned few
(closer to O(N))

Complexity for N queries

- Ask for nearest neighbor to each doc

  N queries

- Brute force 1-NN:

  $O(N^2)$

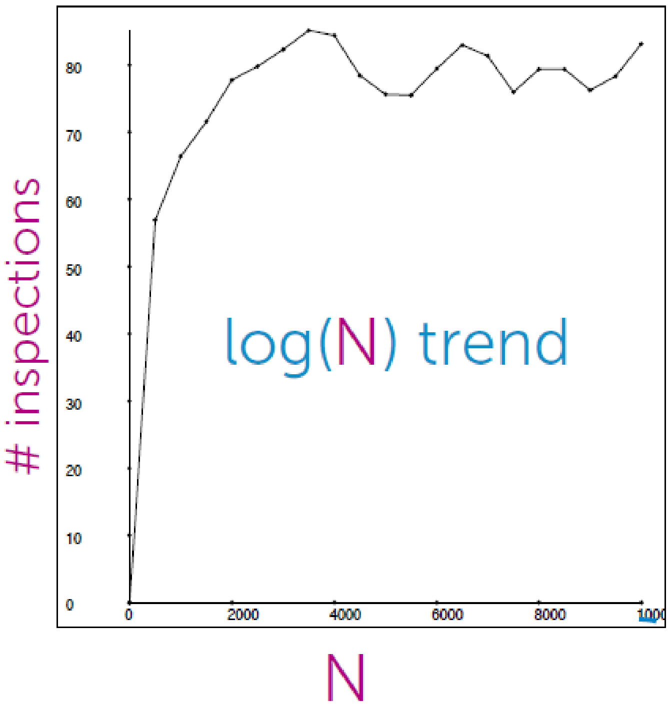
- kd-trees:

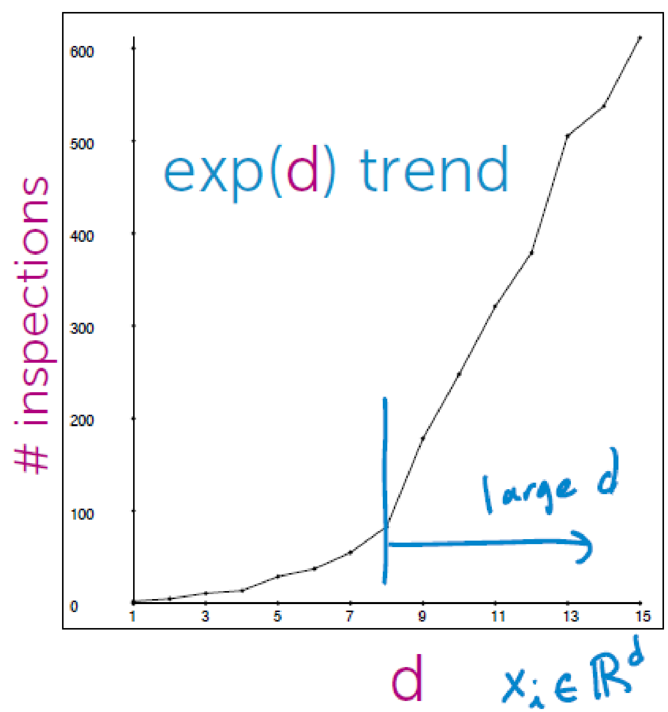
  $O(N \log N) \longrightarrow O(N^2)$

  ↑ potentially very large savings for large N!

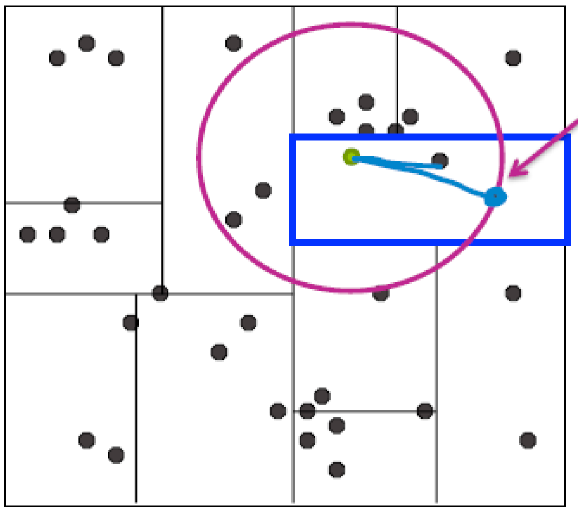
Complexity for N queries

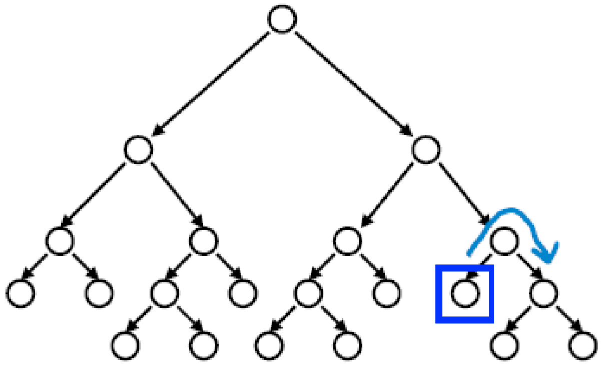
Inspections vs. N and d





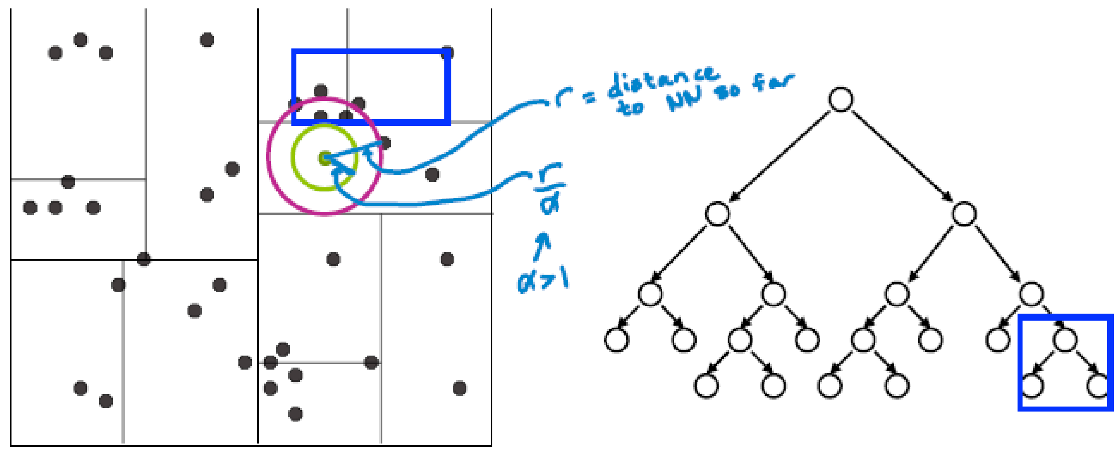
k-NN with KD-trees

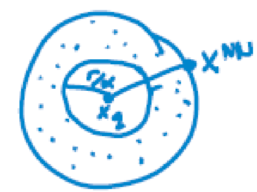




Exactly same algorithm, but maintain distance to
furthest of current *k* nearest neighbors

Approximate k-NN with KD-trees





Before: Prune when distance to bounding box > r

Now: Prune when distance to bounding box > r/ α

Prunes more than allowed, but can guarantee that if we return a neighbor at distance r, then there is no neighbor closer than r/ α

Bound loose…In practice, often closer to optimal.

Saves lots of search time at little cost in quality of NN!

Closing remarks on KD-trees

Tons of variants of kd-trees

- On construction of trees
  (heuristics for splitting, stopping, representing branches...)
- Other representational data structures for fast NN search
  (e.g., ball trees,...)

Nearest Neighbor Search

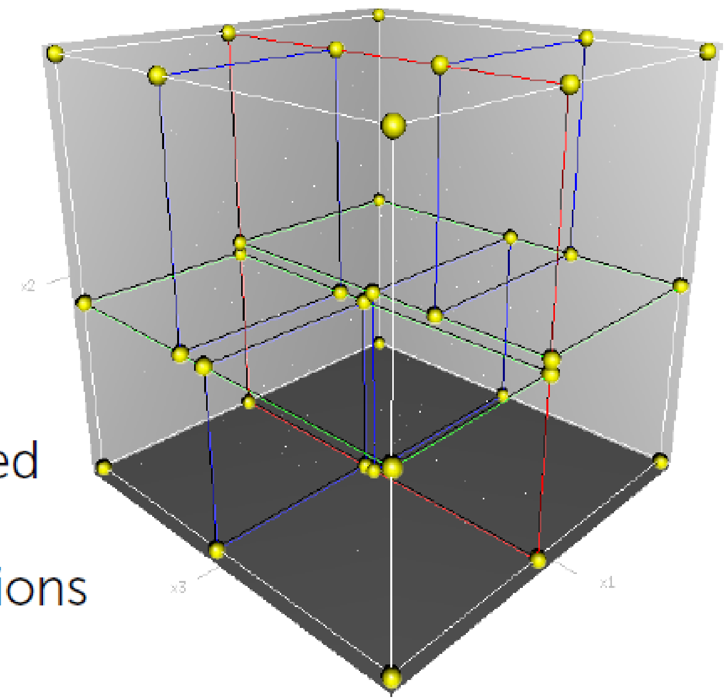
- Distance metric and data representation crucial to answer returned

For both, high-dim spaces are hard!

- Number of kd-tree searches can be exponential in dimension
  - Rule of thumb... $N >> 2^d$... Typically useless for large d.
- Distances sensitive to irrelevant features
  - Most dimensions are just noise → everything is far away
  - Need technique to learn which features are important to given task

KD-tree in high dimmensions

- 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???

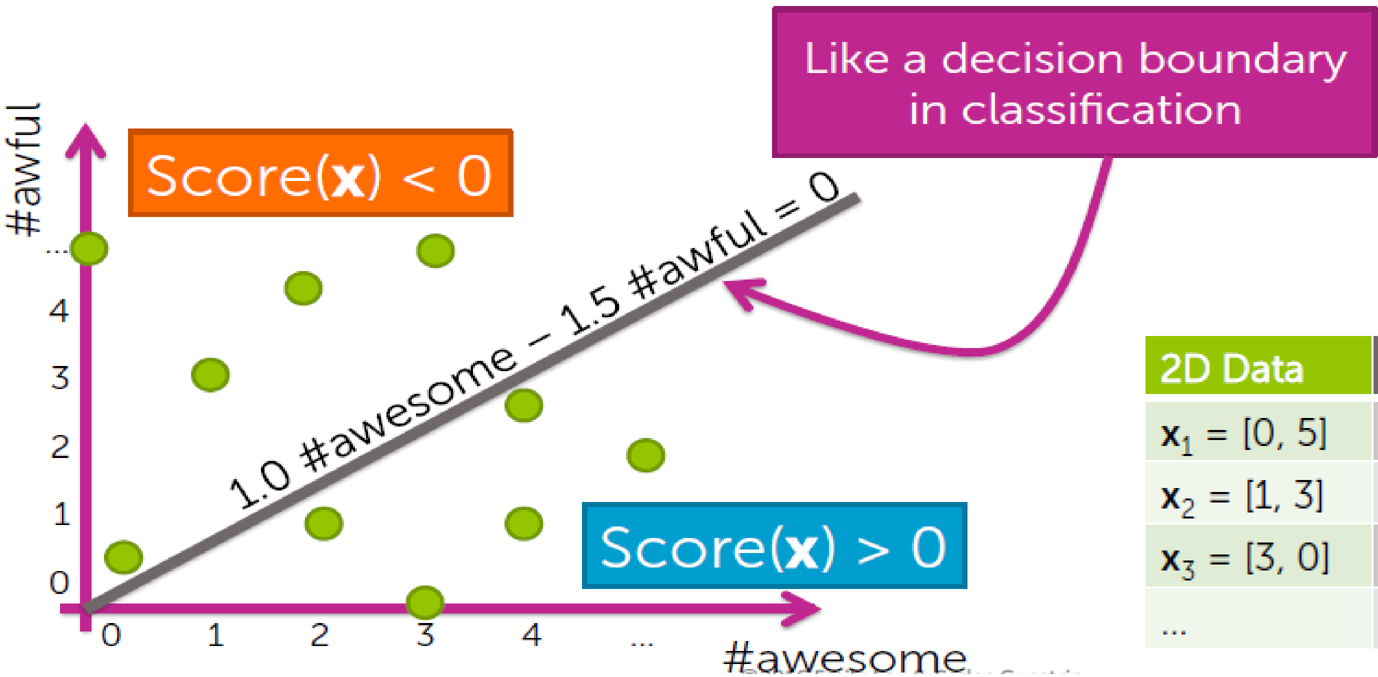
- Focus on methods that provide good probabilistic guarantees on approximation

Locality Sensitive Hashing (LHS) as alternative to KD-trees

Locality sensitive hashing

Simple "binning" of data into 2 bins

Score(**x**) = 1.0 #awesome – 1.5 #awful



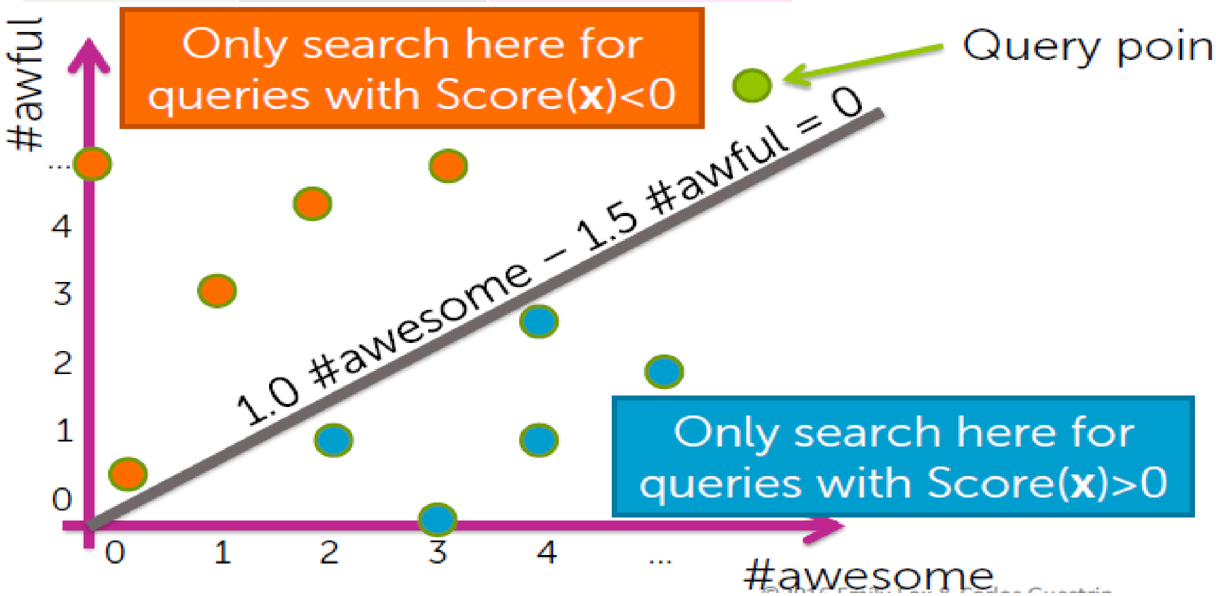
2D Data	Sign(Score)
$\mathbf{x}_1$ = [0, 5]	-1
$\mathbf{x}_2$ = [1, 3]	-1
$\mathbf{x}_3$ = [3, 0]	1
...	...

Locality sensitive hashing

Using bins for NN search

2D Data	Sign(Score)	Bin index
$\mathbf{x}_1 = [0, 5]$	-1	0
$\mathbf{x}_2 = [1, 3]$	-1	0
$\mathbf{x}_3 = [3, 0]$	1	1
…	…	…

candidate neighbors if Score($\mathbf{x}$)<0



Locality sensitive hashing

Using score for NN search

2D Data	Sign(Score)	Bin index
$\mathbf{x}_1$ = [0, 5]	-1	0
$\mathbf{x}_2$ = [1, 3]	-1	0
$\mathbf{x}_3$ = [3, 0]	1	1
…	…	…

candidate neighbors if Score(x)<0

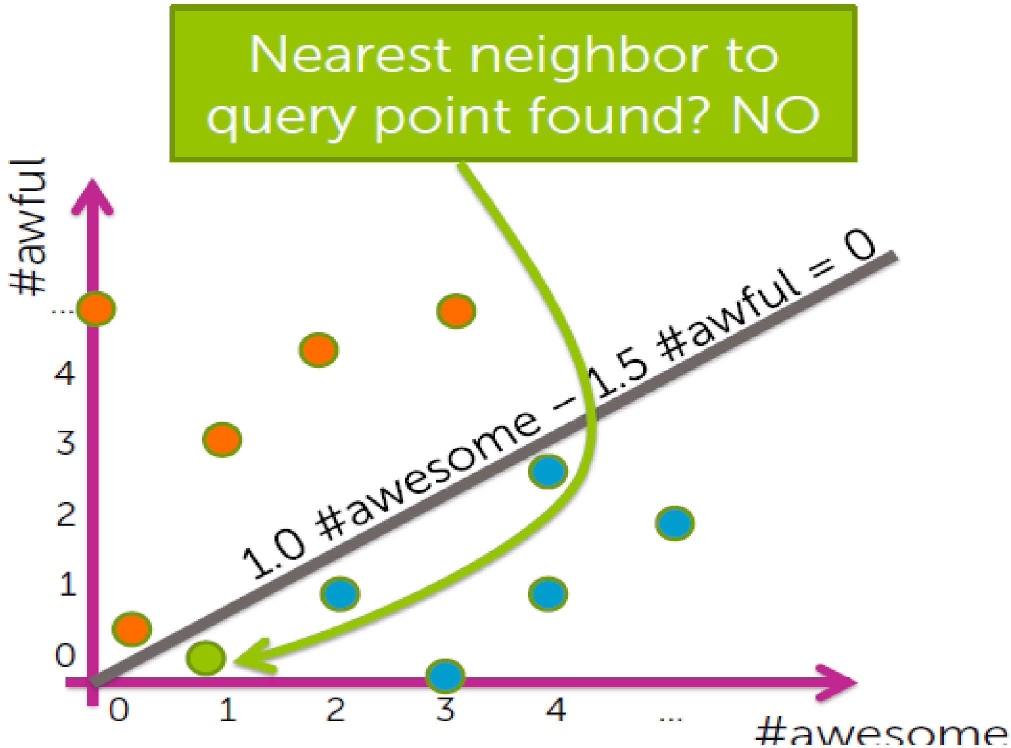
Bin	0	1
List containing indices of datapoints:	{1,2,4,7,…}	{3,5,6,8,…}

HASH TABLE

search for NN amongst this set

Locality sensitive hashing

Provides approximate NN



Locality sensitive hashing

Three potential issues with simple approach

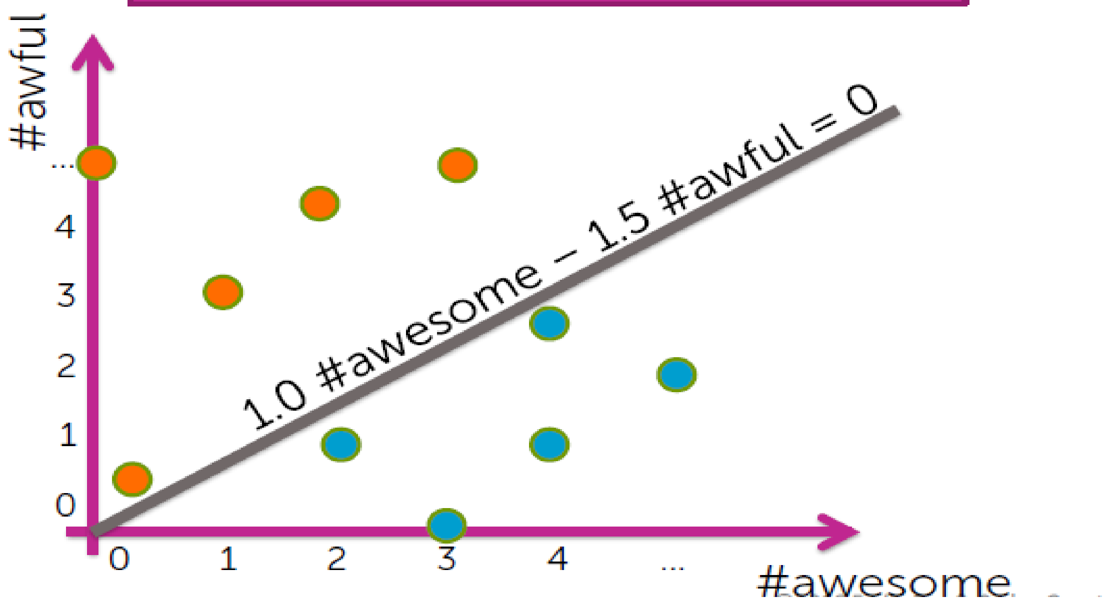
1. Challenging to find good line
2. Poor quality solution:
   - Points close together get split into separate bins
3. Large computational cost:
   - Bins might contain many points, so still searching over large set for each NN query

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

Locality sensitive hashing

How to define the line?

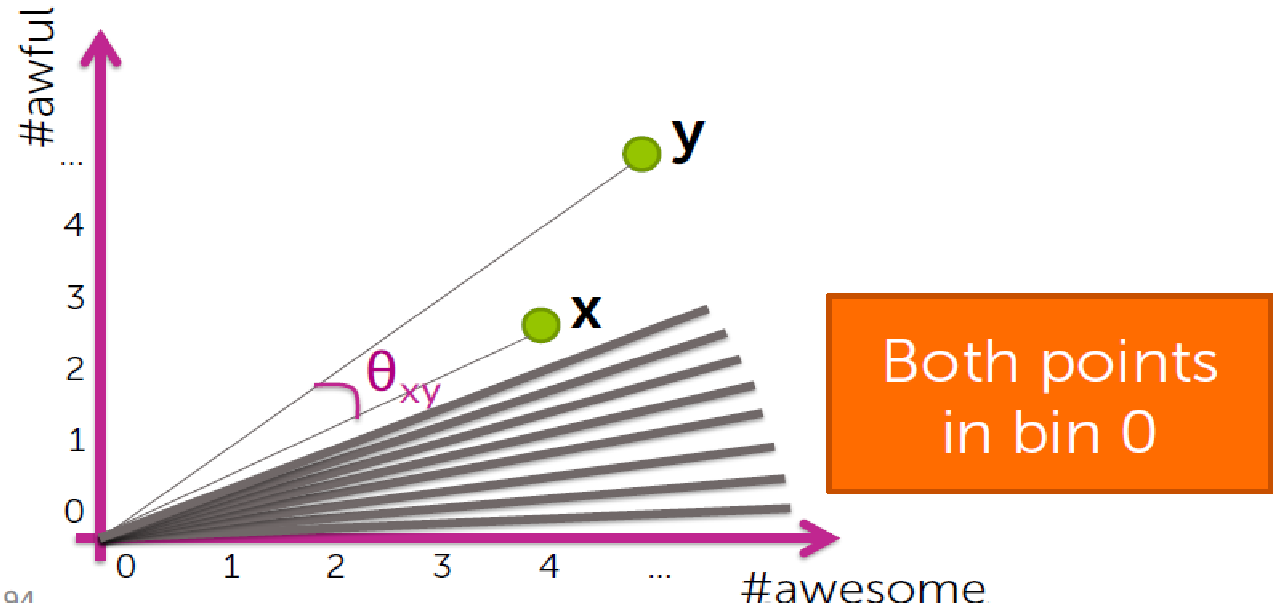
Crazy idea:
Define line randomly!



Locality sensitive hashing

How bad can a random line be?

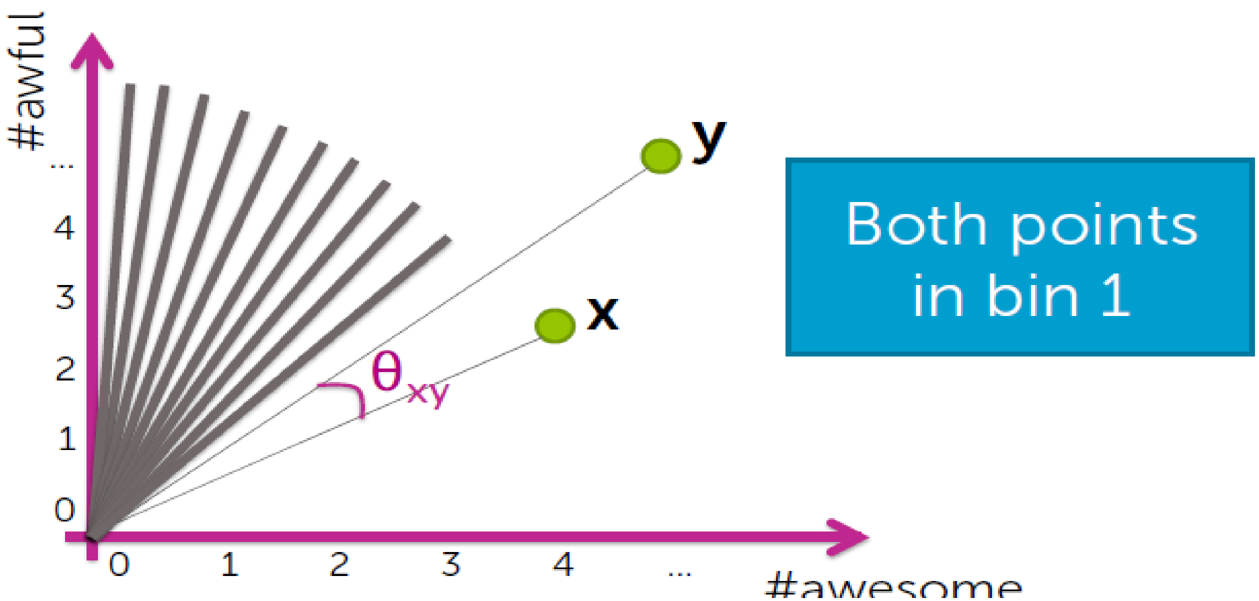
Goal: If **x**,**y** are close (according to cosine similarity), want binned values to be the same.



Locality sensitive hashing

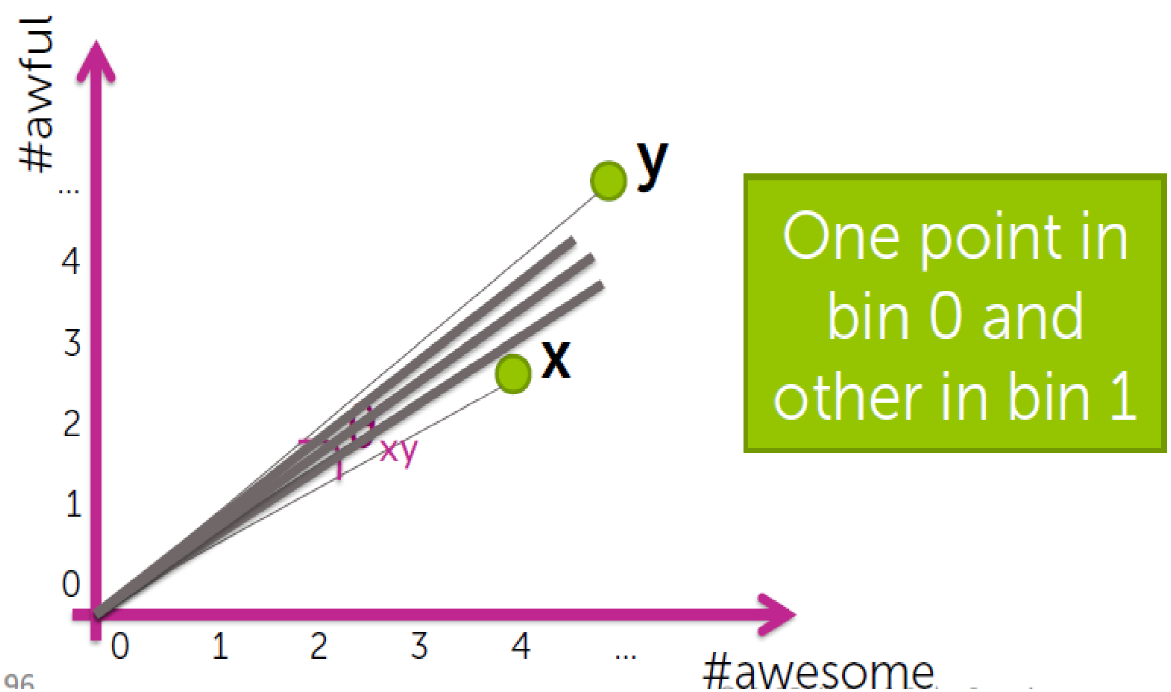
How bad can a random line be?

Goal: If **x**,**y** are close (according to cosine similarity), want binned values to be the same.



Locality sensitive hashing

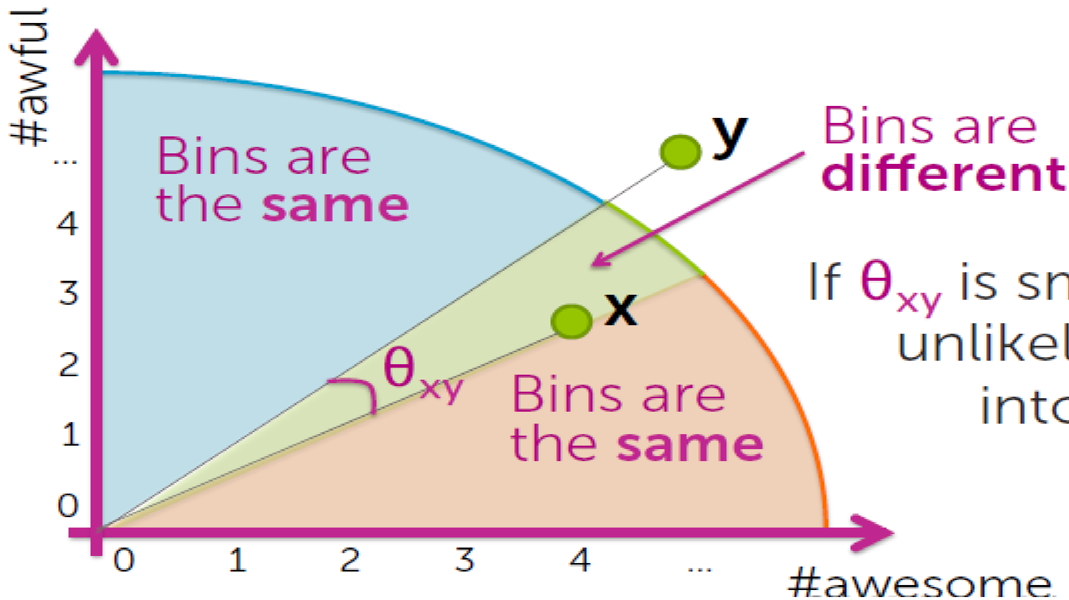
Goal: If **x**,**y** are close (according to cosine similarity), want binned values to be the same.



Locality sensitive hashing

How bad can a random line be?

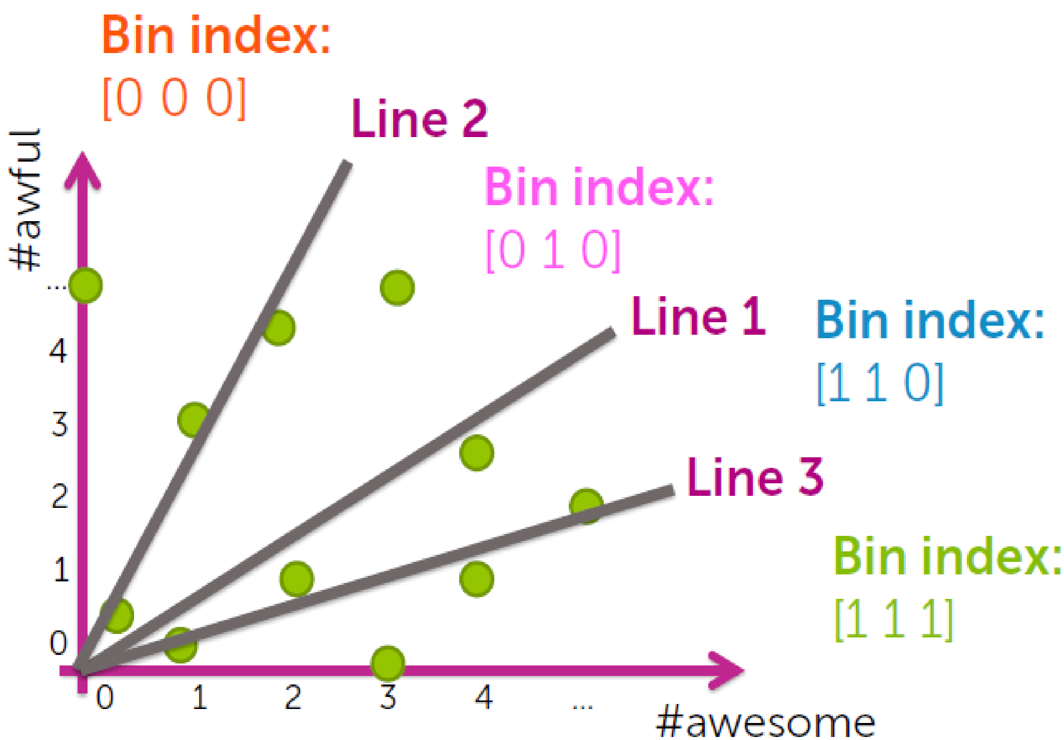
Goal: If **x**,**y** are close (according to cosine similarity), want binned values to be the same.



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

LSH: improving efficiency

Reducing search cost through more bins



LSH: improving efficiency

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
$\mathbf{x}_1$ = [0, 5]	-1	0	-1	0	-1	0
$\mathbf{x}_2$ = [1, 3]	-1	0	-1	0	-1	0
$\mathbf{x}_3$ = [3, 0]	1	1	1	1	1	1
…	…	…	…	…	…	…

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

search for NN amongst this set

LSH: improving efficiency

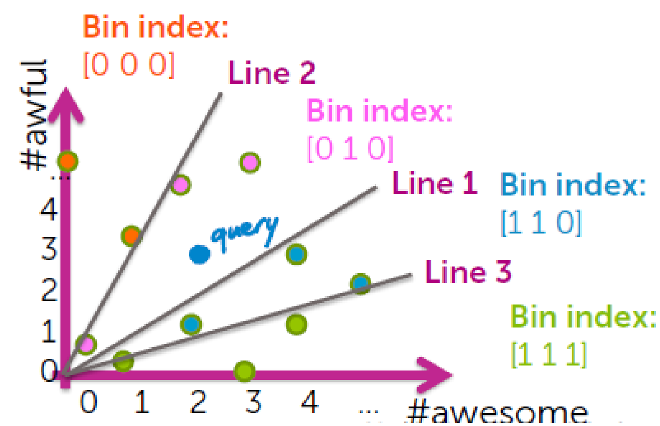
Improving search quality by searching neighboring bins

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

Query point here, but is NN?

Not necessarily

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

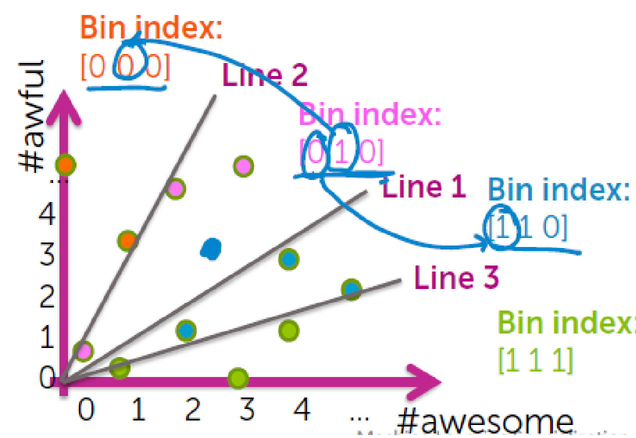


LSH: improving efficiency

Improving search quality by searching neighboring bins

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

Next closest bins
(flip 1 bit)

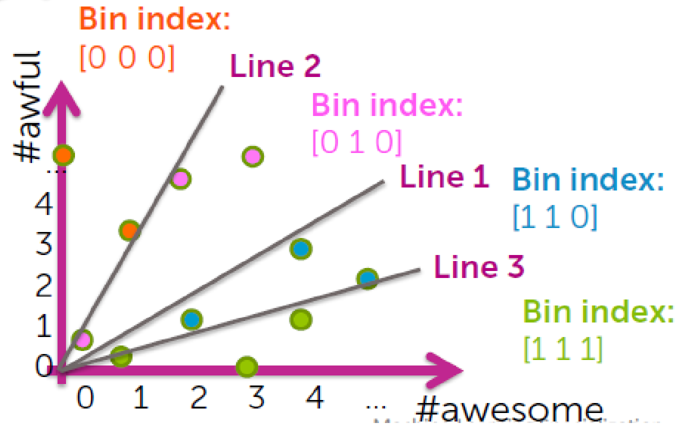


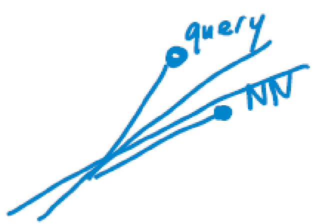
LSH: improving efficiency

Improving search quality by searching neighboring bins

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

Further bin (flip 2 bits)





LSH: improving efficiency

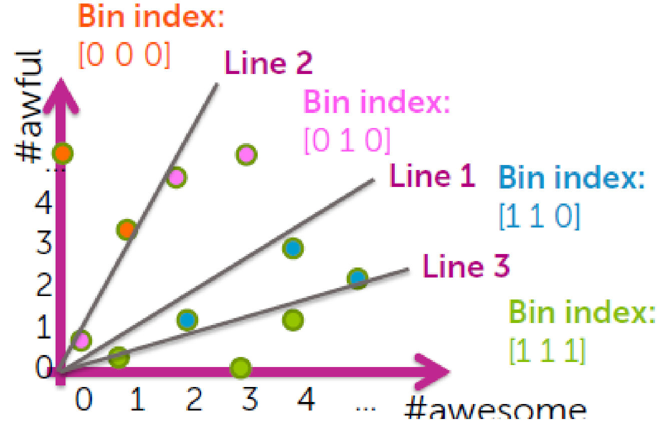
Improving search quality by searching neighboring bins

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

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



105

LSH recap

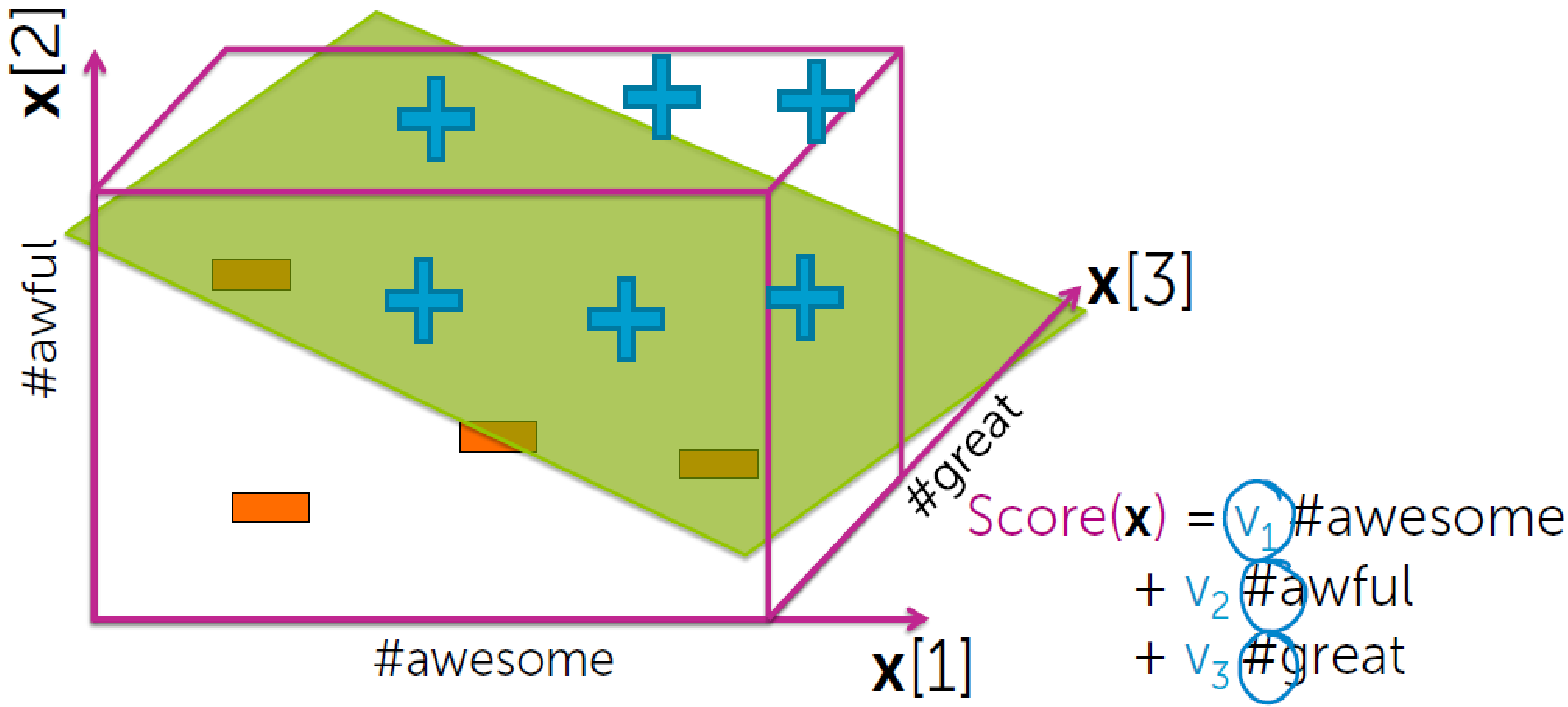
kd-tree competitor data structure

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

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

LSH: moving to higher dimmensions d

Draw random *planes*



Cost of binning points in d-dim

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

i[th] hyperplane

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

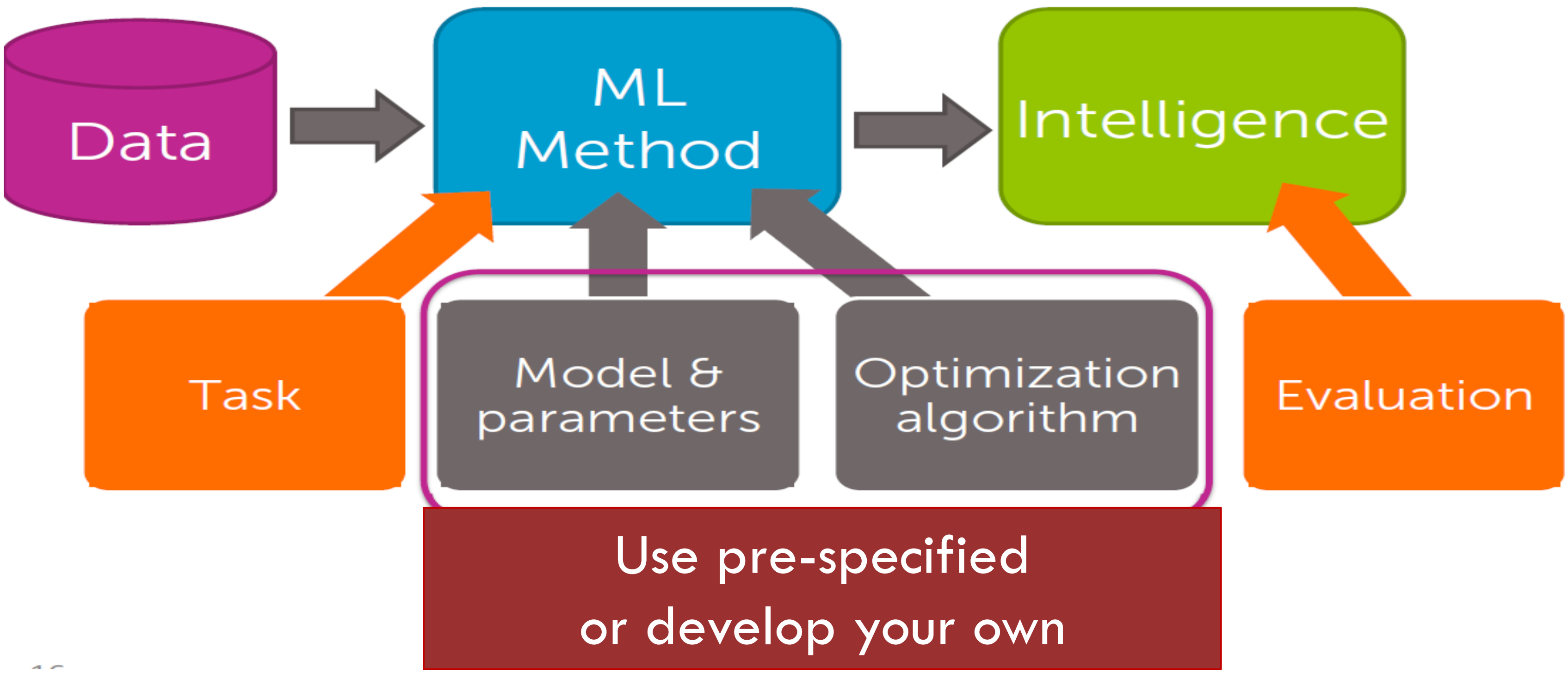
In high-dim, (and some applications) this is often a sparse mult.

One-time cost offset if many queries of fixed dataset

Wrapping up

Deploing inteligence module

Case studied are about building, evaluating, deploying inteligence in data analysis.



Prediction: Predicting house prices

Models	• Linear regression • Regularization: Ridge (L2), Lasso (L1)
Algorithms	• Gradient descent • Coordinate descent
Concepts	• Loss functions, bias-variance tradeoff, cross-validation, sparsity, overfitting, model selection

Classification: Sentiment analysis

Models	• Linear classifiers (logistic regression, SVMs, perceptron) • Kernels • Decision trees
Algorithms	• Stochastic gradient descent • Boosting
Concepts	• Decision boundaries, MLE, ensemble methods, random forests, CART, online learning

Clustering& Retrieval: Finding documents

Models

- Nearest neighbors
- Clustering, mixtures of Gaussians
- Latent Dirichlet allocation (LDA)

Algorithms

- KD-trees, locality-sensitive hashing (LSH)
- K-means
- Expectation-maximization (EM)

Concepts

- Distance metrics, approximation algorithms, hashing, sampling algorithms, scaling up with map-reduce