INTRODUCTION TO DATA SCIENCE

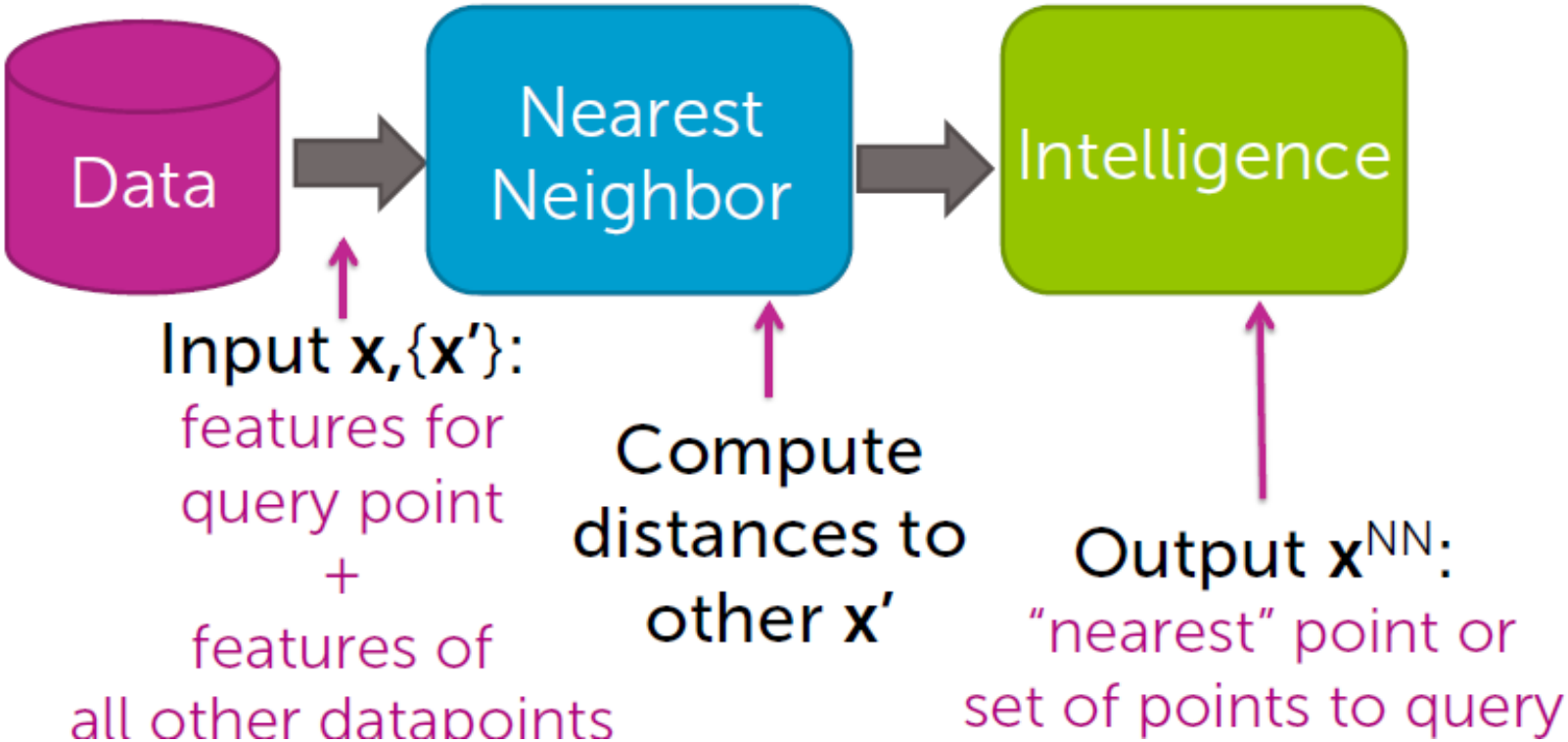
This lecture is
based on course by E. Fox and C. Guestrin, Univ of Washington

12.11, 19.11
2019

WFAiS UJ, Informatyka Stosowana
I stopień studiów

What is retrieval?

Search for related items



What is retrieval?

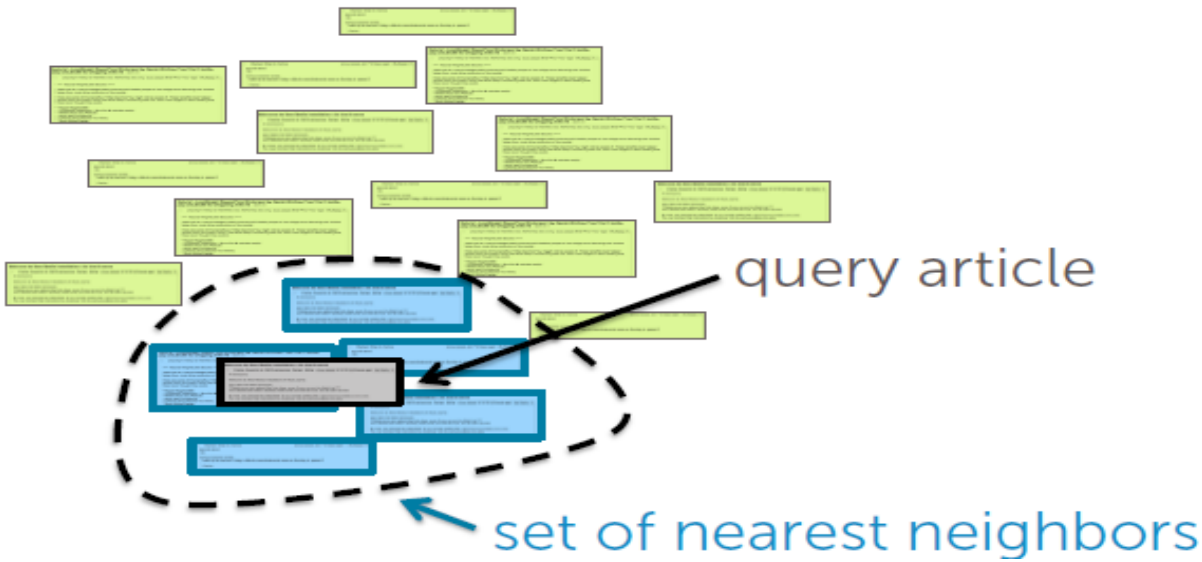
Retrieve "nearest neighbor" article

Space of all articles,
organized by similarity of text

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:

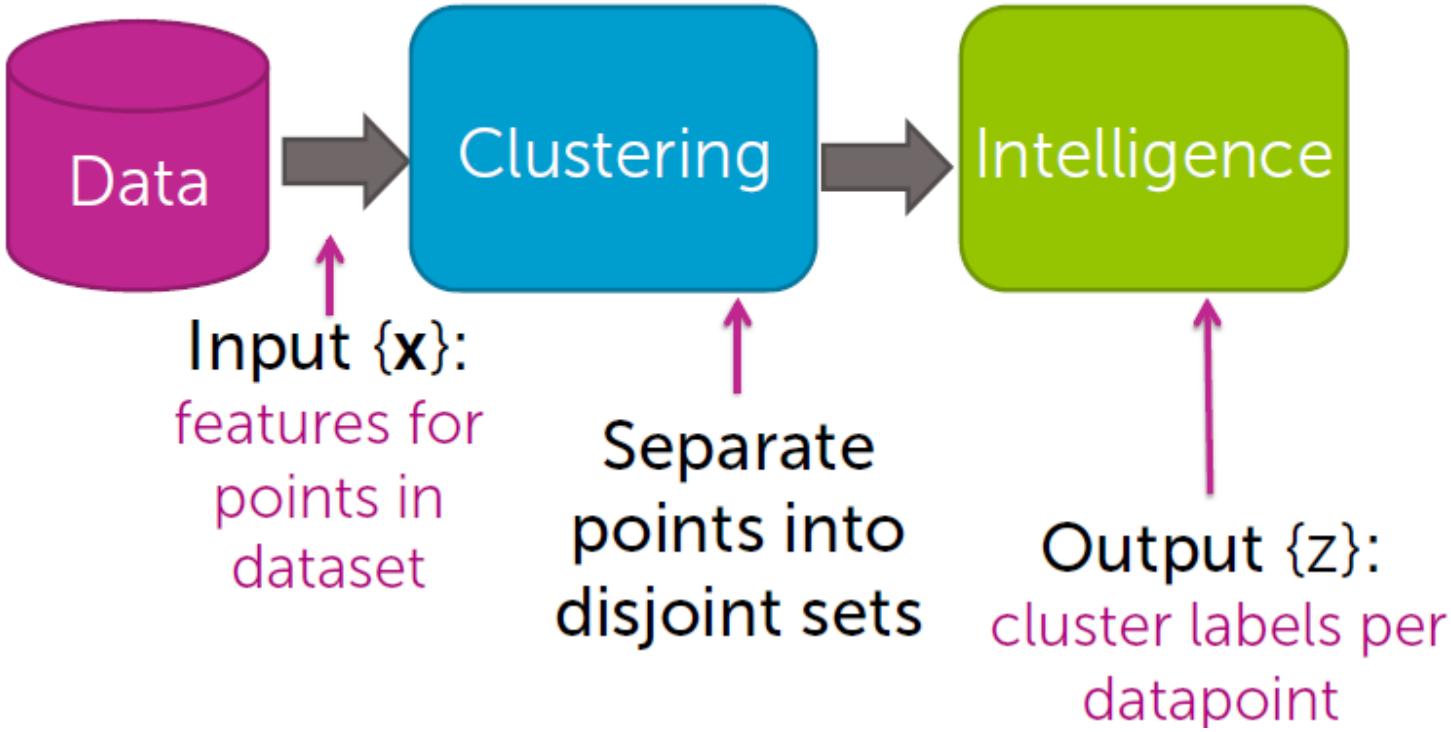
- Songs
- Movies
- TV shows
- ...

News articles

Social networks
(people you might want to connect with)

What is clustering?

Discover groups of similar inputs



Clustring applications

Clustering documents by "topic"

Clustering applications

Clustering images

For search, group as:

- Ocean
- Pink flower
- Dog
- Sunset
- Clouds
- ...

Impact of retrieval & clustering

- Foundational ideas
- Lots of information can be extracted using these tools (exploring user interests and interpretable structure relating groups of users based on observed behavior)

Overwiew of the extended content

Models

- Nearest neighbors
- Clustering
- Mixture of Gaussians
- Latent Dirichlet allocation

Algorithms

- KD-trees
- Locality sensitive hashing
- k-means
- MapReduce
- Expectation Maximization
- Gibbs sampling

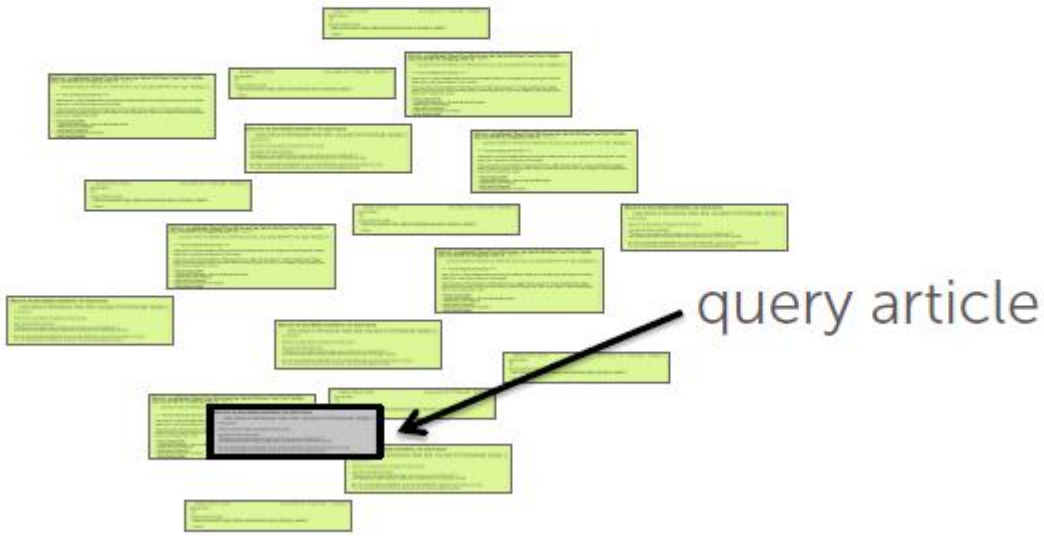
Core ML

- Distance metrics
- Approximation algorithms
- Unsupervised learning
- Probabilistic modeling
- Data parallel problems
- Bayesian inference

Retrieval as k-nearest neighbor search

1-NN search for retrieval

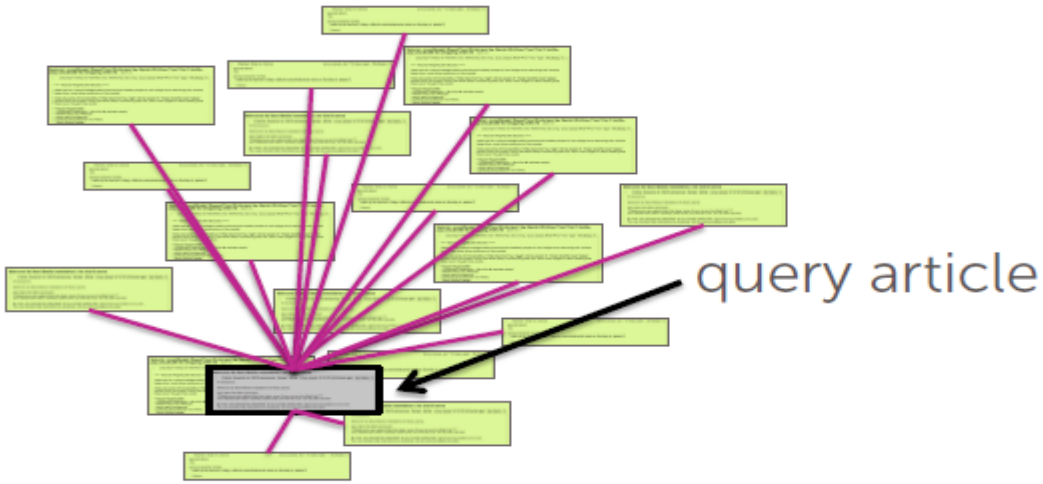
Space of all articles,
organized by similarity of text



1-NN search for retrieval

Compute distances to all docs

Space of all articles,
organized by similarity of text



1-NN search for retrieval

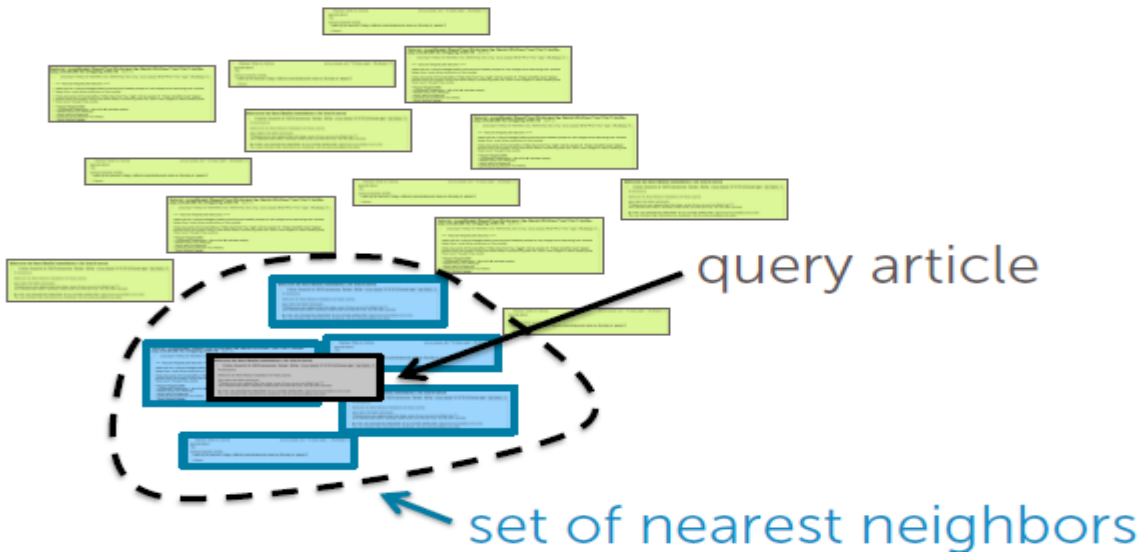
Retrieve "nearest neighbor"

Space of all articles,
organized by similarity of text

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** = ∞, = ∅

For i=1,2,...,N

query document x_q

x_i

Compute: δ = **distance(** , **)**

If δ < **Dist2NN**

document i from corpus

x_i

set =

set **Dist2NN** = δ

Return most similar document ← x^{NN}

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 <u>not in</u> 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(δ_1,…,δ_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 k^{th} 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:

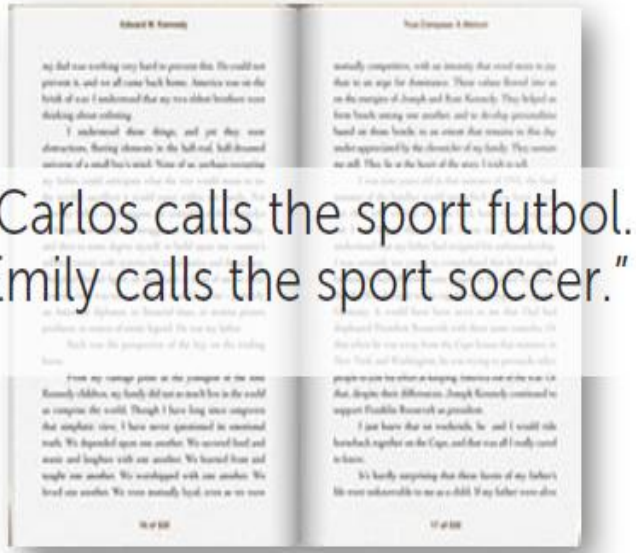
$\delta = \text{distance}(\mathbf{x}_i, \mathbf{x}_q)$

Document representation

Bag of words model

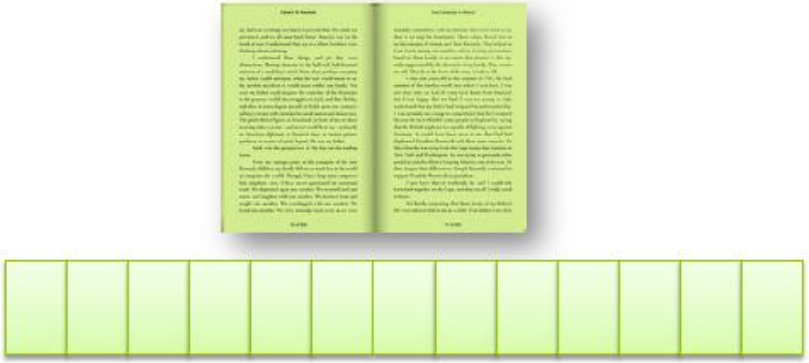
- Ignore order of words
- Count # of instances of each word in vocabulary

1	2		1			1	2			2		1
carlos	the		Emily			soccer	calls			sport		Futbol



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

tf * idf

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

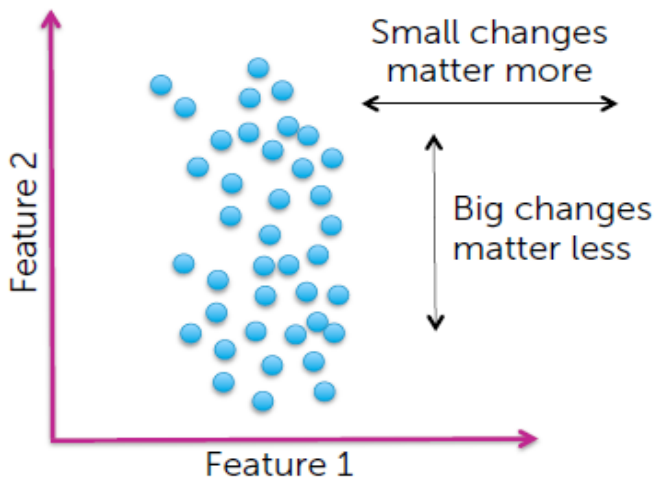
Distance metrics:

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

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

Setting weights as 0 or 1 is equivalent to **feature selection**

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

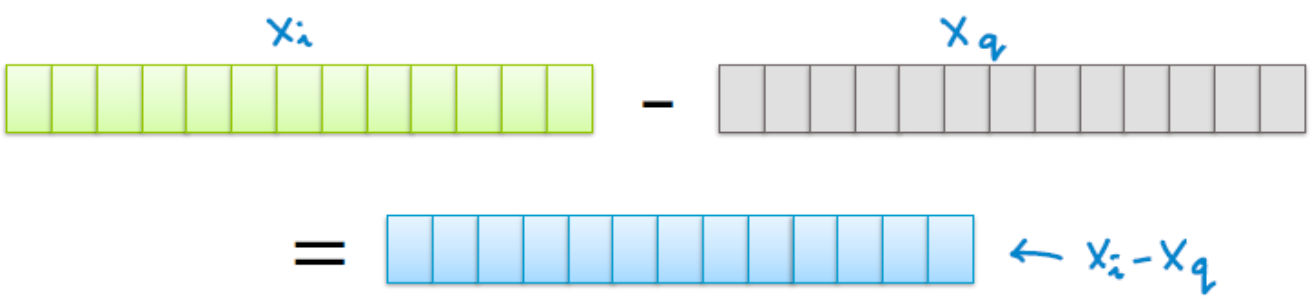
Distance metrics:

(non-scaled) Euclidean distance

Defined in terms of inner product

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

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



Distance metrics:

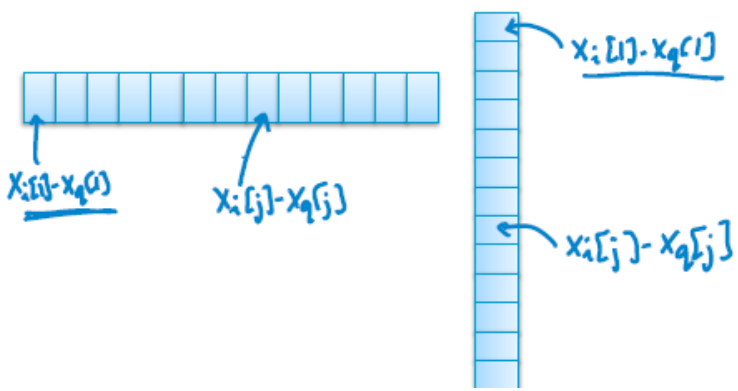
(non-scaled) Euclidean distance

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 =



take sq.rt.

Distance metrics:

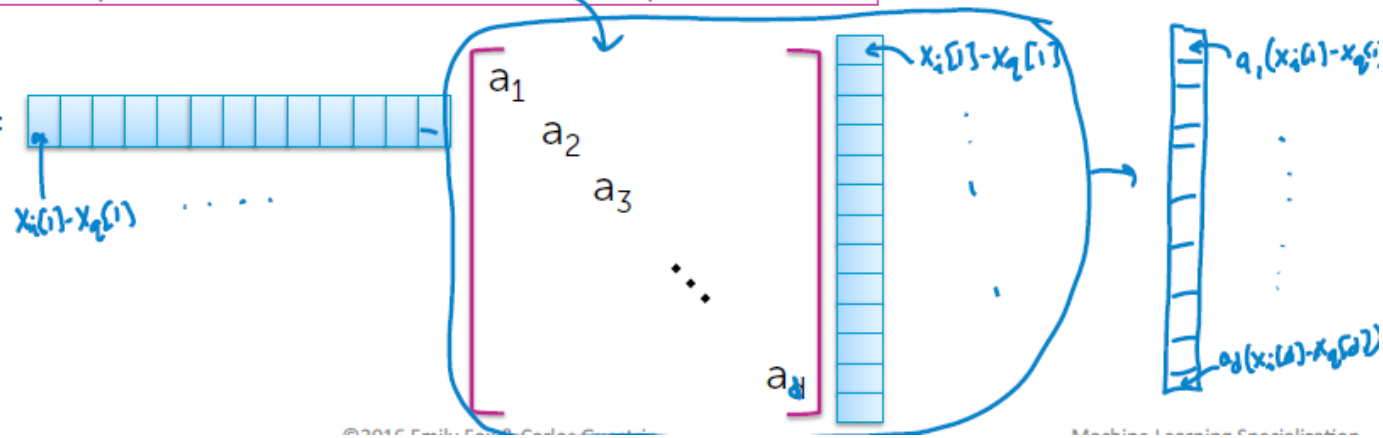
Scaled Euclidean distance

Defined in terms of inner product

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

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

distance^2 =



Distance metrics:

Another natural inner product measure

x_q

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

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

x_i

Similarity

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

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

$$= 13$$

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

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

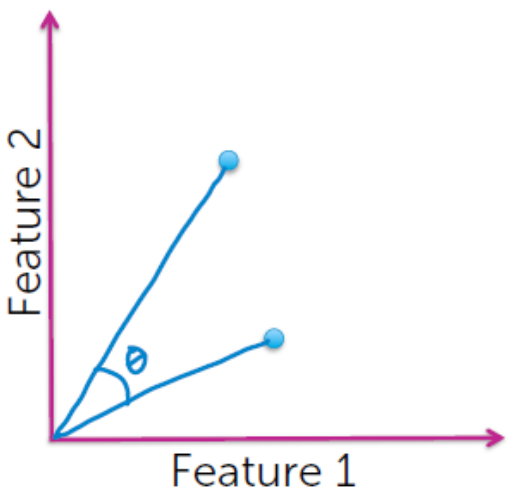
$a^T b = \|a\| \|b\| \cos(\theta)$

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

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

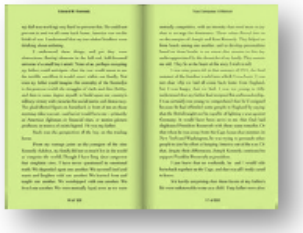
$$= \left(\frac{x_i}{\|x_i\|}\right)^T \left(\frac{x_q}{\|x_q\|}\right)$$

first normalize



Distance metrics

Normalize



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

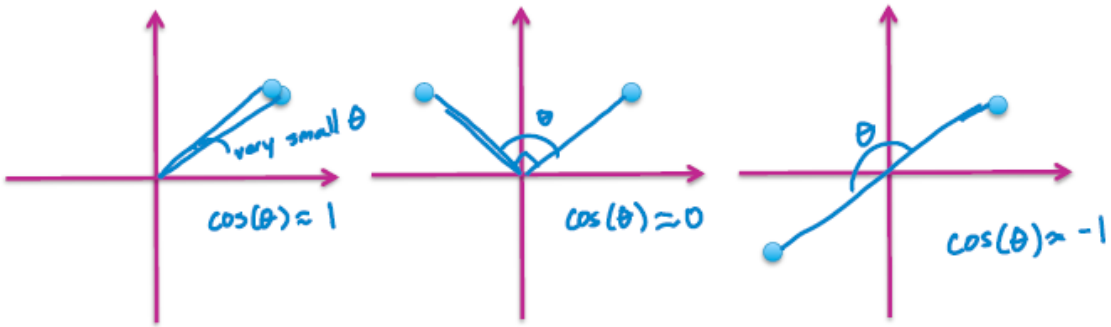
$\leftarrow x_i$

$$\sqrt{(1^2 + 5^2 + 3^2 + 1^2)} \leftarrow \|x_i\| = \sum_{j=1}^{d} x_i[j]^2$$

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

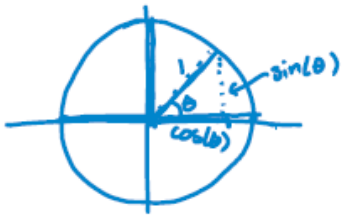
Distance metrics

Cosine similarity



In general, $-1 <$ similarity < 1

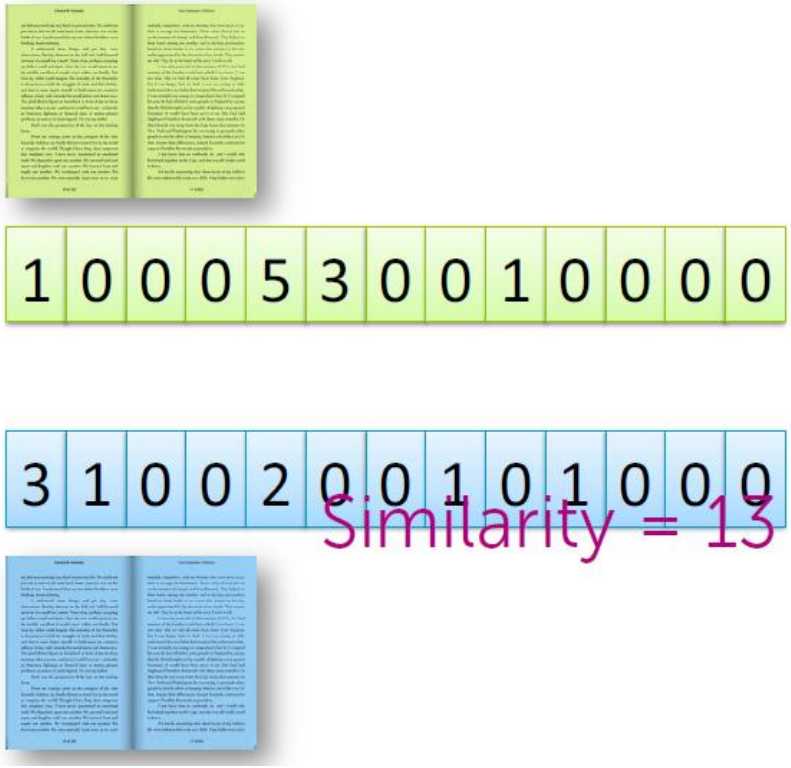
For positive features (like tf-idf) Our focus
$0 <$ similarity < 1

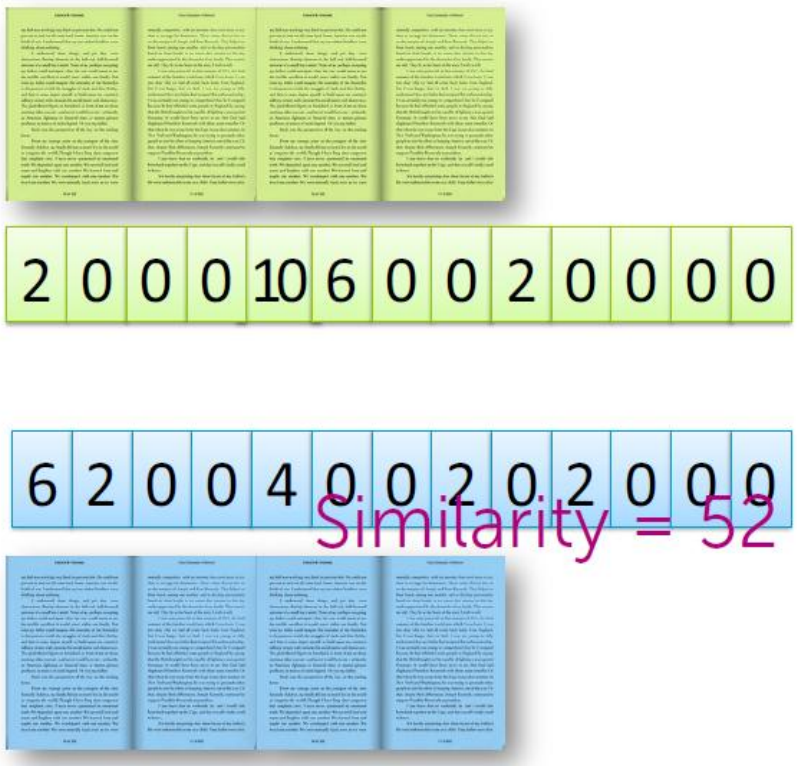


Define **distance = 1-similarity**

Distance metrics

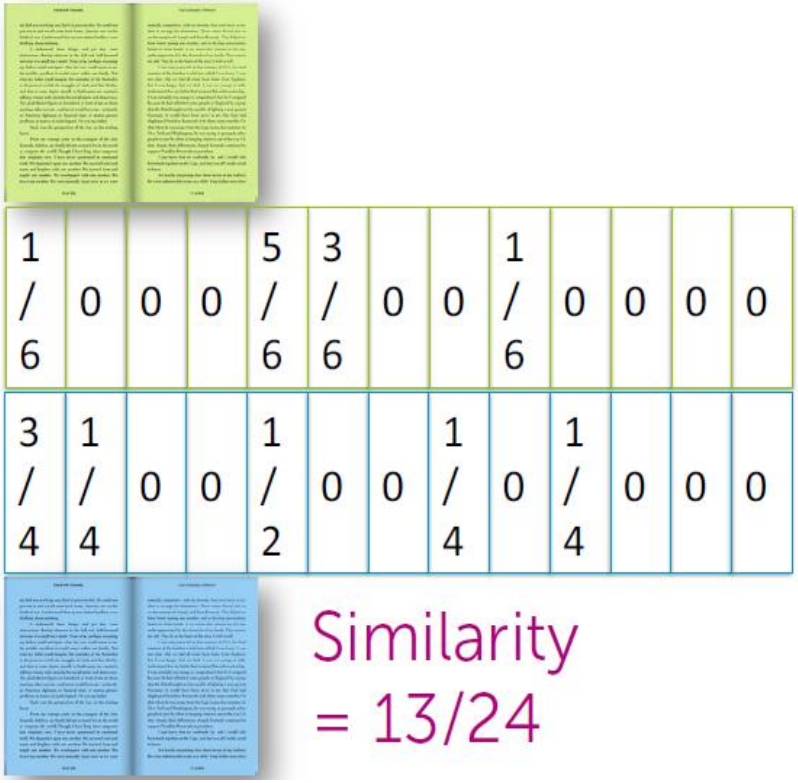
To normalize or not?





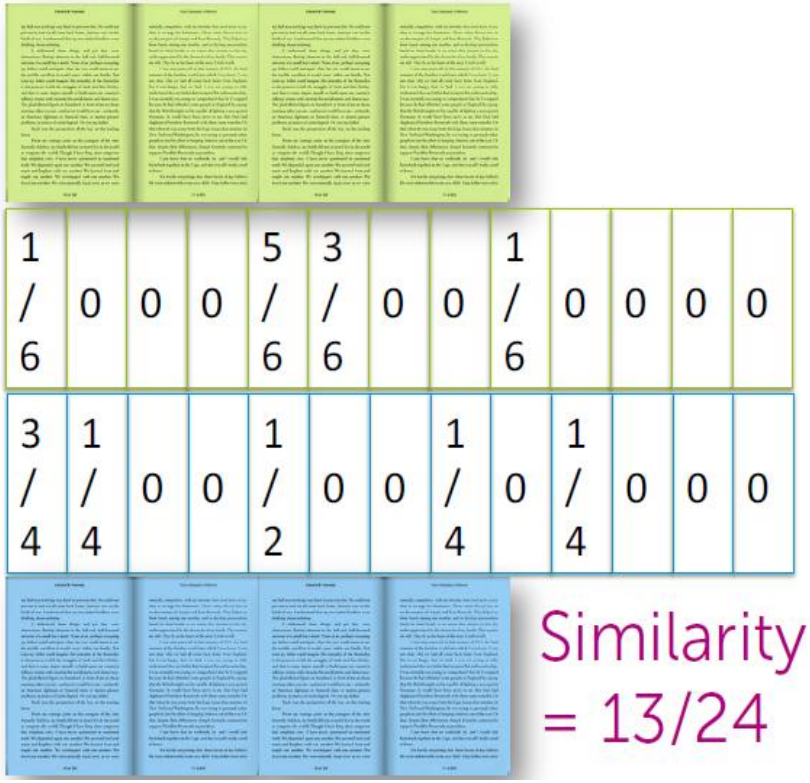
Distance metrics

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

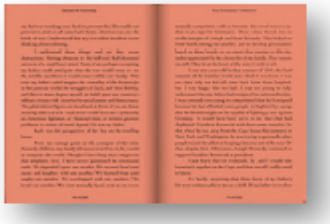
But not always desired...

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

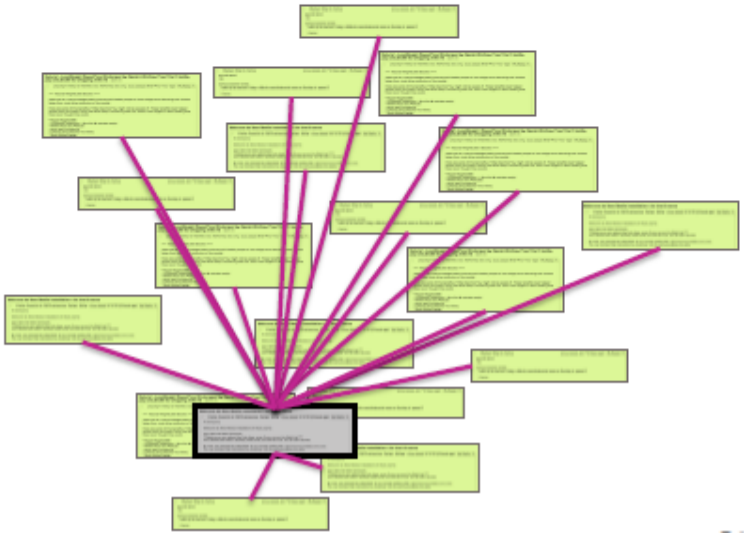
Add together with user-specified weights

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(Nlog$k$) per k-NN query!

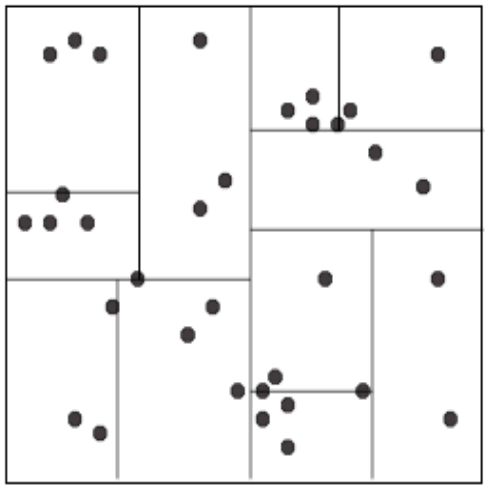


What if N is huge???
(and many queries)

KD-trees

Structured organization of documents

- Recursively partitions points into axis aligned boxes.



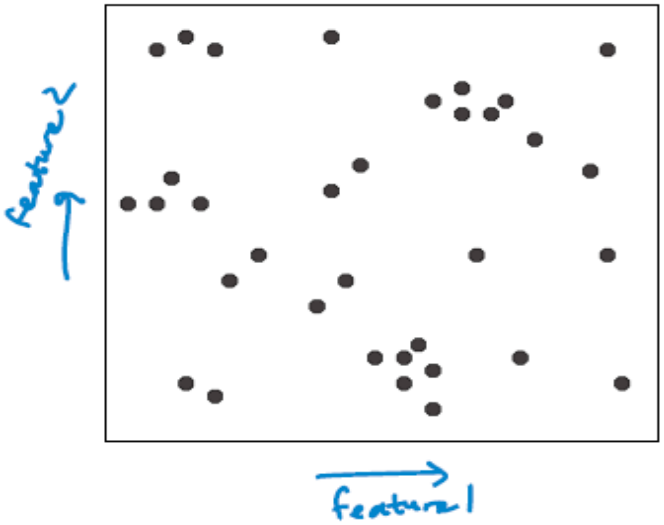
Enables more efficient pruning of search space

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
...	...	...

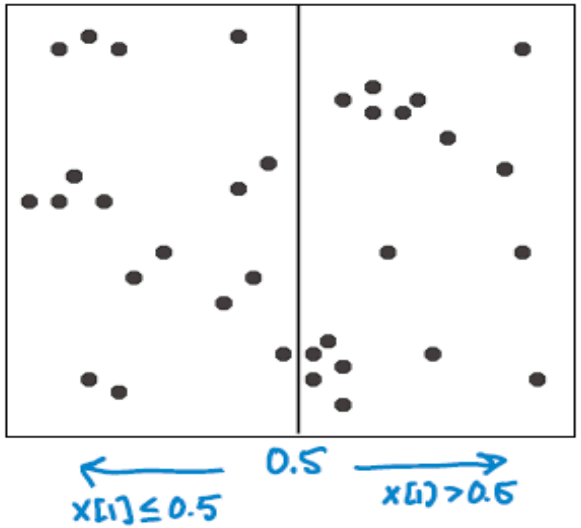
obs. indices

Feat. 1 (word 1)

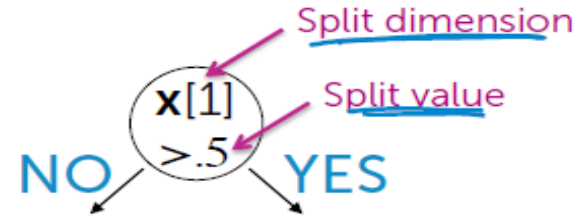
Feat. 2 (word 2)

KD-trees

KD-tree construction



Split points into 2 groups

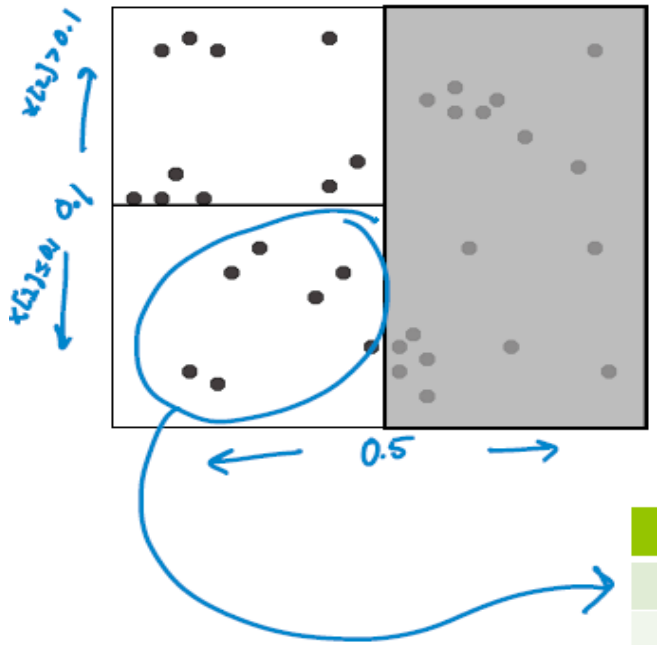


Pt	x[1]	x[2]
1	0.00	0.00
3	0.13	2.85
...	...	...

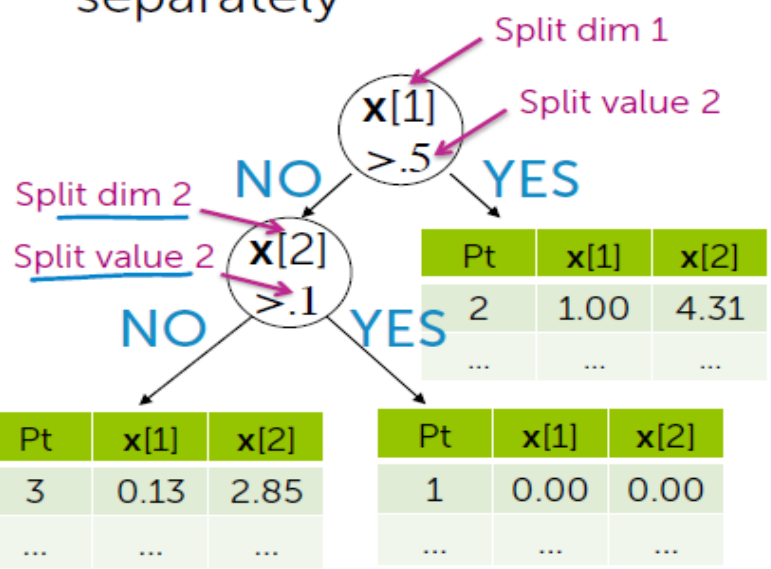
Pt	x[1]	x[2]
2	1.00	4.31
...	...	...

KD-trees

KD-tree construction



Recurse on each group separately



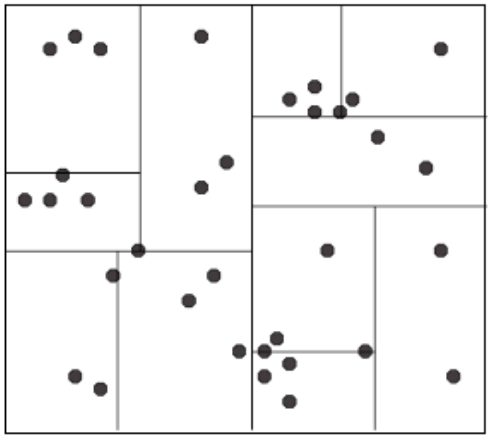
Pt	x[1]	x[2]
2	1.00	4.31
...	...	...

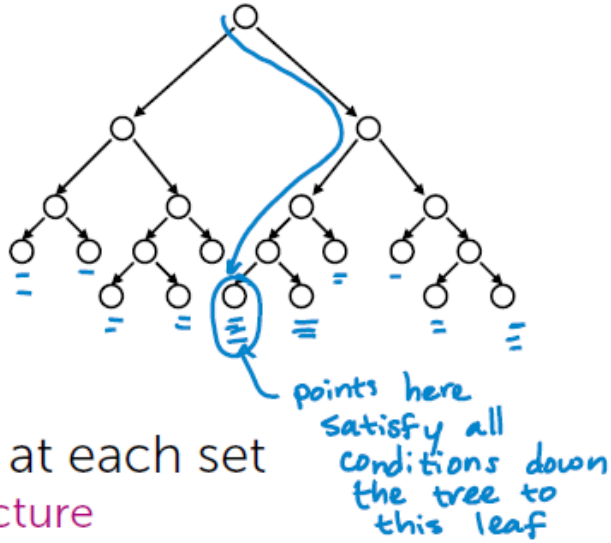
Pt	x[1]	x[2]
3	0.13	2.85
...	...	...

Pt	x[1]	x[2]
1	0.00	0.00
...	...	...

KD-trees

KD-tree construction





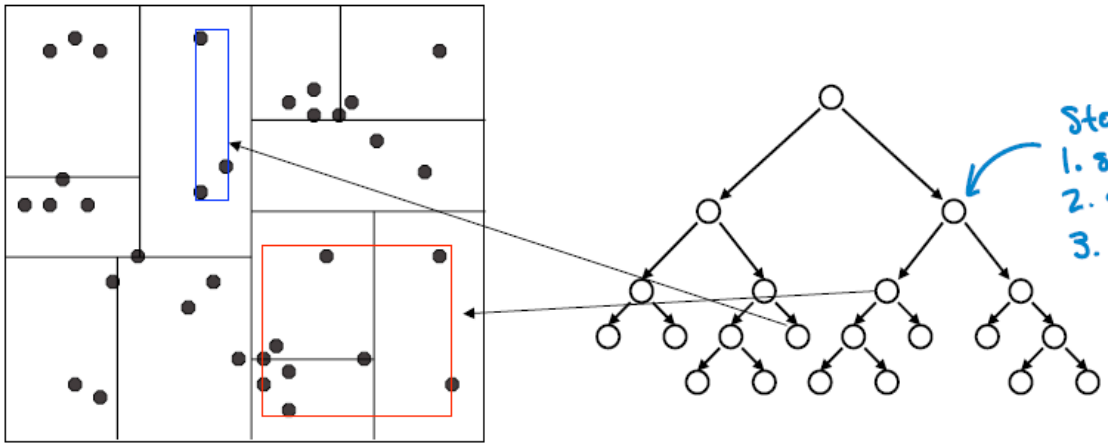
Continue splitting points at each set

- 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?

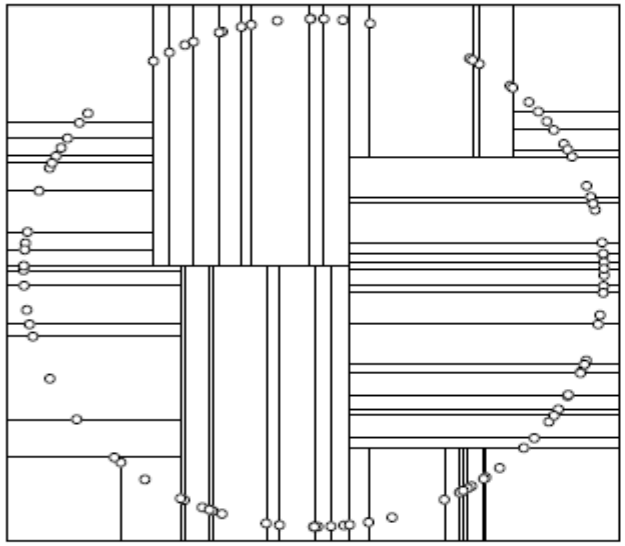
  fewer than m pts left

  or

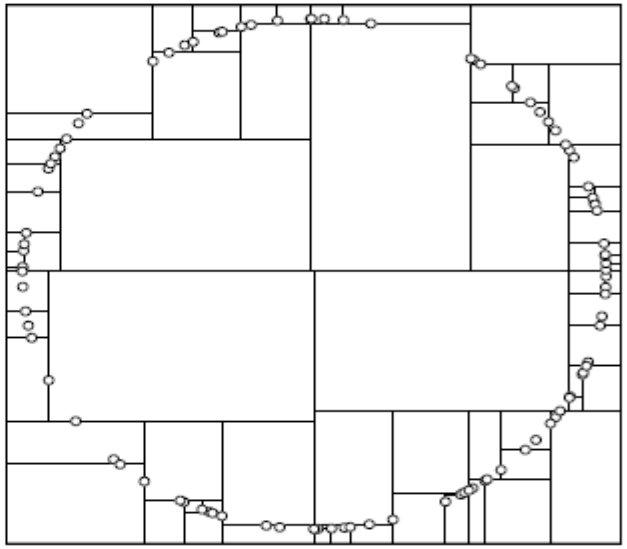
  box hits minimum width

KD-trees

Many heuristics...

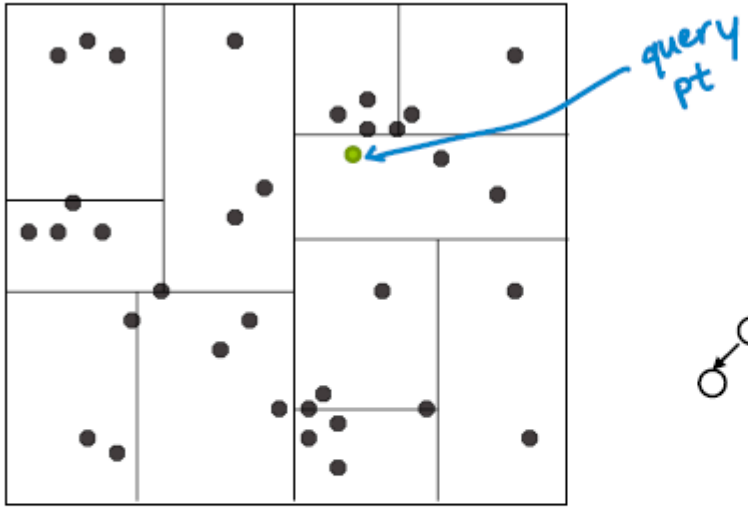


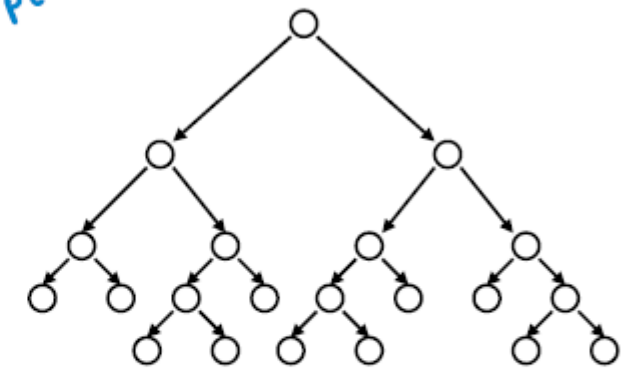
median heuristic



center-of-range heuristic

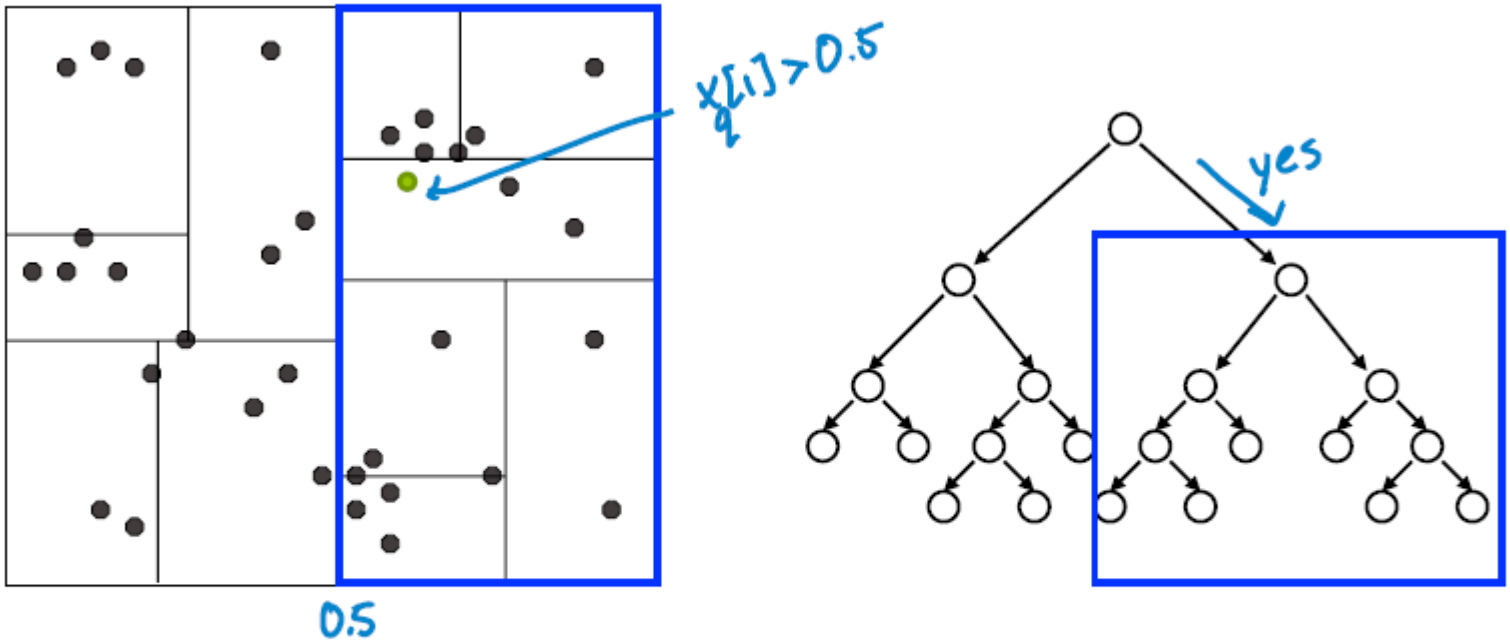
Nearest neighbor with KD-trees





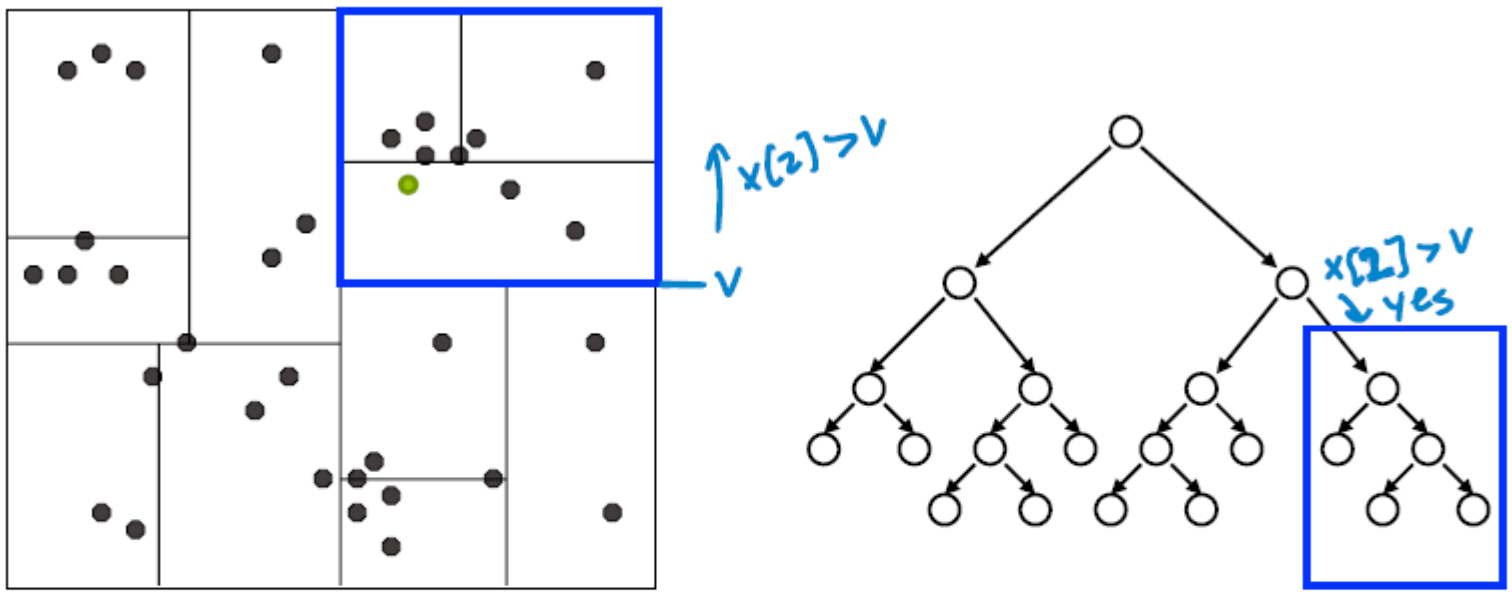
Traverse tree looking for nearest neighbor to query point

Nearest neighbor with KD-trees



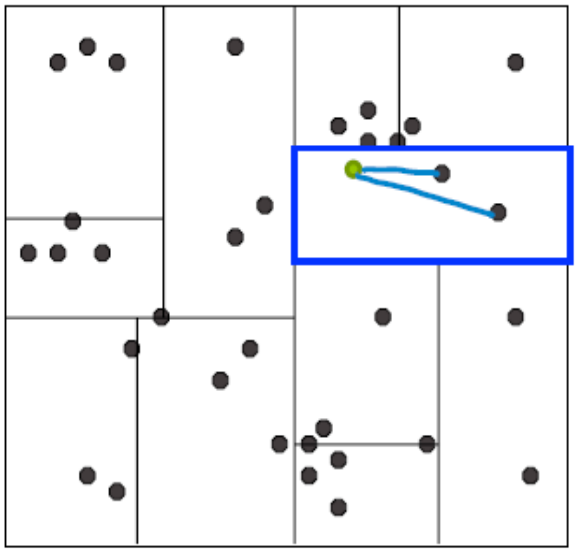
1. Start by exploring leaf node containing query point

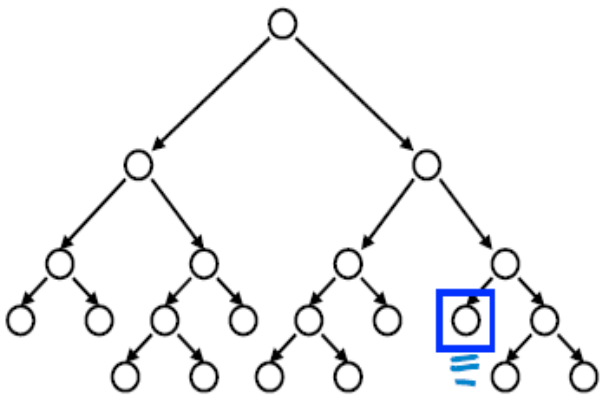
Nearest neighbor with KD-trees



1. Start by exploring leaf node containing query point

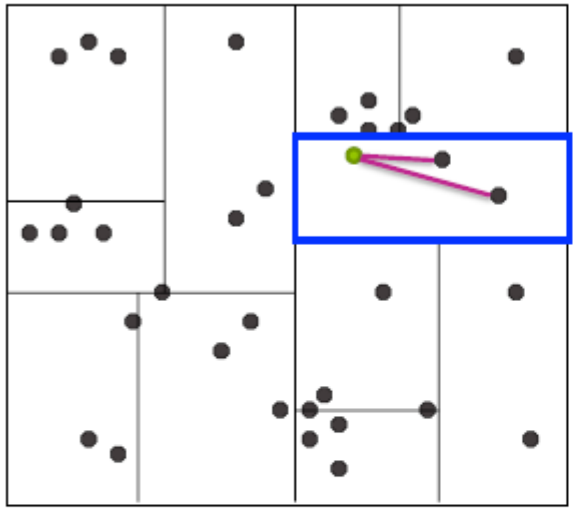
Nearest neighbor with KD-trees

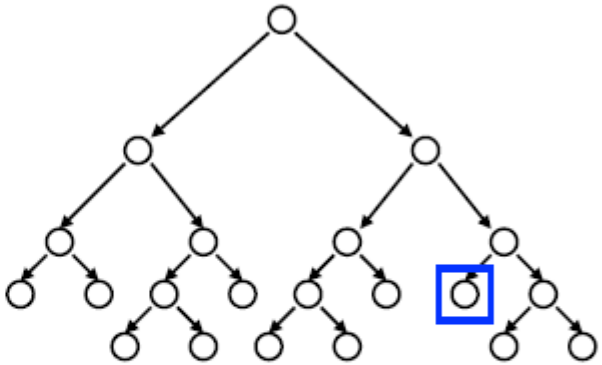




1. Start by exploring leaf node containing query point

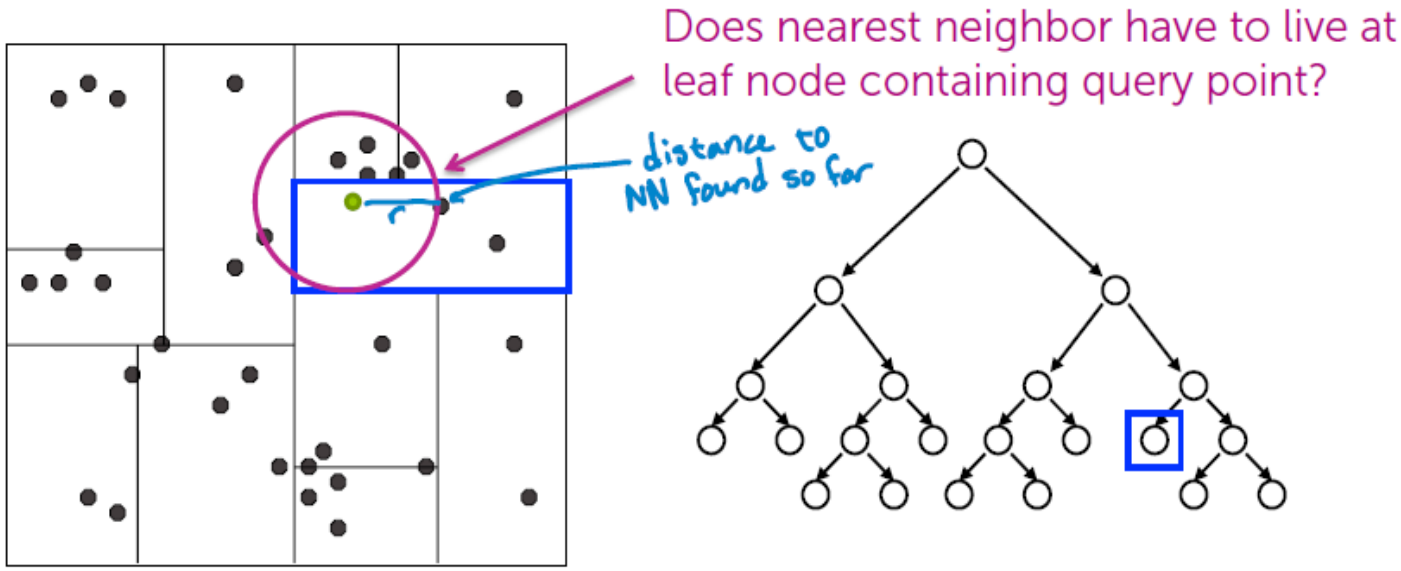
Nearest neighbor with KD-trees





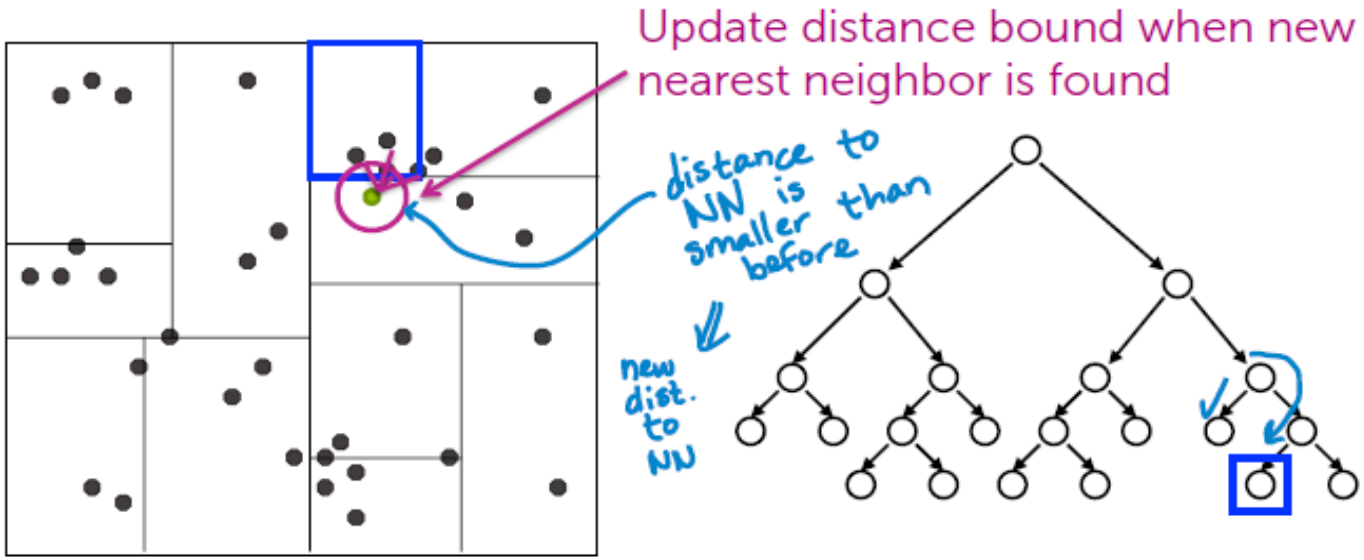
1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node

Nearest neighbor with KD-trees



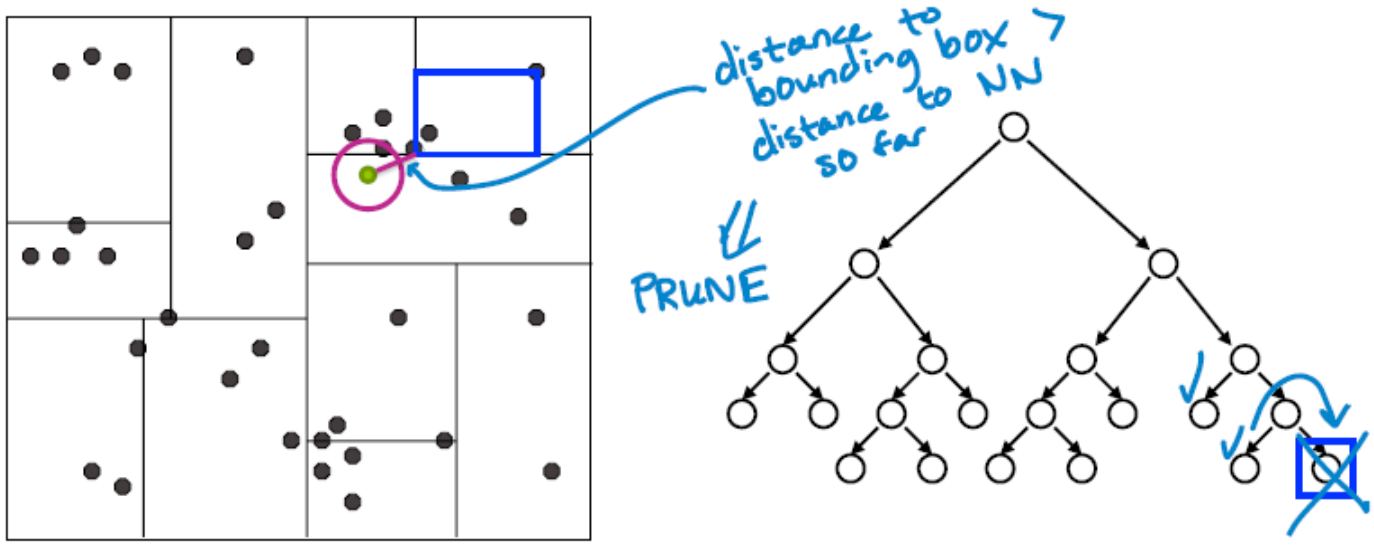
1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node

Nearest neighbor with KD-trees



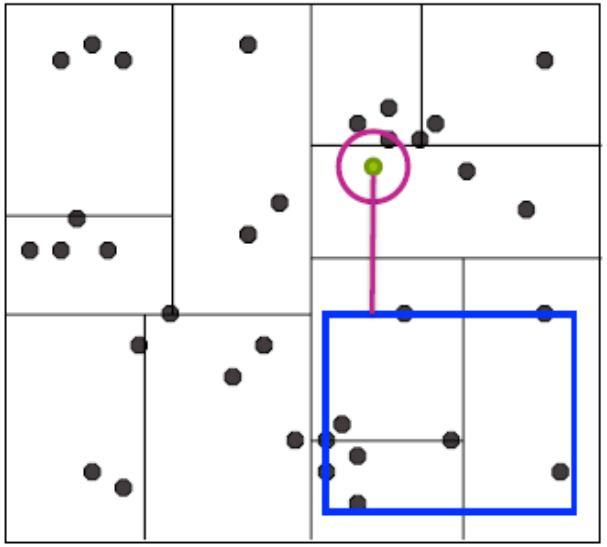
1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node
3. Backtrack and try other branch at each node visited

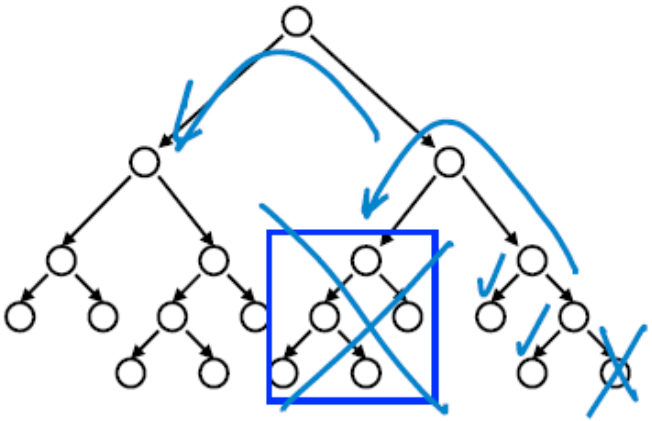
Nearest neighbor with KD-trees



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

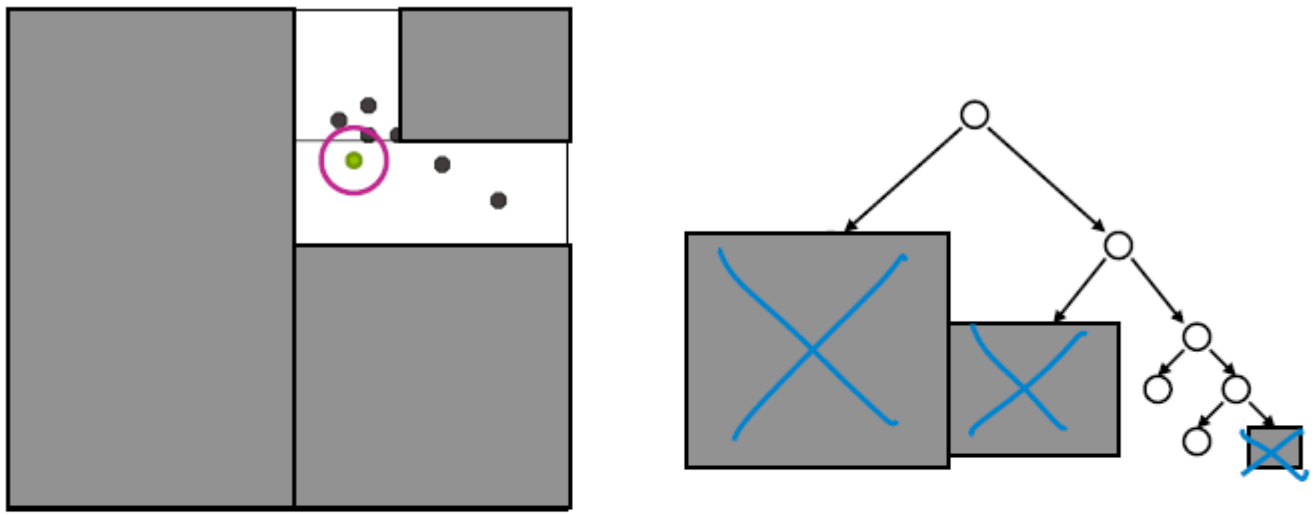
Nearest neighbor with KD-trees





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

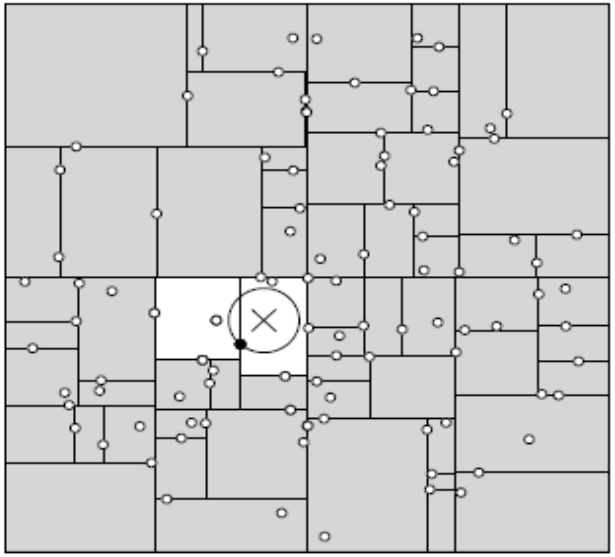
For (nearly) balanced, binary trees...

- 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)

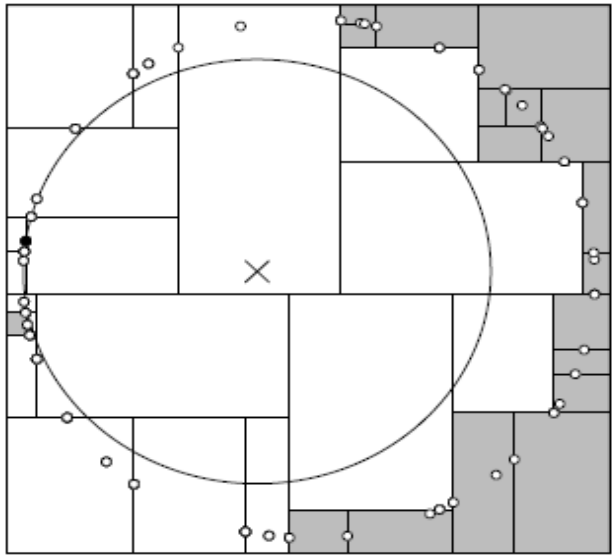
Under some assumptions on distribution of points, we get O(logN) but exponential in d

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:

  $O(N^2)$

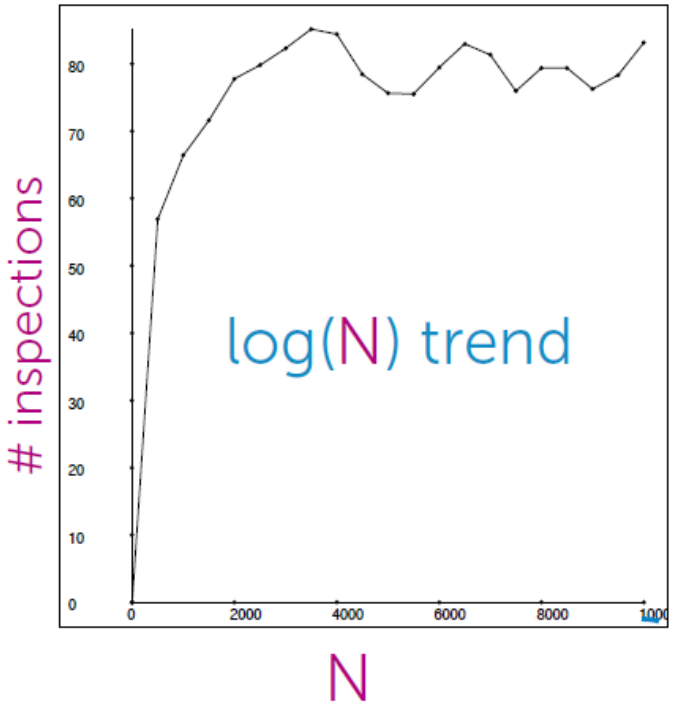
- kd-trees:

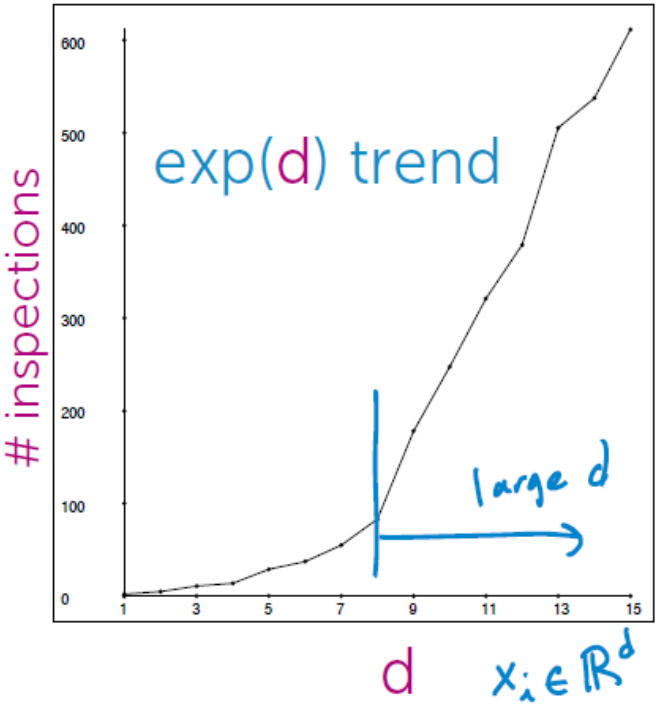
  $O(N \log N) \rightarrow O(N^2)$

  potentially very large savings for large N!

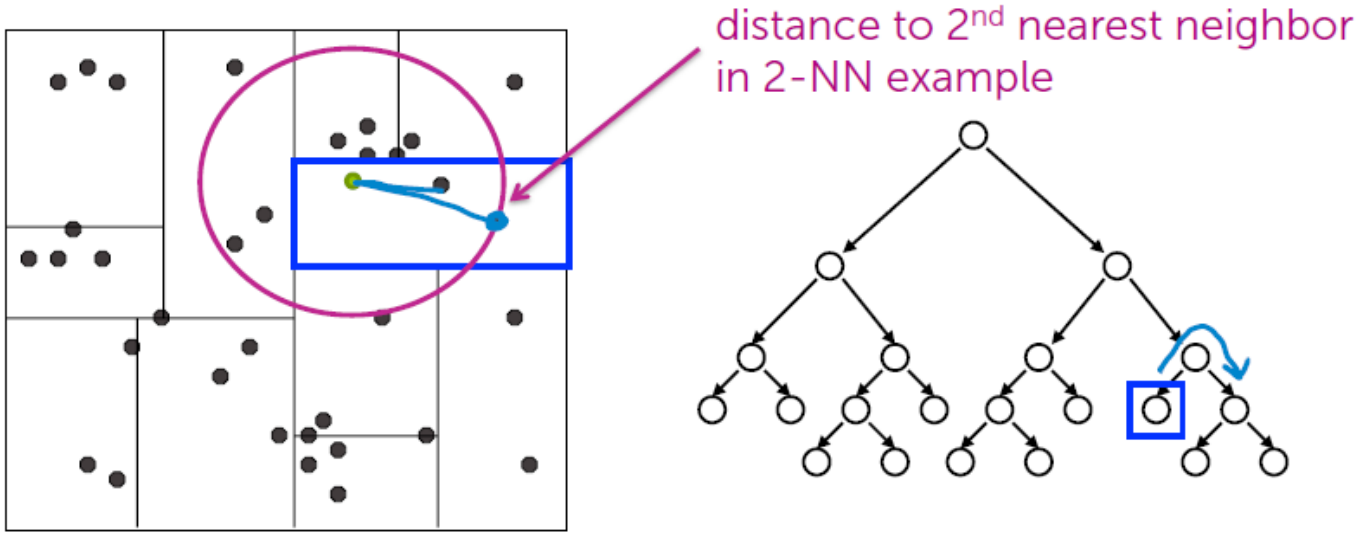
Complexity for N queries

Inspections vs. N and d



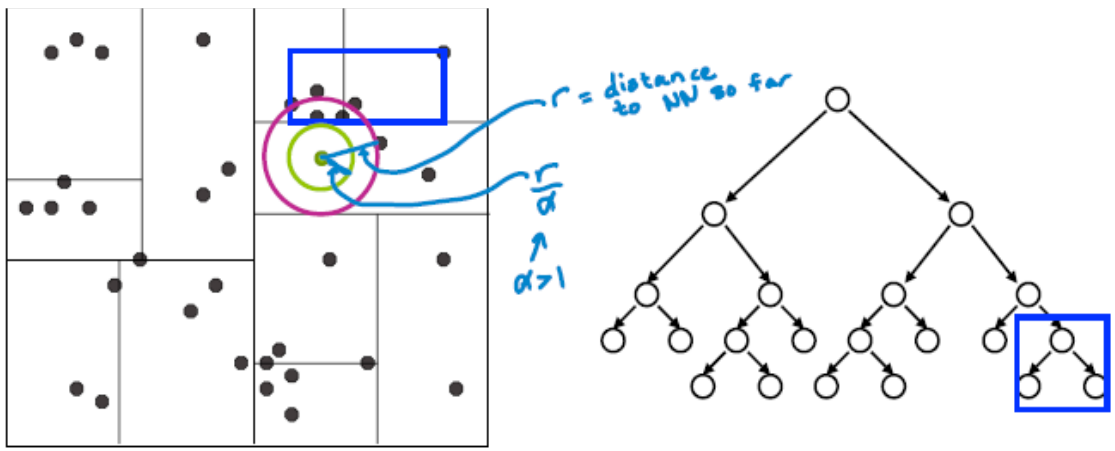


k-NN with KD-trees



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

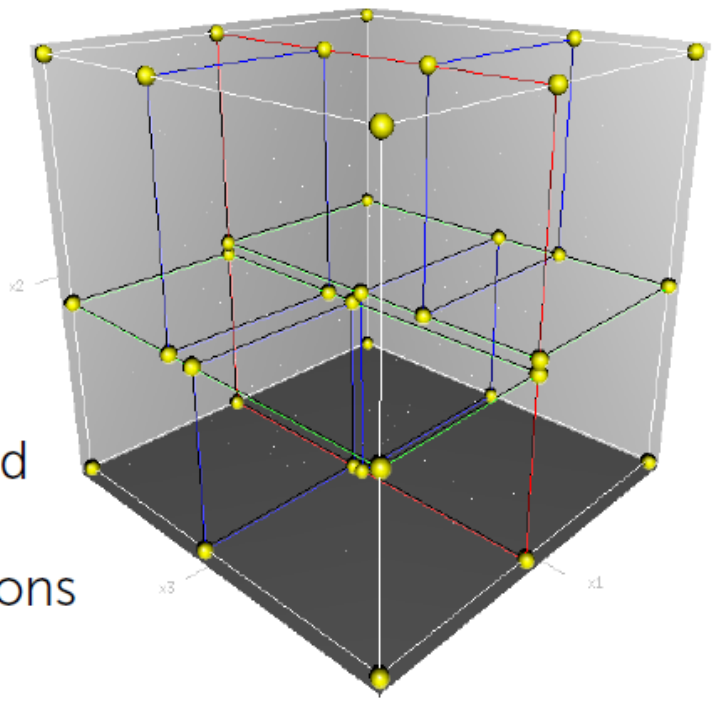
- Distance metric and data representation crucial to answer returned

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???

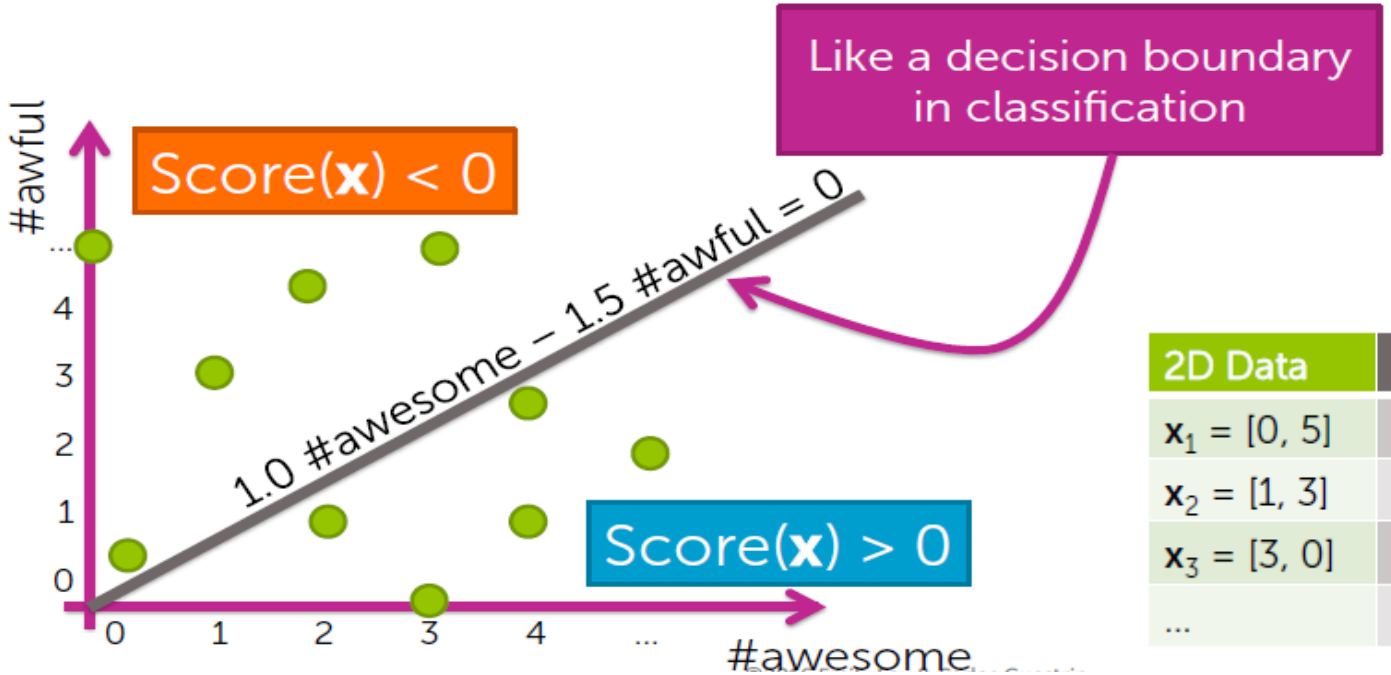
- Focus on methods that provide good probabilistic guarantees on approximation

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



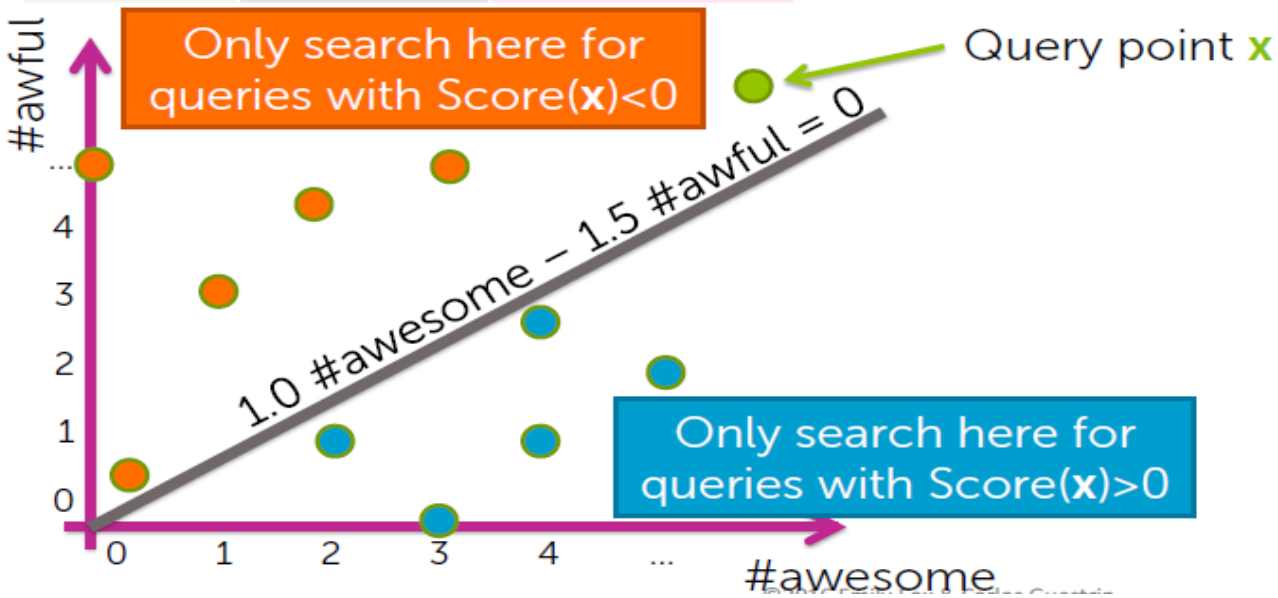
2D Data	Sign(Score)
$\mathbf{x}_1$ = [0, 5]	-1
$\mathbf{x}_2$ = [1, 3]	-1
$\mathbf{x}_3$ = [3, 0]	1
...	...

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(**x**)<0



Locality sensitive hashing

Using score 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

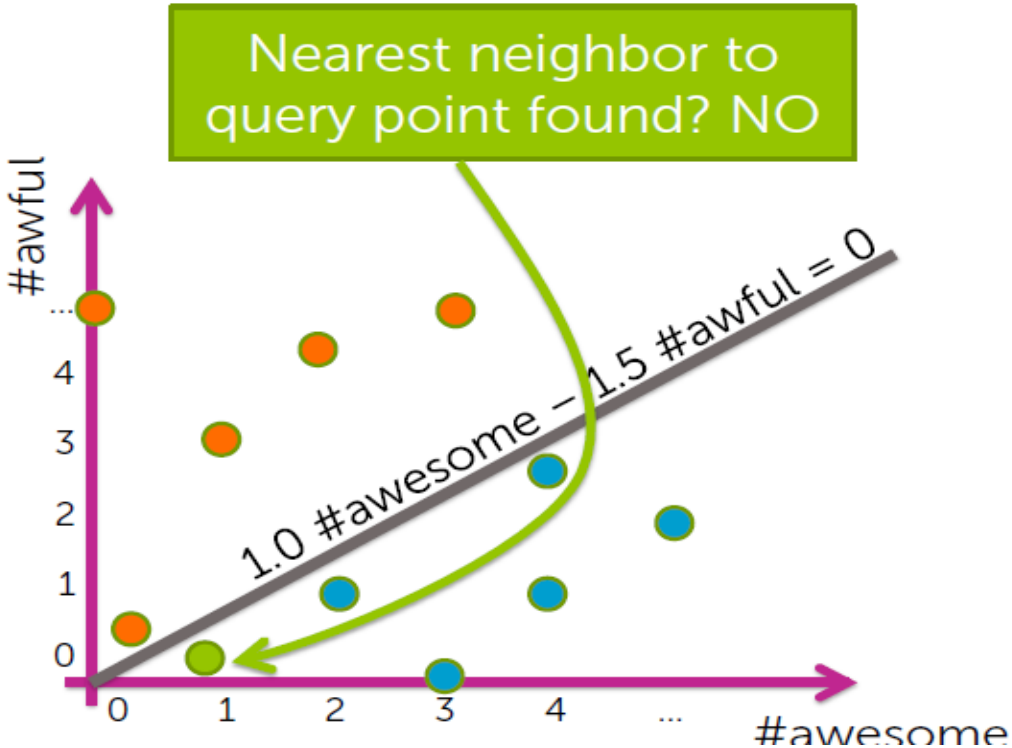
Bin	0	1
List containing indices of datapoints:	{1,2,4,7,...}	{3,5,6,8,...}

HASH TABLE

search for NN amongst this set

Locality sensitive hashing

Provides approximate NN



Locality sensitive hashing

Three potential issues with simple approach

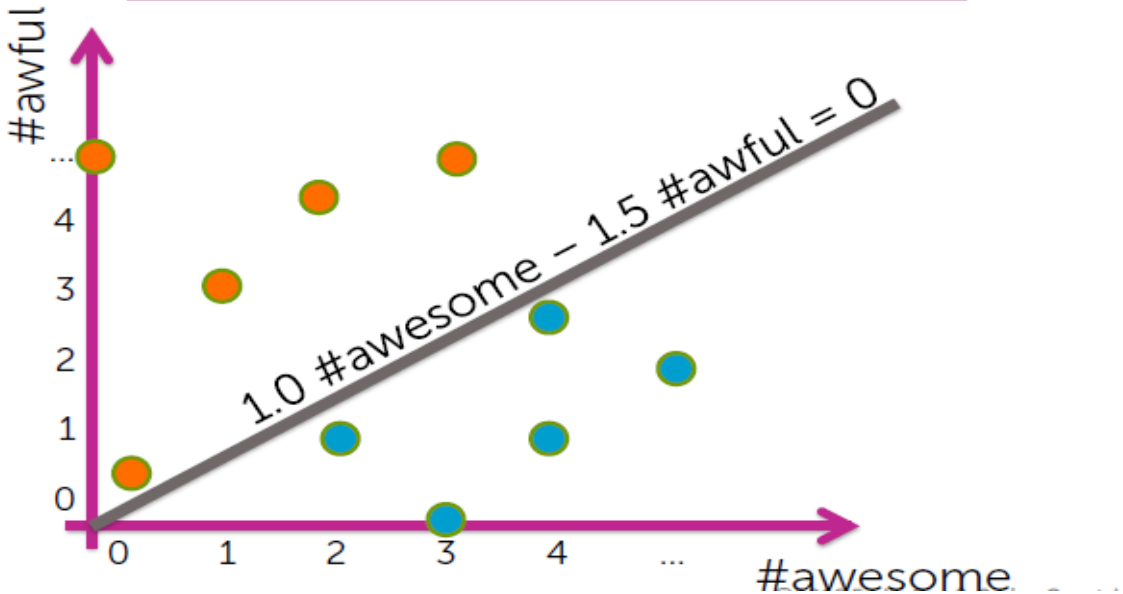
1. Challenging to find good line
2. Poor quality solution:
   - Points close together get split into separate bins
3. Large computational cost:
   - Bins might contain many points, so still searching over large set for each NN query

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

Locality sensitive hashing

How to define the line?

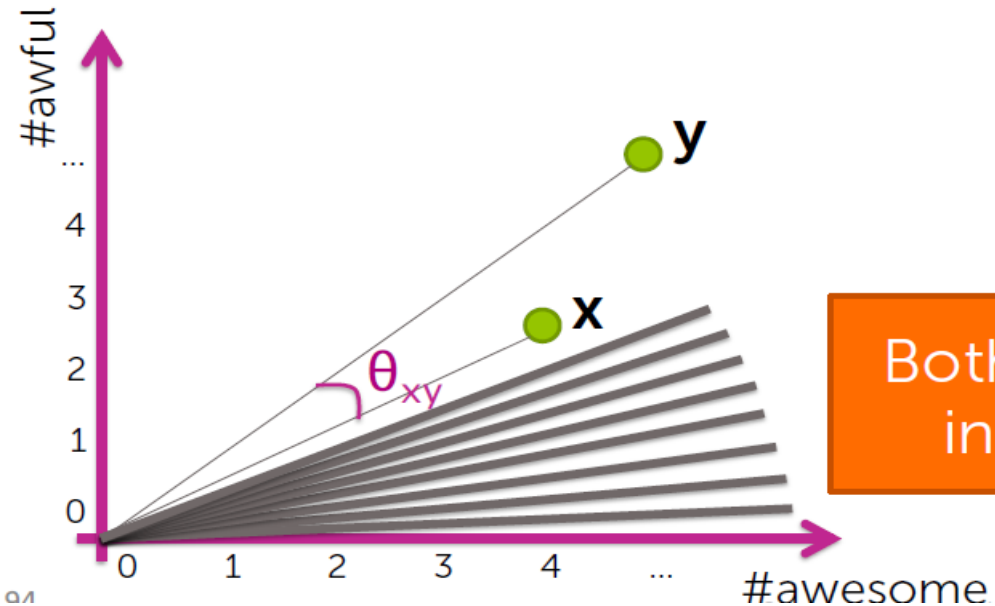
Crazy idea:
Define line randomly!



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.

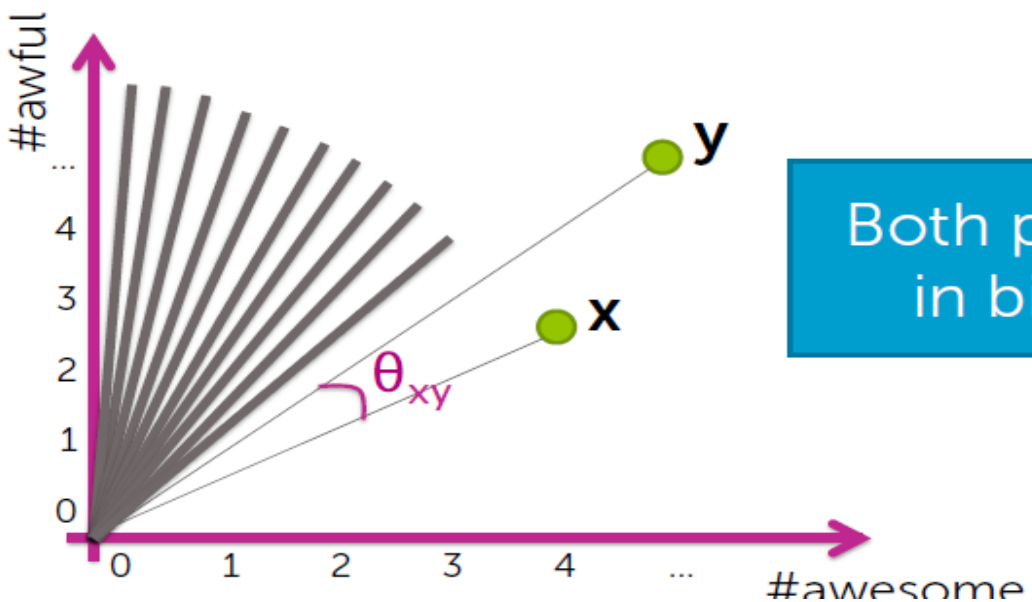


Both points in bin 0

Locality sensitive hashing

How bad can a random line be?

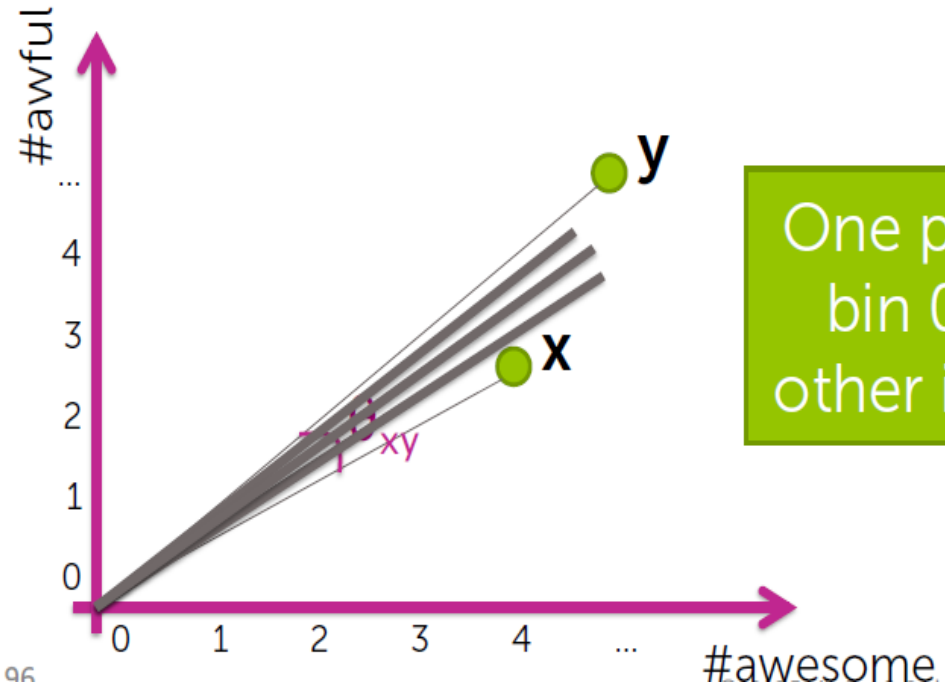
Goal: If **x**,**y** are close (according to cosine similarity), want binned values to be the same.



Both points in bin 1

Locality sensitive hashing

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

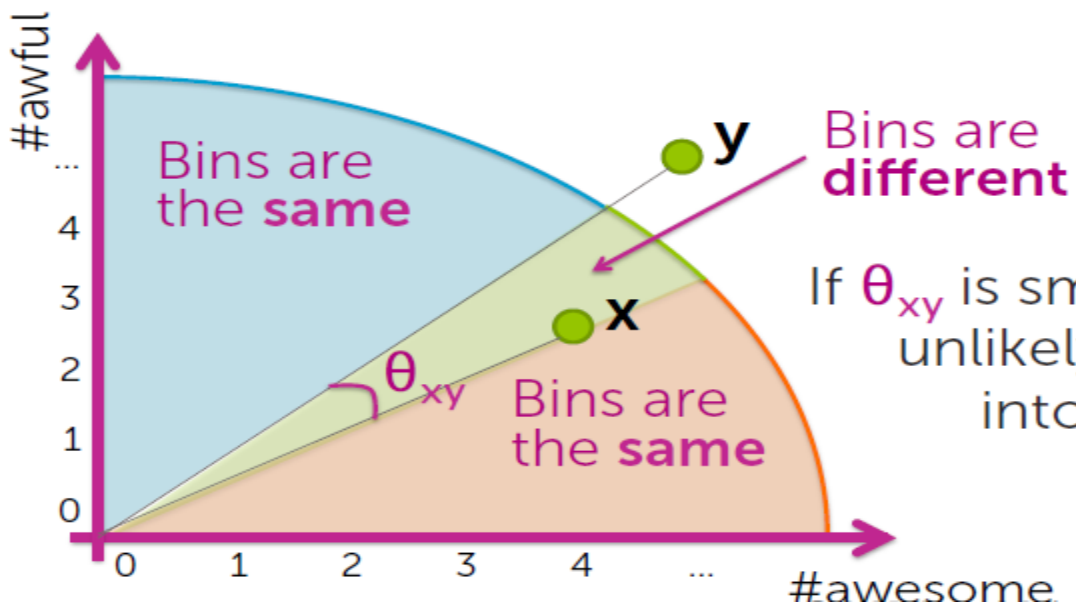


One point in bin 0 and other in bin 1

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.

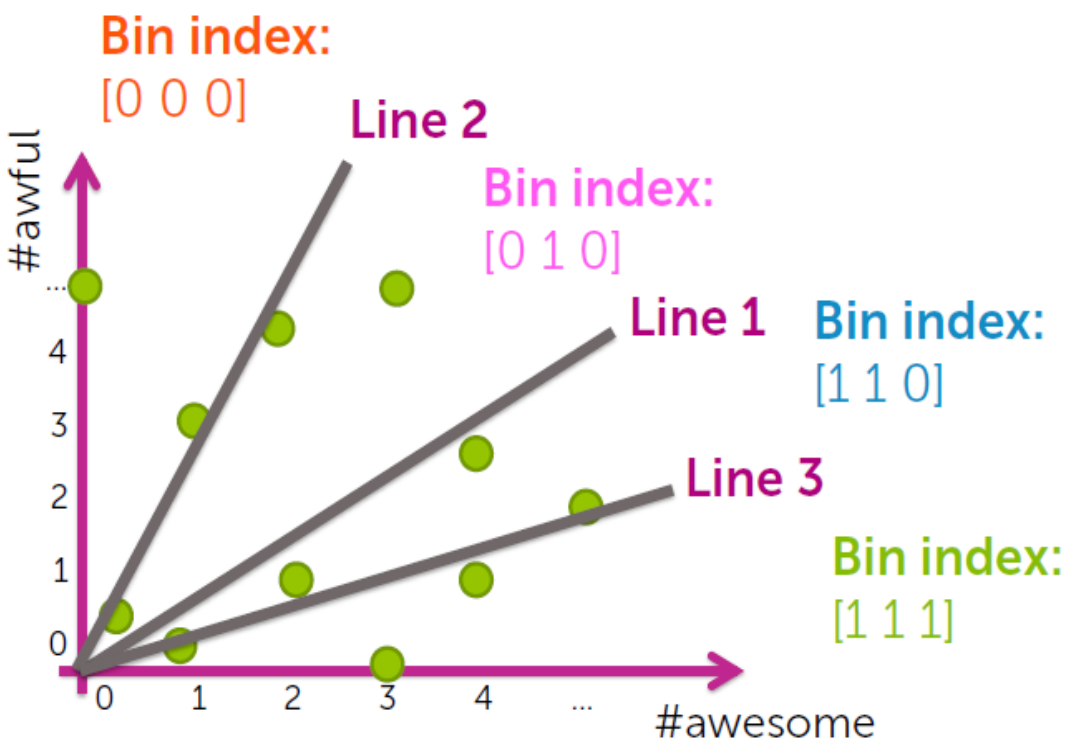


Bins are **different**

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

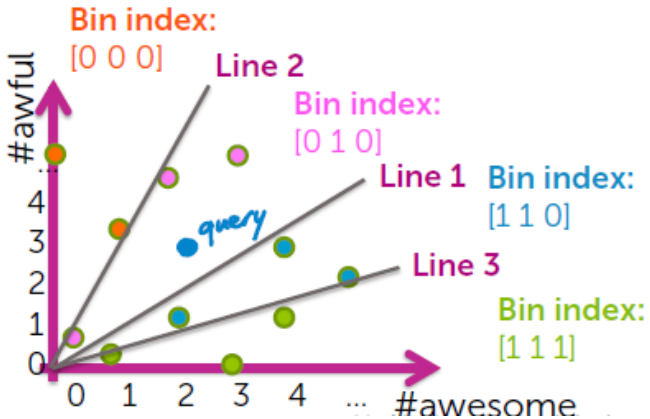
Improving search quality by searching neighboring bins

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

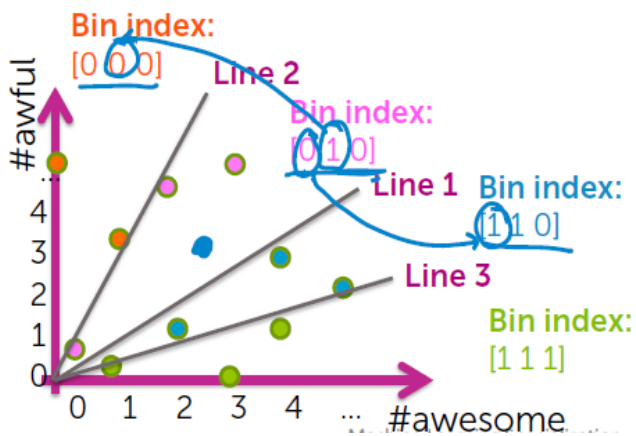


LSH: improving efficiency

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)

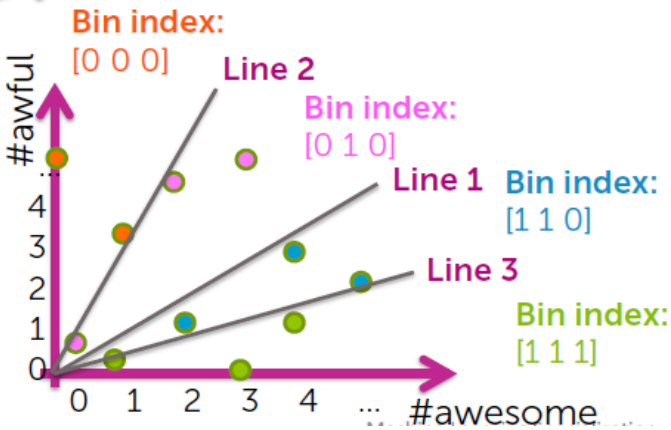


LSH: improving efficiency

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

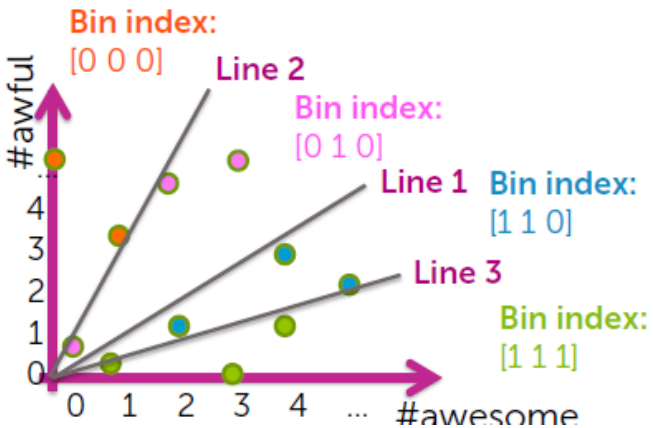
Improving search quality by searching neighboring bins

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



LSH recap

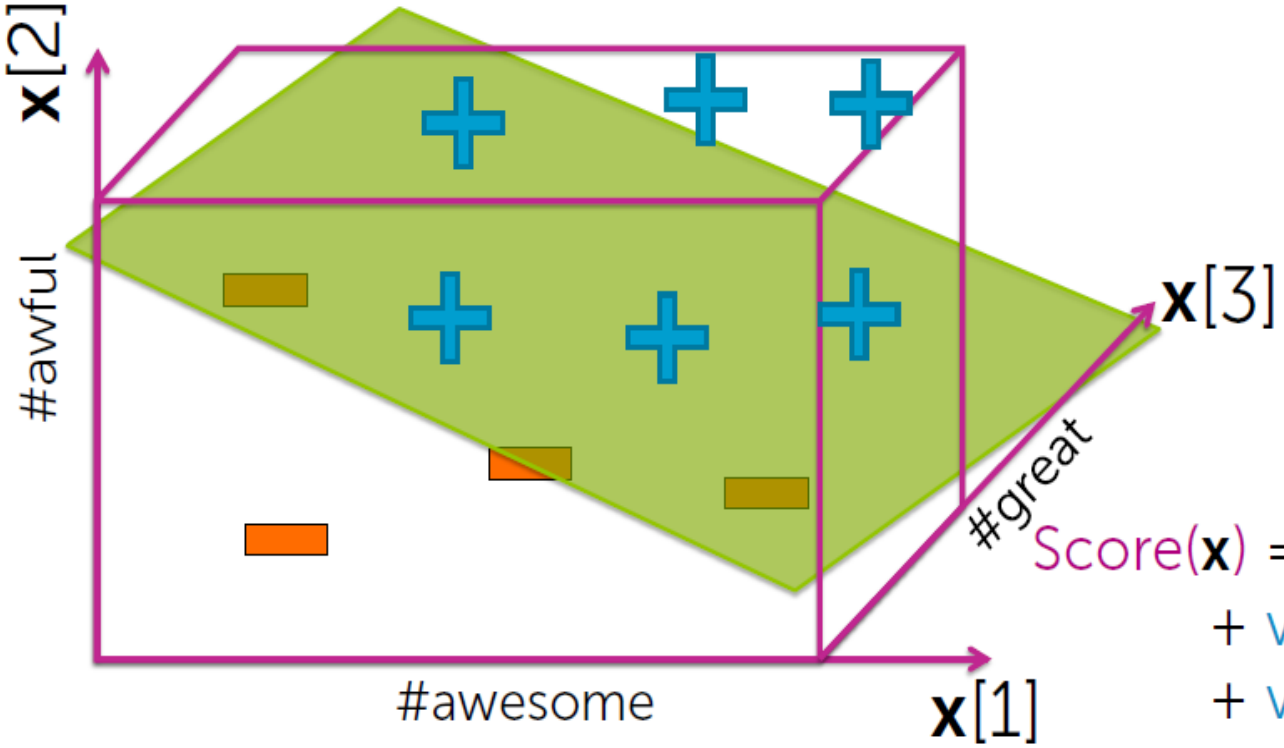
kd-tree competitor data structure

- 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*



Score(**x**) = v_1 #awesome
+ v_2 #awful
+ v_3 #great

LSH: moving to higher dimmensions d

Cost of binning points in d-dim

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

ith 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

What you can do now …

- Implement nearest neighbor search for retrieval tasks
- Contrast document representations (e.g., raw word counts, tf-idf,…)
  - Emphasize important words using tf-idf
- Contrast methods for measuring similarity between two documents
  - Euclidean vs. weighted Euclidean
  - Cosine similarity vs. similarity via unnormalized inner product
- Describe complexity of brute force search
- Implement KD-trees for nearest neighbor search
- Implement LSH for approximate nearest neighbor search
- Compare pros and cons of KD-trees and LSH, and decide which is more appropriate for given dataset

Clustering:
An unsupervised learning task

Motivation

Goal: Structure documents by topic

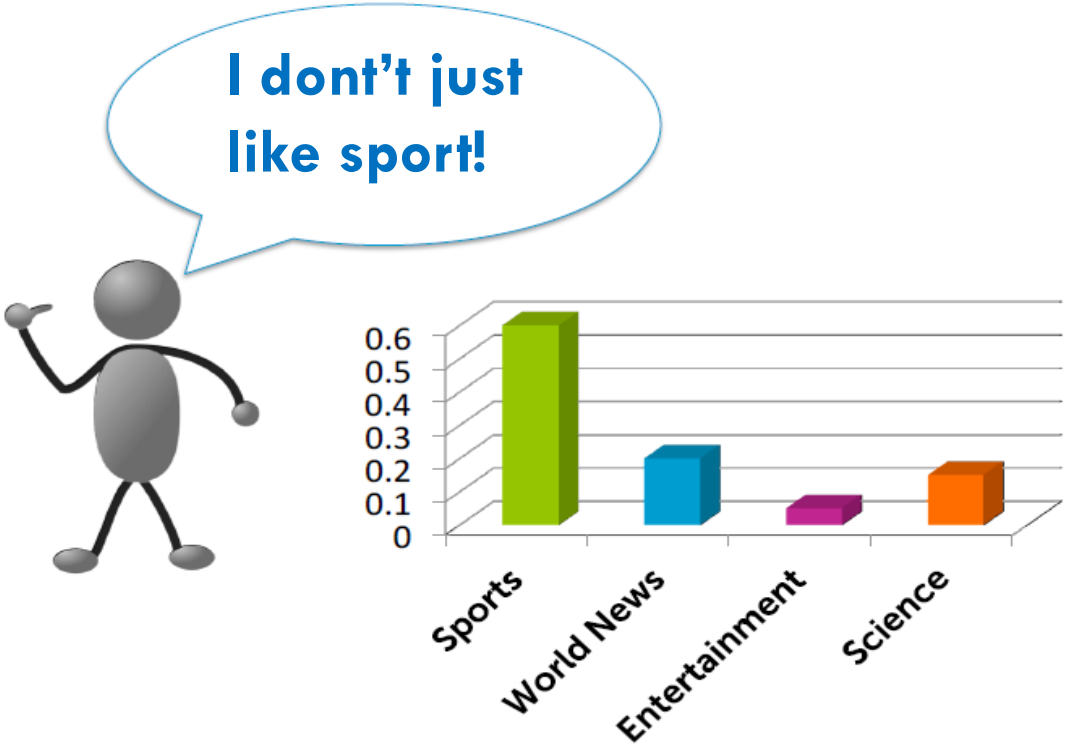
Discover groups (*clusters*) of related articles

SPORTS

WORLD NEWS

Motivation

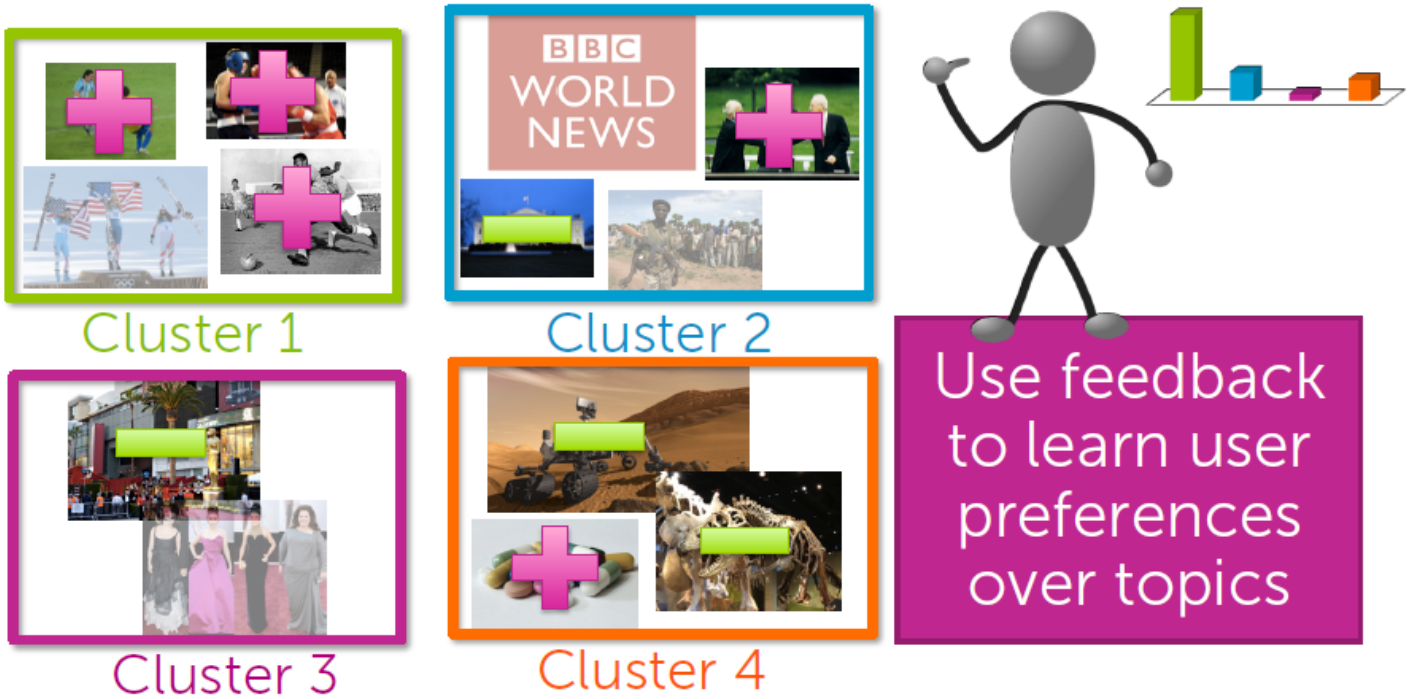
Why might clustering be useful?



Motivation

Learn user preferences

Set of clustered documents read by user



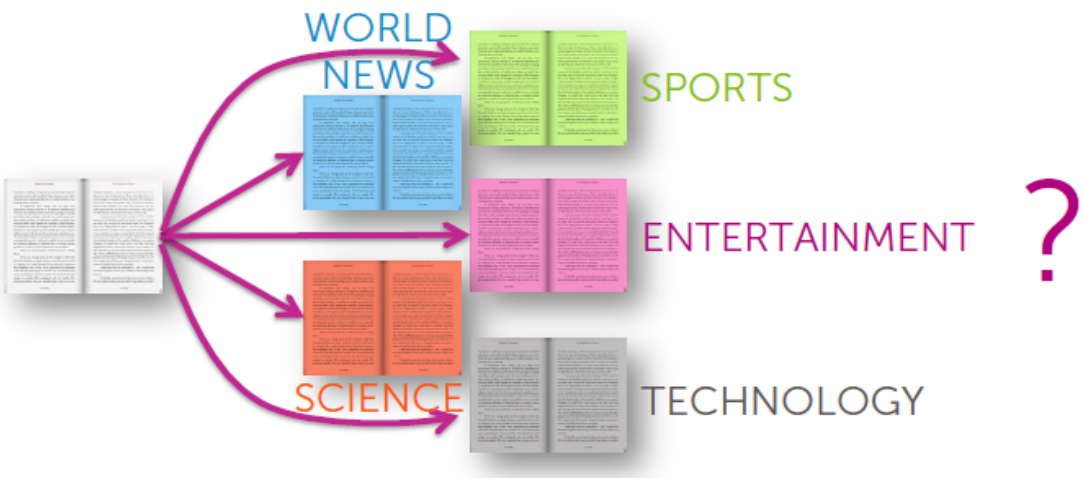
Clustering: a supervised learning

What if some of the labels are known?

Training set of labeled docs

Custering: a supervised learning

Multiclass classification problem



Example of supervised learning

Clustering: an unsupervised learning

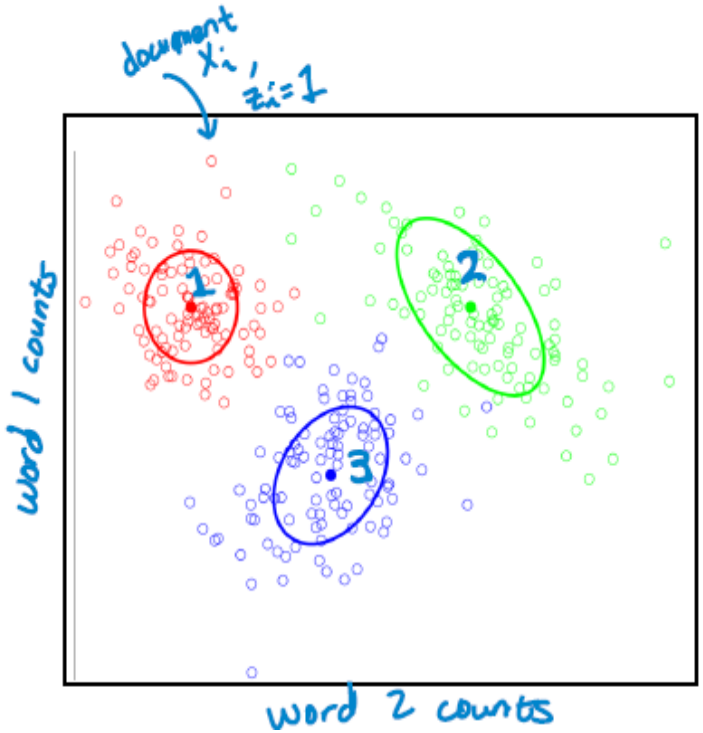
No labels provided

...uncover cluster structure from input alone

Input: docs as vectors $\mathbf{x}_i$

Output: cluster labels z_i

An unsupervised learning task

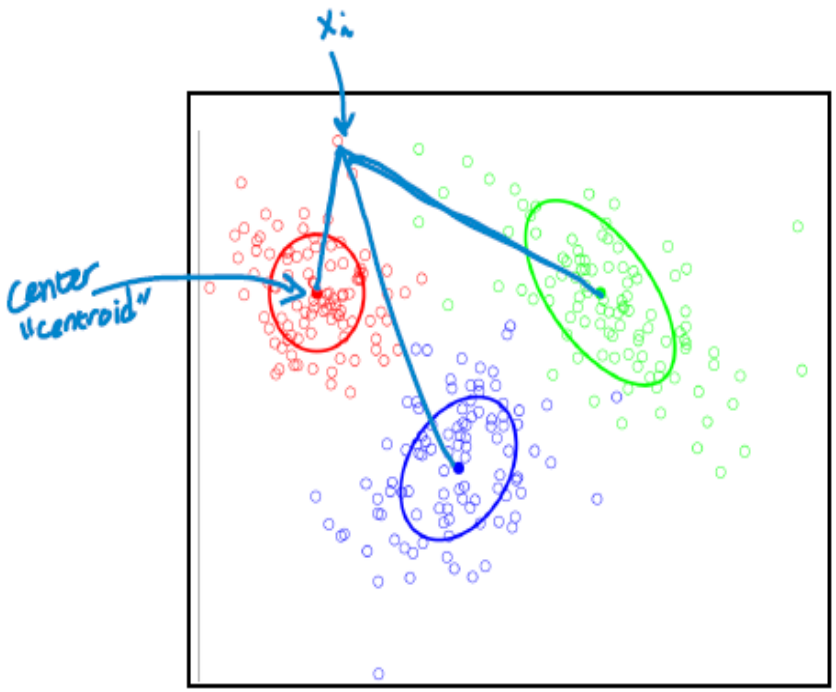


What defines a cluster ?

Cluster defined by
center & shape/spread

Assign observation $\mathbf{x}_i$ (doc) to cluster k (topic label) if

- Score under cluster k is higher than under others
- For simplicity, often define score as distance to cluster center (ignoring shape)

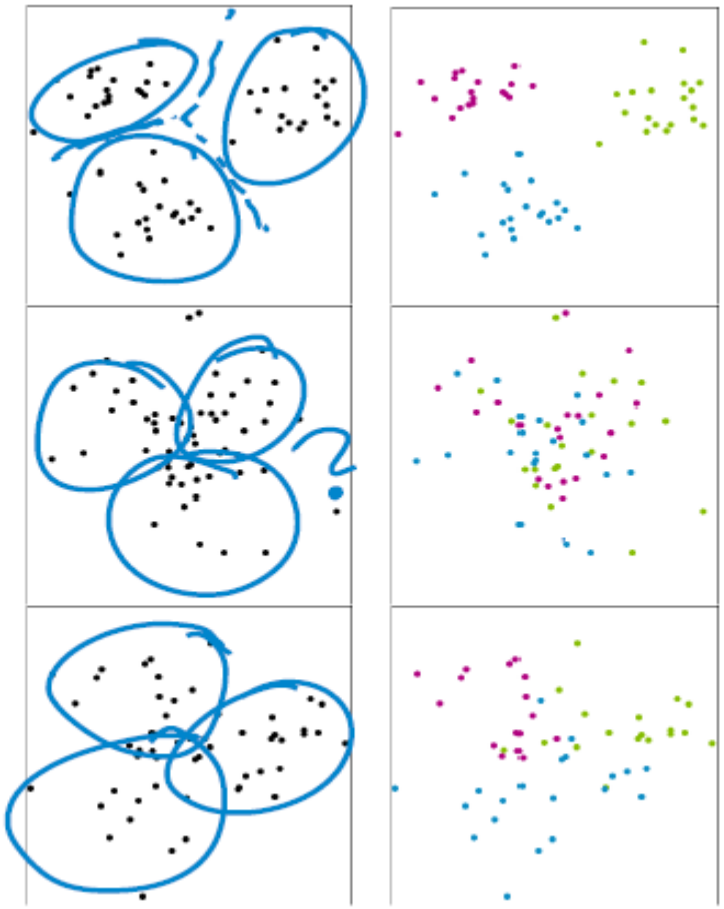


Hope for unsupervised learning

Easy

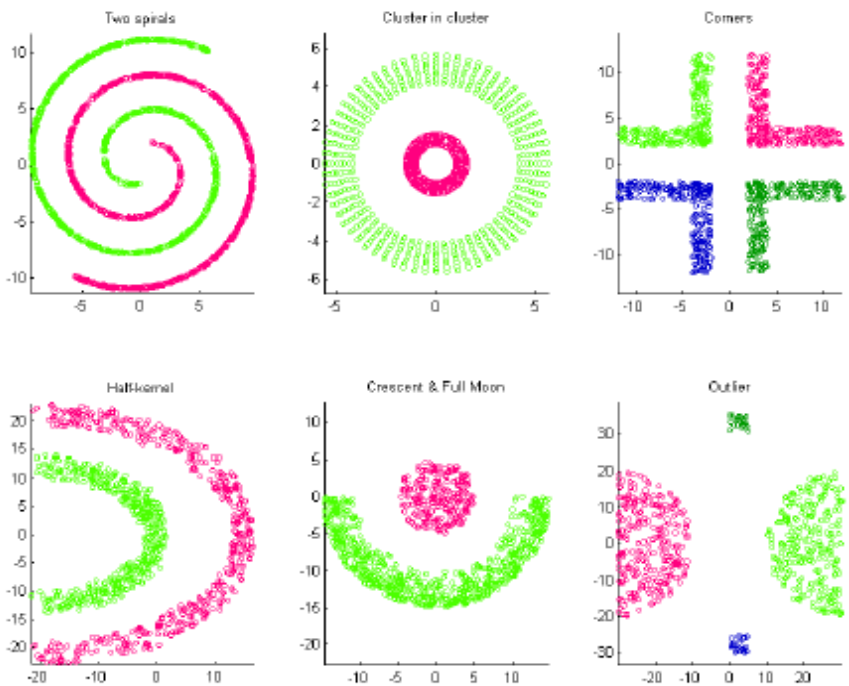
Impossible

In between



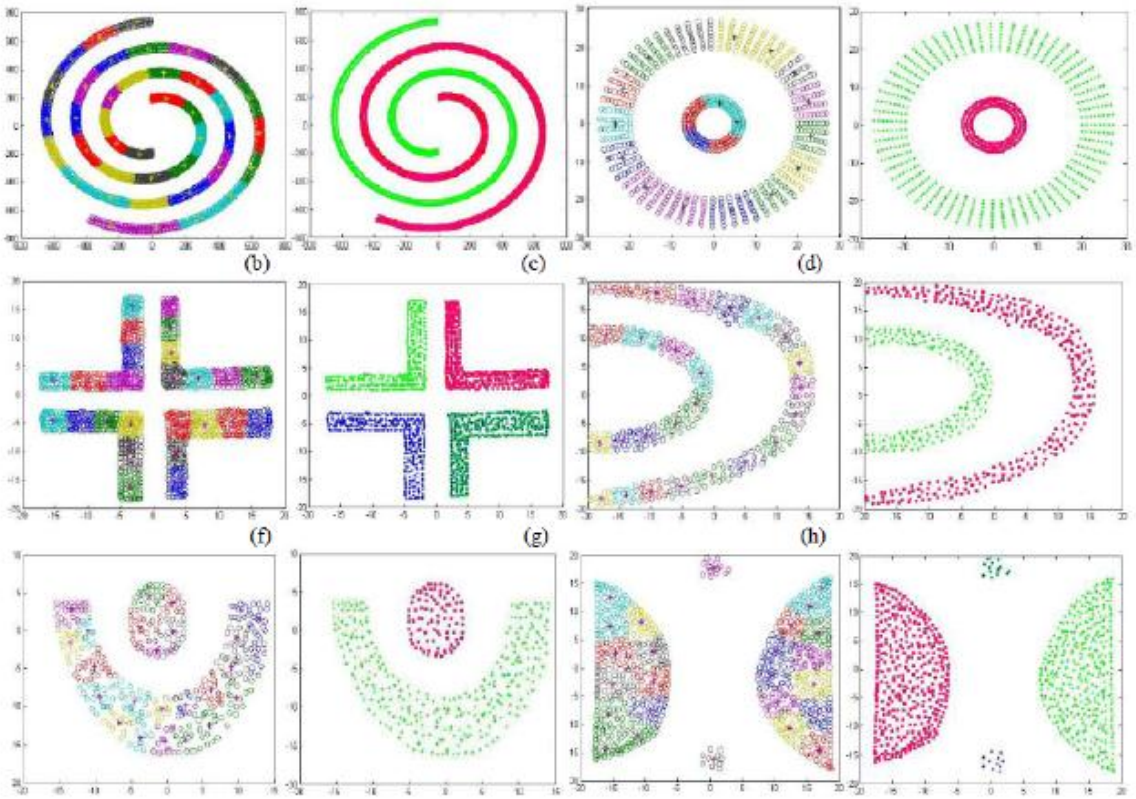
Other (challenging!) clusters to discover

Analysed by your eyes



Other (challenging!) clusters to discover

Analysed by clustering algorithms

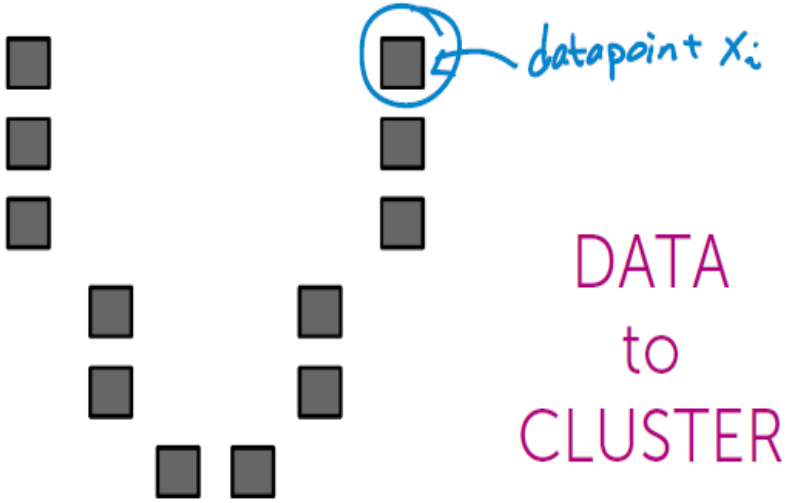


k-means clustering algorithm

k-means clustering algorithm

Assume

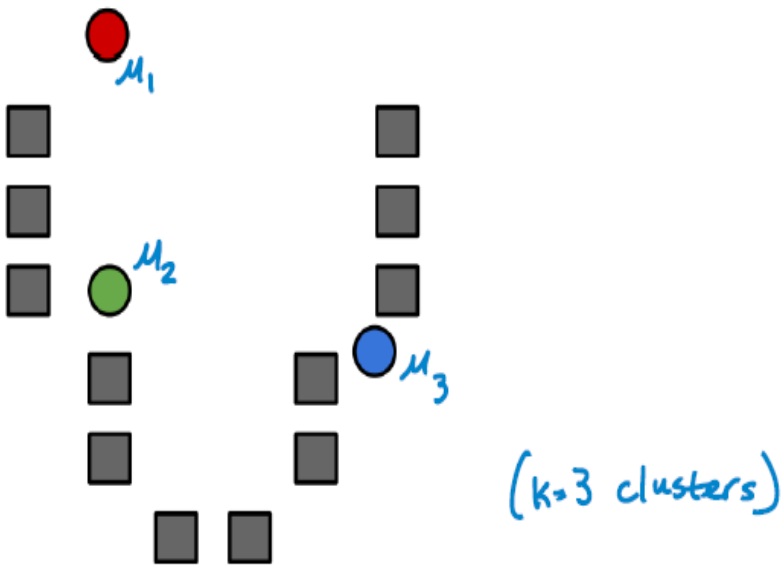
- Score= distance to cluster center (smaller better)



k-means clustering algorithm

0. Initialize cluster centers

$$\mu_1, \mu_2, \ldots, \mu_k$$



k-means clustering algorithm

0. Initialize cluster centers
1. Assign observations to closest cluster center

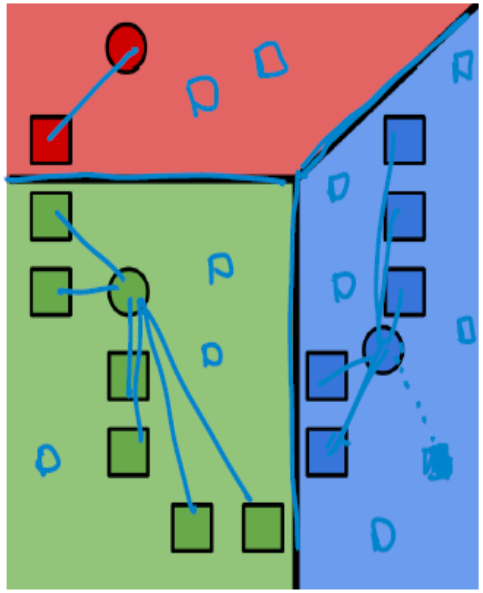
$$z_i \leftarrow \arg\min_j ||\mu_j - \mathbf{x}_i||_2^2$$

jth cluster center (varying)

ith obs. (fixed)

Inferred label for obs i, whereas supervised learning has given label y_i

return index j of the cluster whose center is closest to obs x_i (whereas min returning minimum value of $\|\cdot\|_2^2$)



Voronoi tesselation

(for visualization only ... you don't need to compute this)

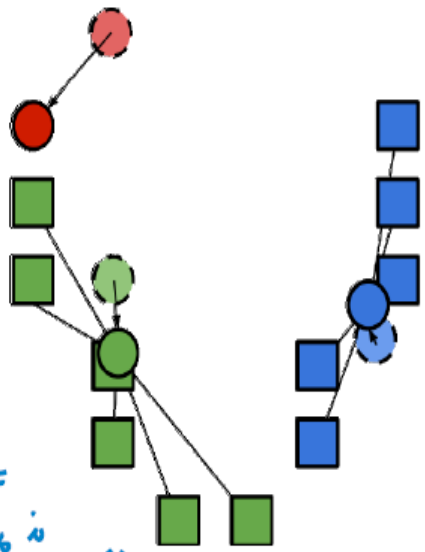
k-means clustering algorithm

0. Initialize cluster centers
1. Assign observations to closest cluster center
2. Revise cluster centers as mean of assigned observations

$$\mu_j = \frac{1}{n_j} \sum_{i:z_i=j} \mathbf{x}_i$$

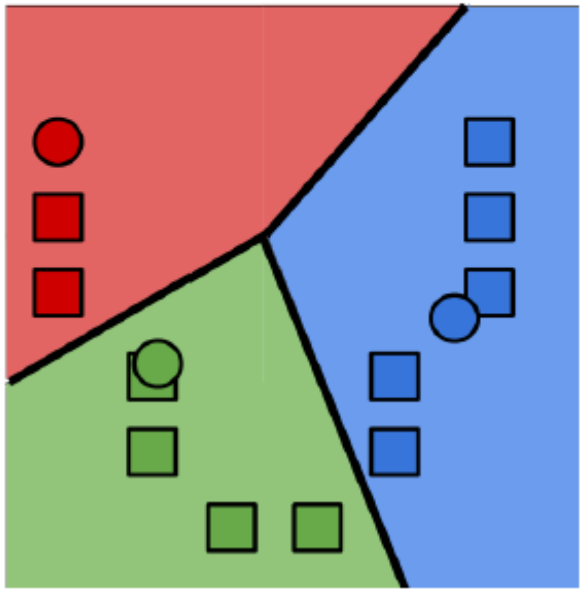
all obs. i such that $z_i = j$ (obs i is in cluster j)

# of obs. in cluster j



k-means clustering algorithm

0. Initialize cluster centers
1. Assign observations to closest cluster center
2. Revise cluster centers as mean of assigned observations
3. Repeat 1.+2. until convergence



k-means as coordinate descent algorithm

1. Assign observations to closest cluster center

$$z_i \leftarrow \arg\min_j ||\mu_j - \mathbf{x}_i||_2^2$$

2. Revise cluster centers as mean of assigned observations

$$\mu_j = \frac{1}{n_j} \sum_{i:z_i=j} \mathbf{x}_i$$

equivalent to

$$\mu_j \leftarrow \arg\min_\mu \sum_{i:z_i=j} ||\mu - \mathbf{x}_i||_2^2$$

K-means as coordinate descent algorithm

1. Assign observations to closest cluster center

$$z_i \leftarrow \arg\min_j ||\mu_j - \mathbf{x}_i||_2^2$$

2. Revise cluster centers as mean of assigned observations

$$\mu_j \leftarrow \arg\min_\mu \sum_{i:z_i=j} ||\mu - \mathbf{x}_i||_2^2$$

Alternating minimization
1. (z given μ) and 2. (μ given z)
= **coordinate descent**

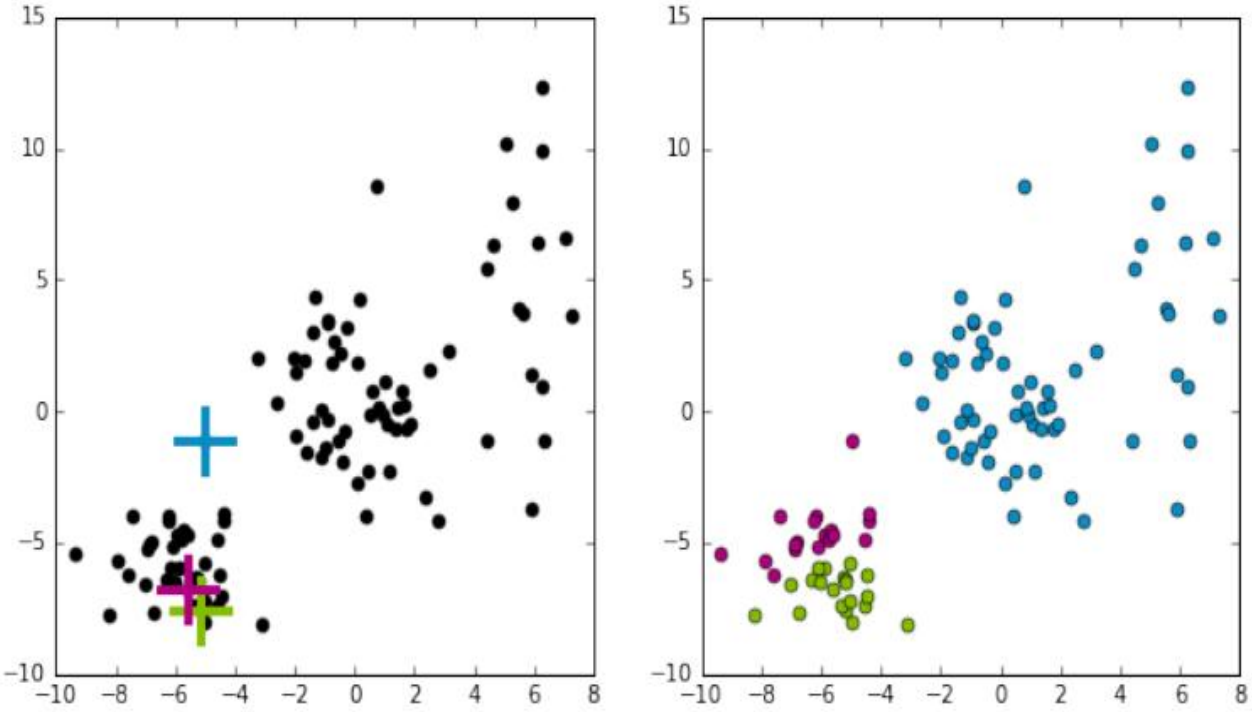
Convergence of k-means

Converges to:

- ~~Global optimum~~
- Local optimum
- ~~neither~~

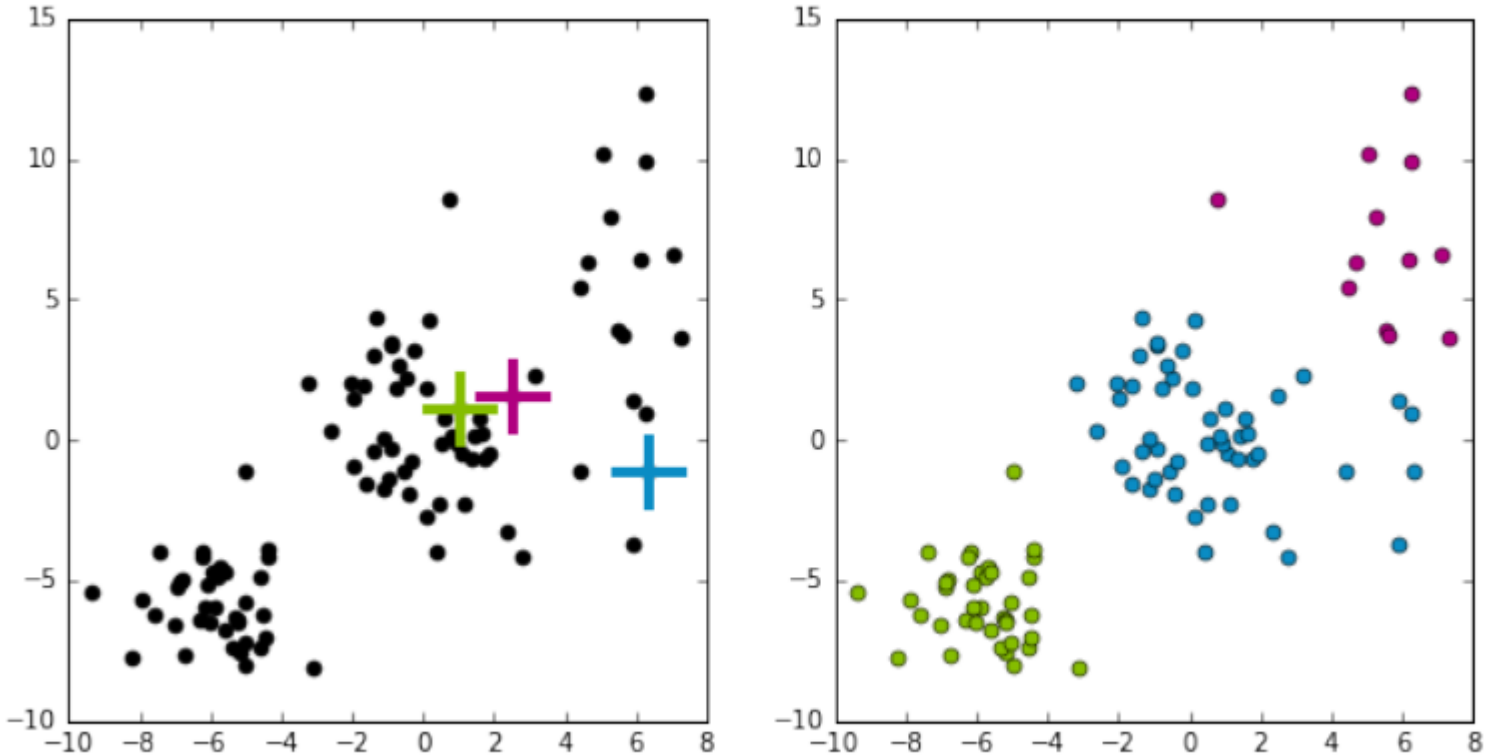
Because we can cast k-means as coordinate descent algorithm we know that we are converging to local optimum

Convergence of k-mans to local mode



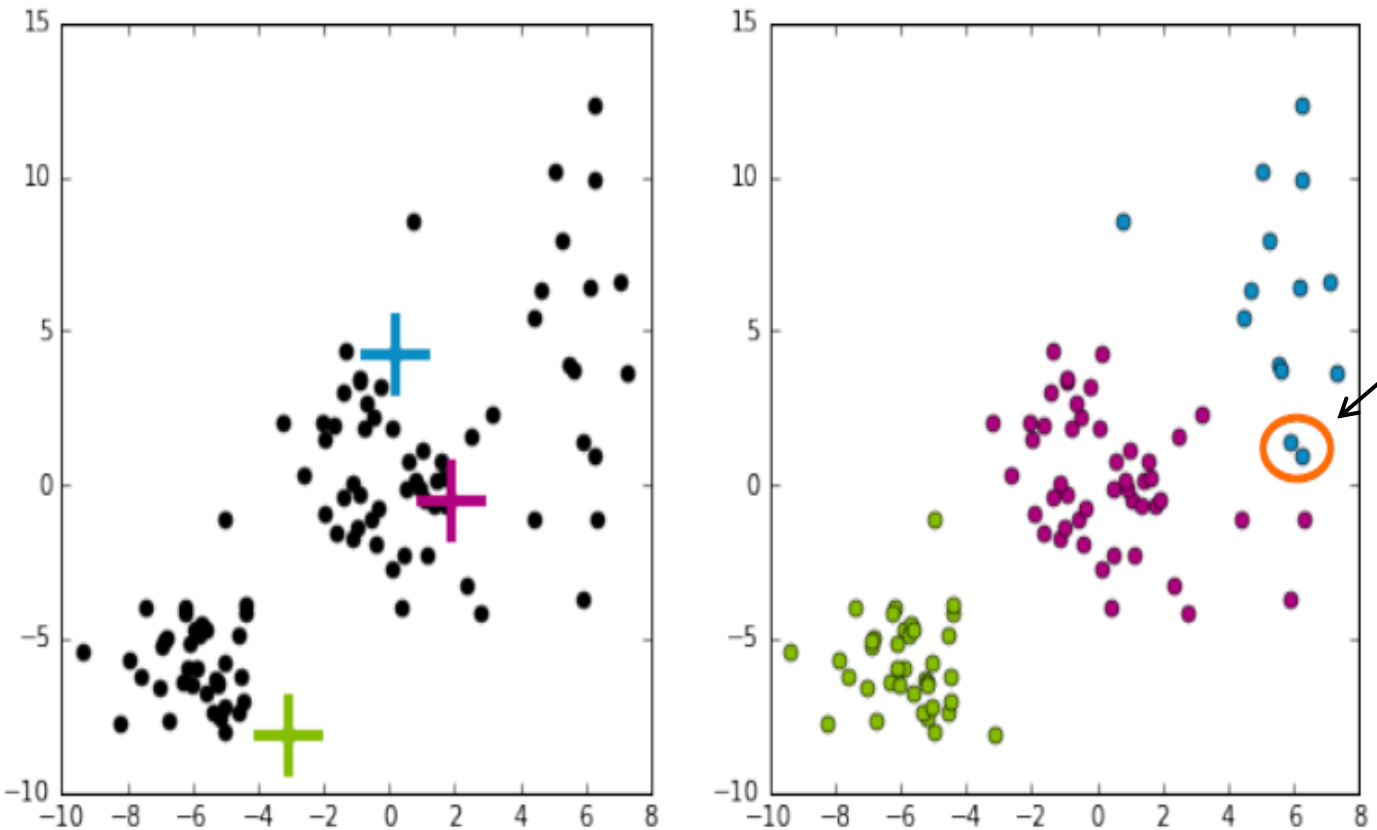
Crosses: initialised centers

Convergence of k-mans to local mode



Crosses: initialised centers

Convergence of k-mans to local mode



Assigment to which group has changed

k-means very sensitive to initiased centers

Crosses: initialised centers

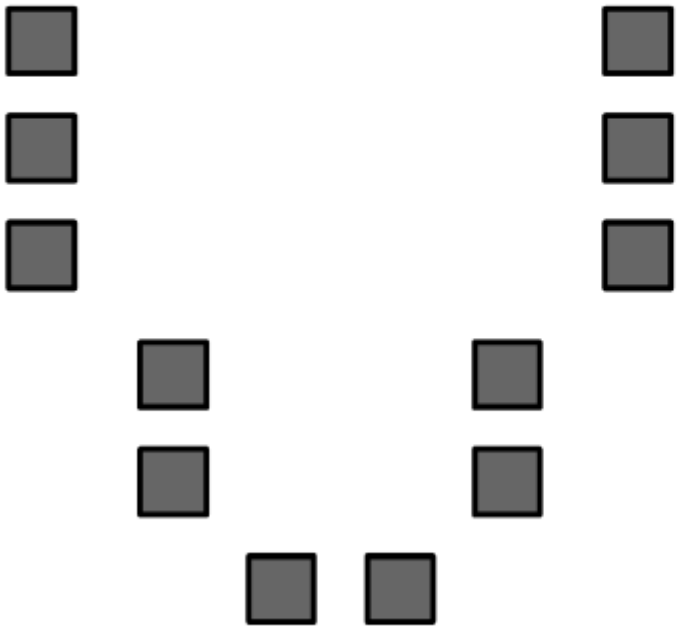
Smart initialisation: k-means++ overwiew

Initialization of k-means algorithm is critical to quality of local optima found

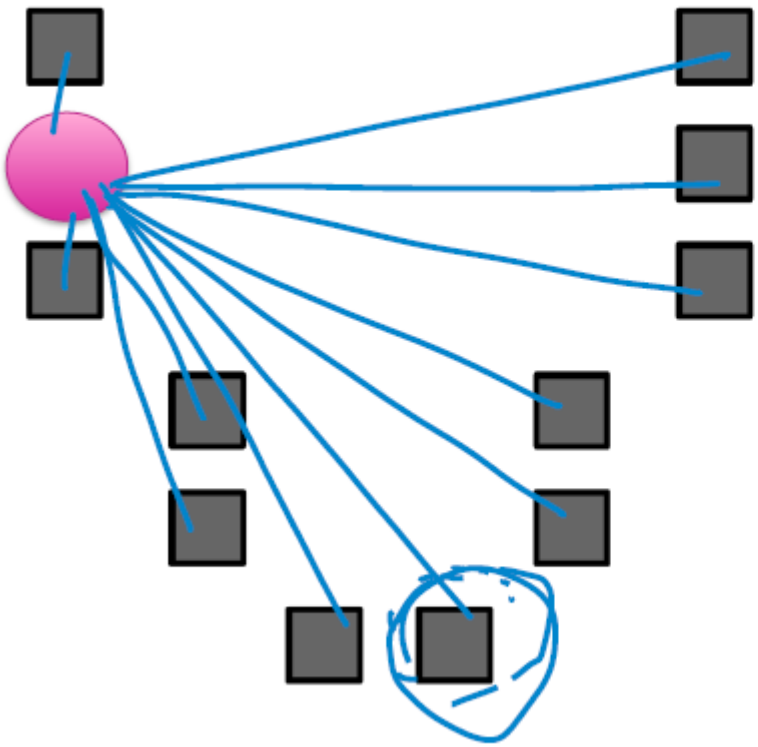
Smart initialization:

1. Choose first cluster center uniformly at random from data points
2. For each obs **x**, compute distance d(**x**) to nearest cluster center
3. Choose new cluster center from amongst data points, with probability of **x** being chosen proportional to $d(\mathbf{x})^2$
4. Repeat Steps 2 and 3 until k centers have been chosen

k-means++ visualised

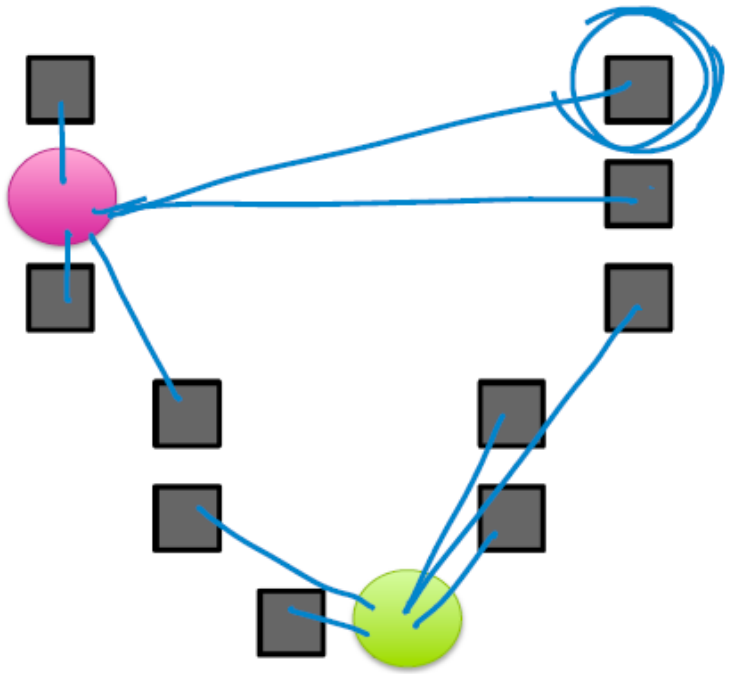


k-means++ visualised

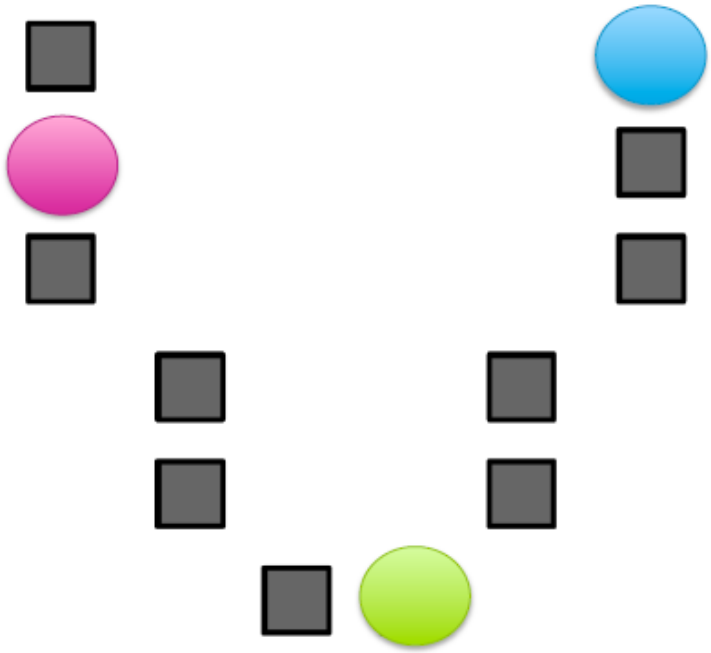


more likely to select a datapoint as a cluster center if that datapoint is far away (dist^2 increases this effect)

k-means++ visualised



k-means++ visualised



Smart initialisation: k-means++ overwiew

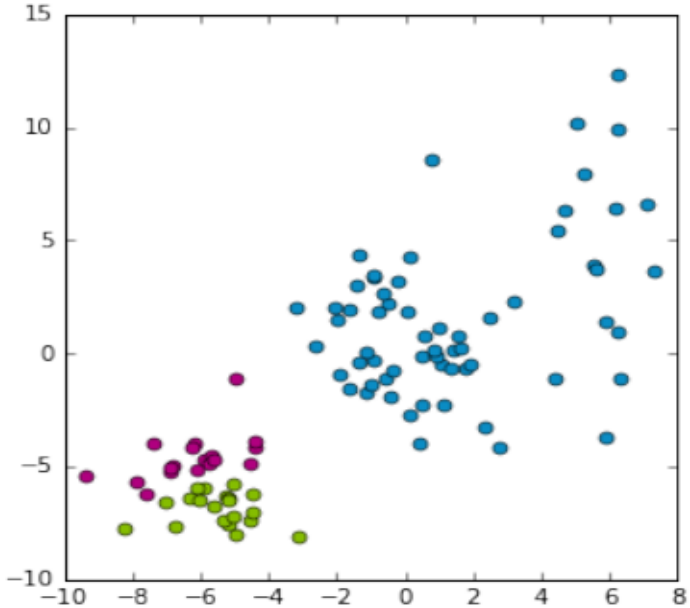
k-means++ pros/cons

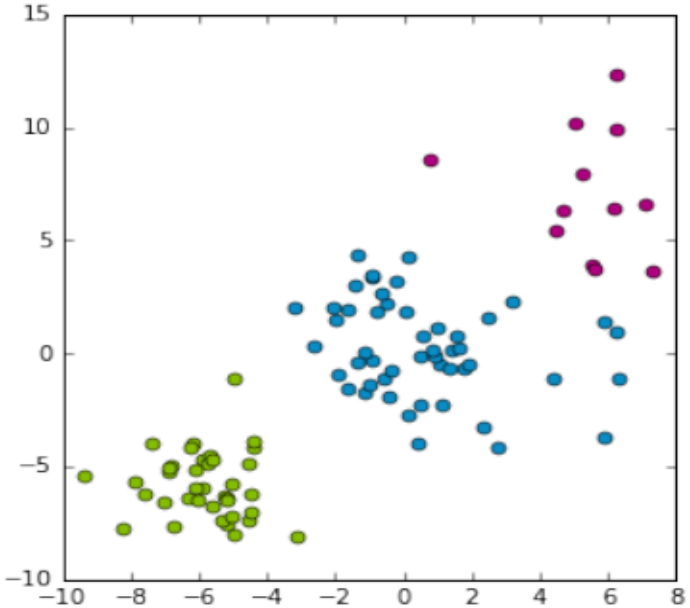
Computationally costly relative to random initialization, but the subsequent k-means often converges more rapidly

Tends to improve quality of local optimum and lower runtime

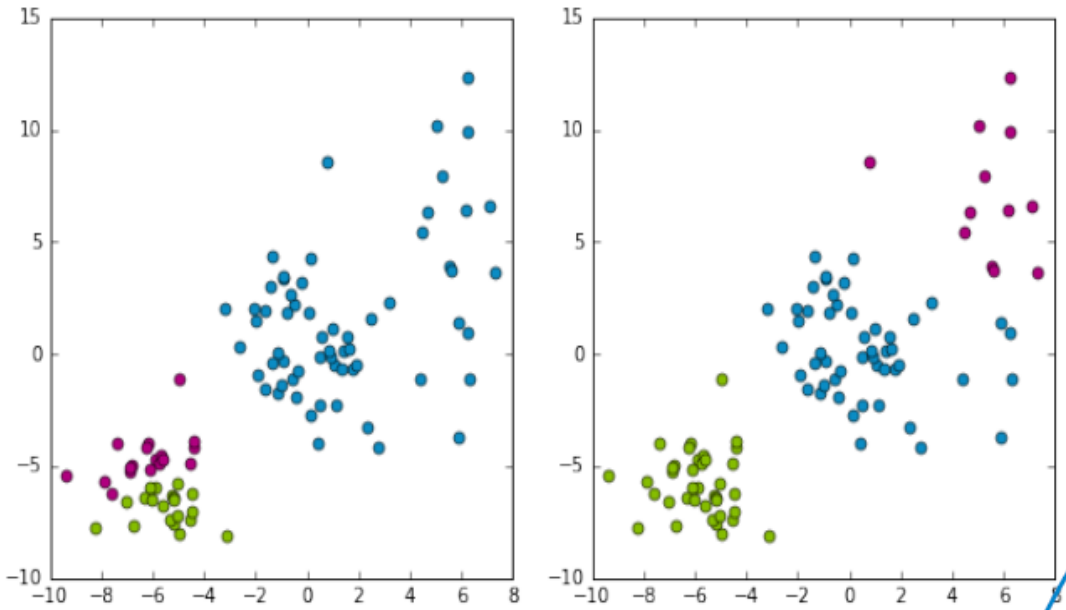
Assessing quality of the clustering

Which clustering do I prefer?





k-means objective



k-means is trying to minimize the sum of squared distances:

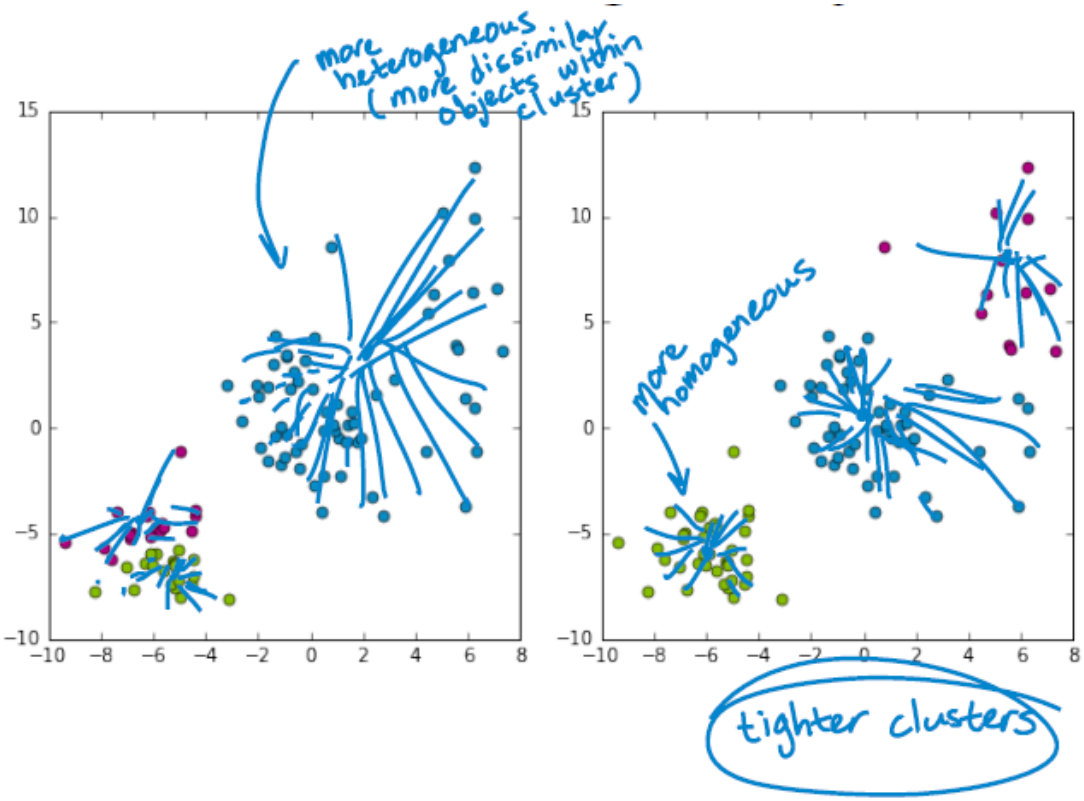
sum over all clusters

sum of squared distances in cluster j

$$\sum_{j=1}^{k} \sum_{i:z_i=j} ||\mu_j - \mathbf{x}_i||_2^2$$

$\min_{\{z_i\},\{\mu_j\}}$

Cluster heterogeneity



Measure of quality of given clustering:

$$\sum_{j=1}^{k} \sum_{i:z_i=j} ||\mu_j - \mathbf{x}_i||_2^2$$

Lower is better!

What happens to heterogeneity as k increases?

Can refine clusters more and more to the data

→ overfitting!

# of observations

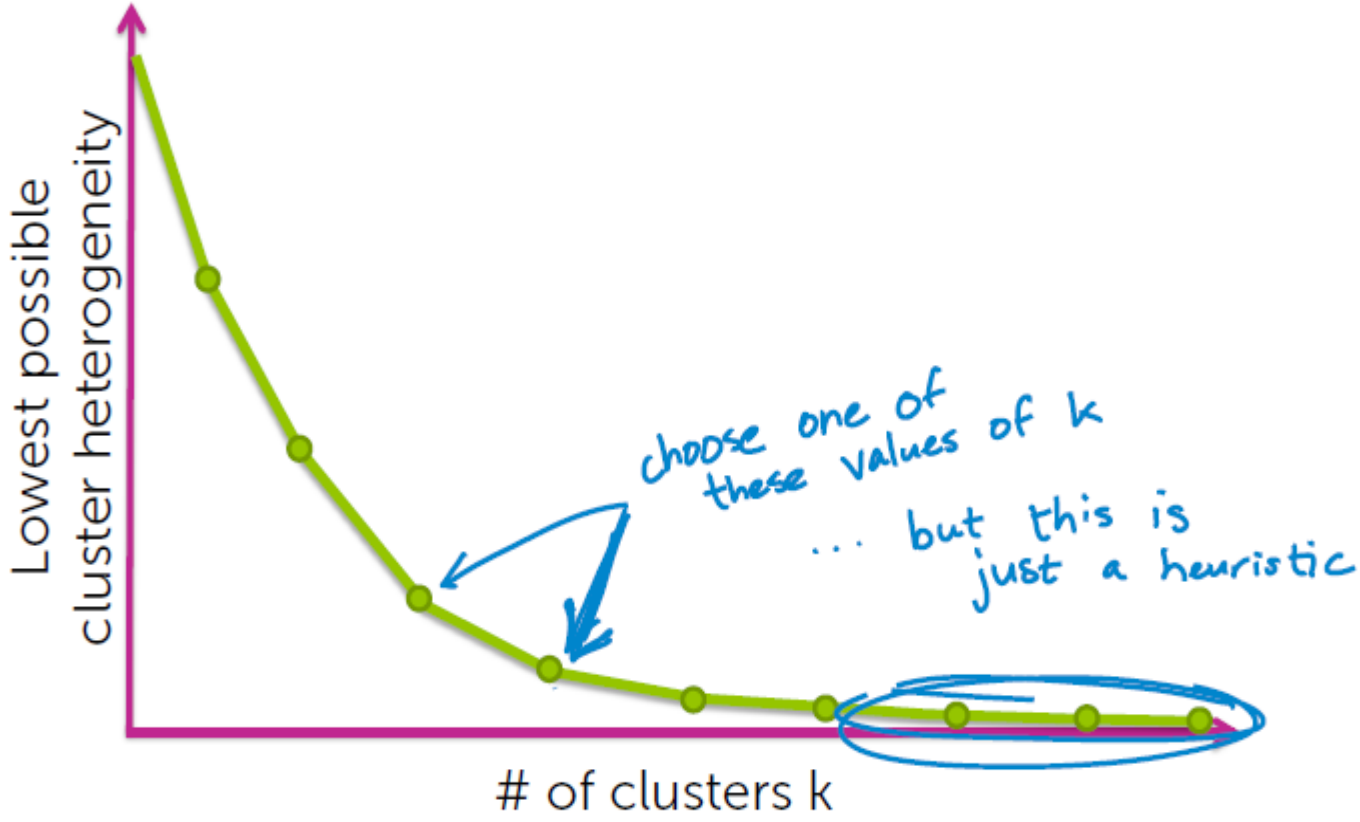
Extreme case of k=N:

- can set each cluster center equal to datapoint
- heterogeneity = 0 !

(all distances to cluster centers are 0)

Lowest possible cluster heterogeneity decreases with increasing k

How to choose k?



What you can do now …

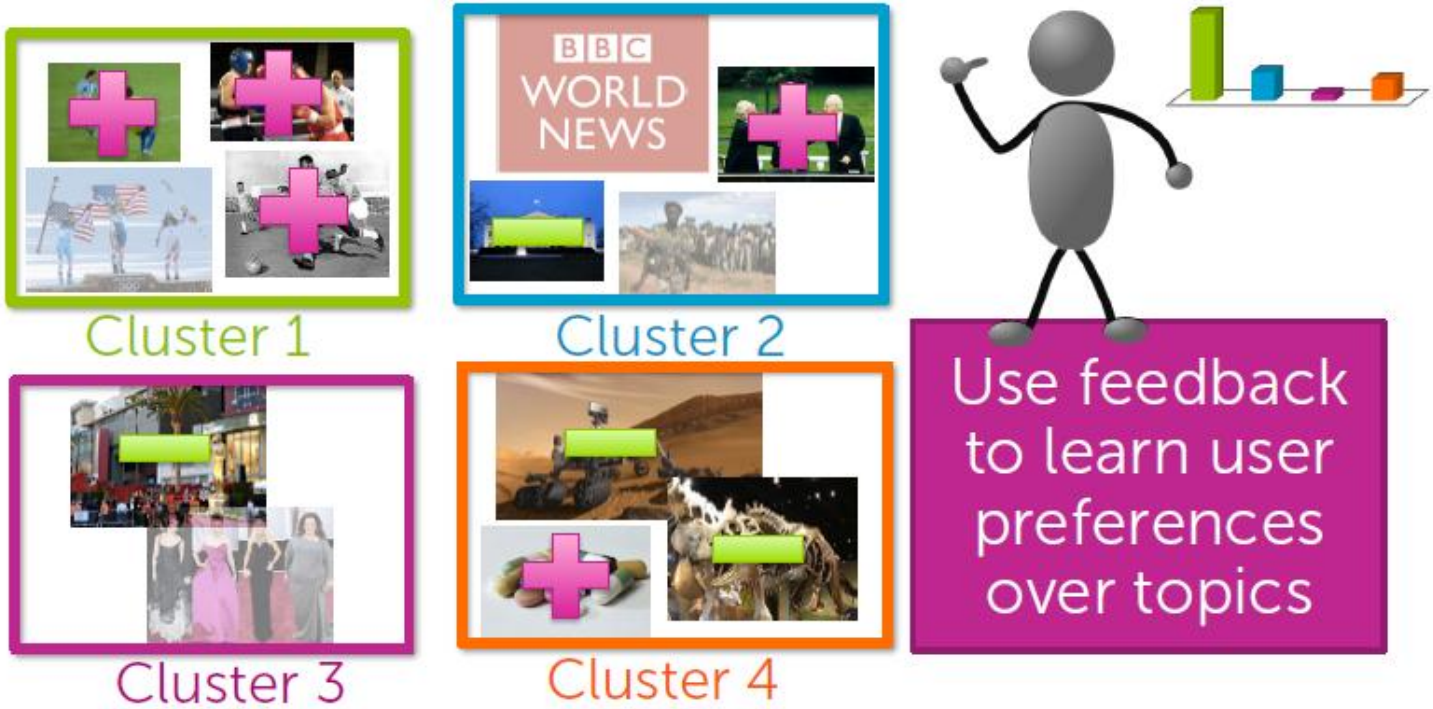
- Describe potential applications of clustering
- Describe the input (unlabeled observations) and output (labels) of a clustering algorithm
- Determine whether a task is supervised or unsupervised
- Cluster documents using k-means
- Interpret k-means as a coordinate descent algorithm
- Define data parallel problems
- Explain Map and Reduce steps of MapReduce framework
- Use existing MapReduce implementations to parallelize k-means, understanding what's being done under the hood

Probabilistic approach: mixture model

Why probabilistic approach?

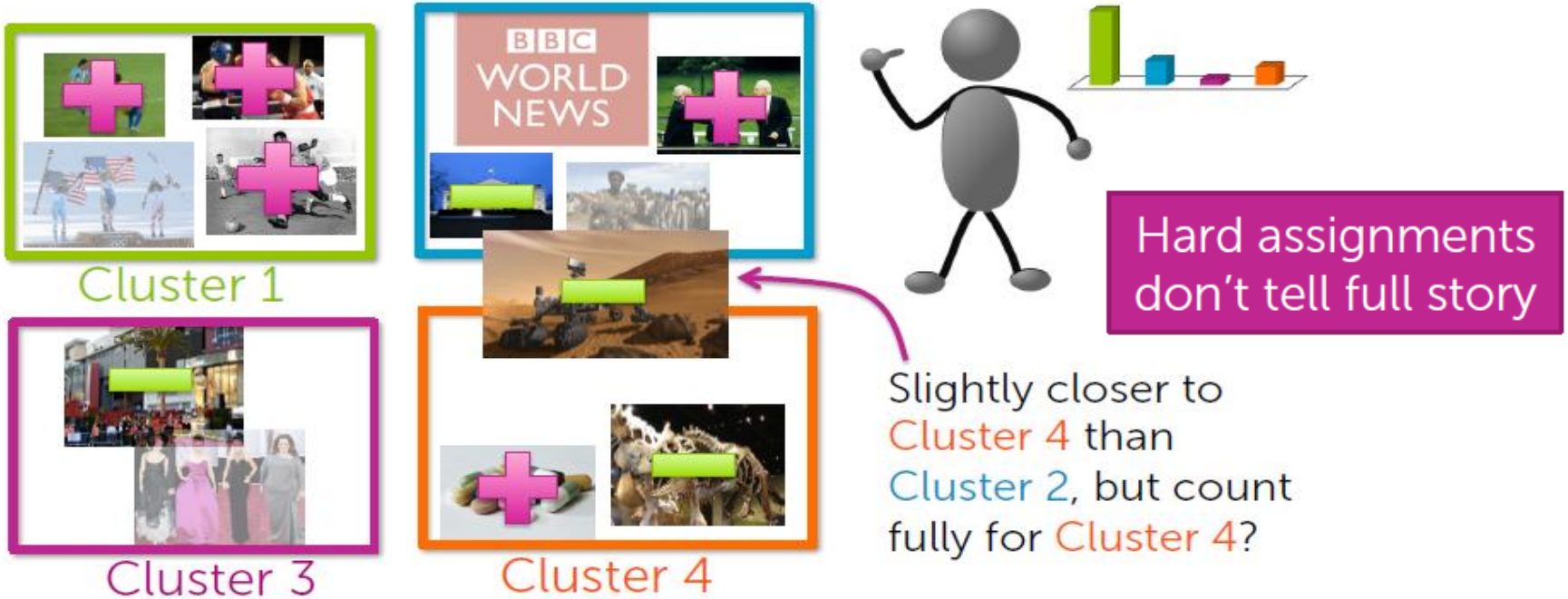
Learn user preferences

Set of clustered documents read by user



Why probabilistic approach?

Uncertainty in cluster assignments



Why probabilistic approach?

Other limitations of k-means

Assign observations to closest cluster center

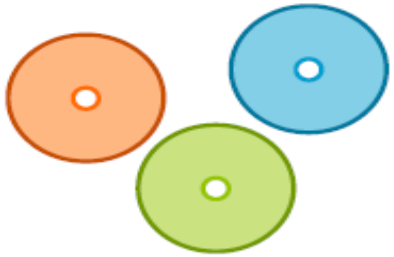
$$z_i \leftarrow \arg\min_j ||\mu_j - \mathbf{x}_i||_2^2$$

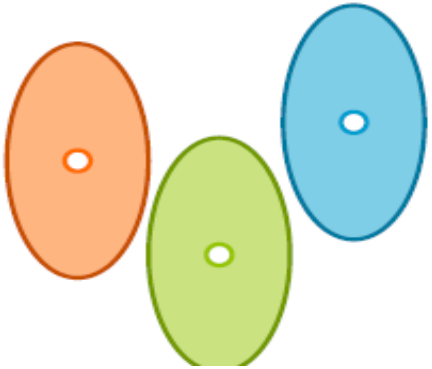
Can use weighted Euclidean, but requires *known* weights

Only center matters

Still assumes all clusters have the same axis-aligned ellipses

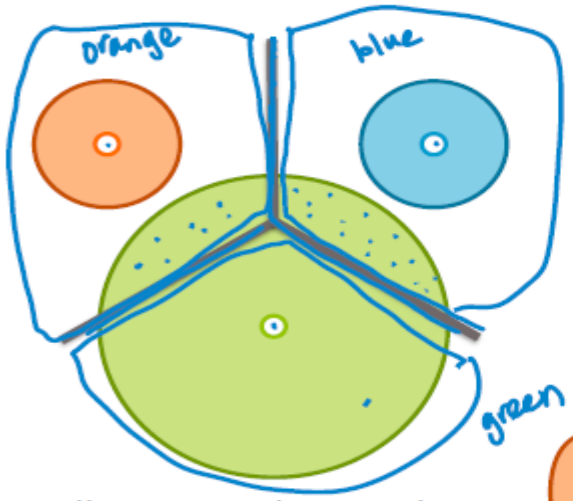
Equivalent to assuming spherically symmetric clusters



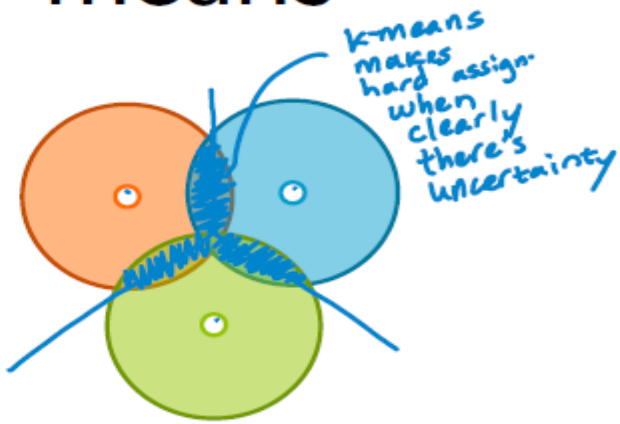


Why probabilistic approach?

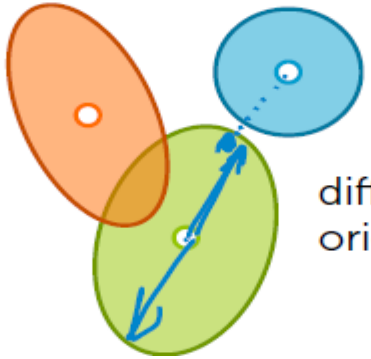
Failure modes of k-means



disparate cluster sizes



overlapping clusters



different shaped/
oriented clusters

Mixture models

- Provides **soft assignments** of observations to clusters (uncertainty in assignment)
  - e.g., 54% chance document is world news, 45% science, 1% sports, and 0% entertainment
- Accounts for cluster shapes not just centers
- Enables learning weightings of dimensions
  - e.g., how much to weight each word in the vocabulary when computing cluster assignment

Application: clustering images

Discover groups of similar images

- Ocean
- Pink flower
- Dog
- Sunset
- Clouds
- ...

Application: clustering images

Simple image representation

Consider average red, green, blue pixel intensities

[R = 0.05, G = 0.7, B = 0.9]

[R = 0.85, G = 0.05, B = 0.35]

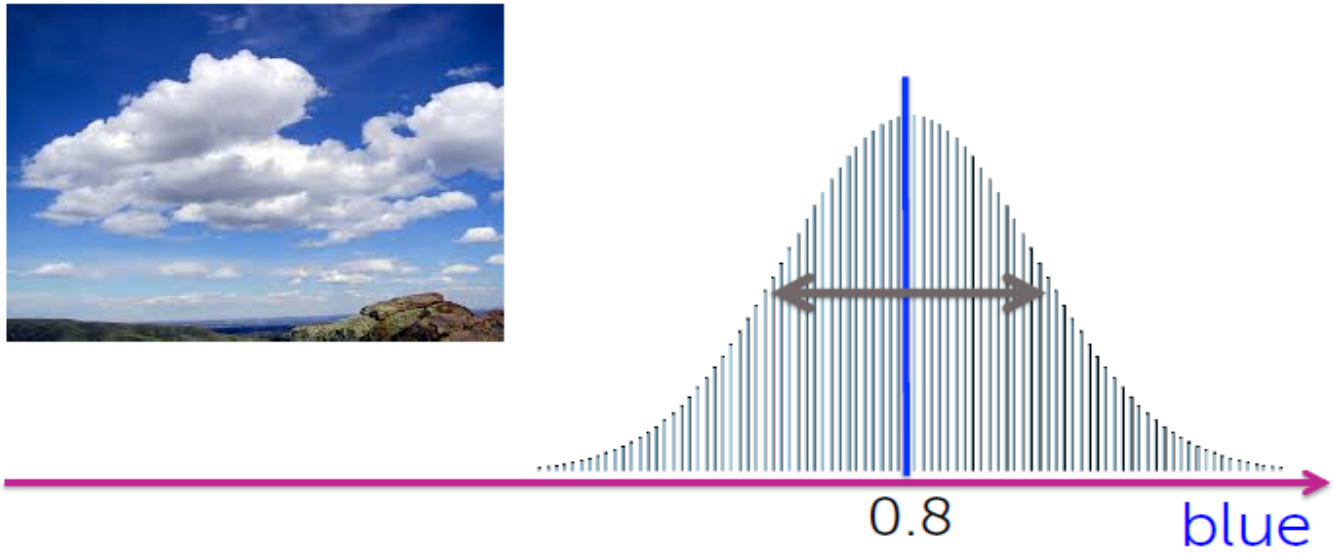
[R = 0.02, G = 0.95, B = 0.4]

Single RGB vector per image

Application: clustering images

Distribution over all cloud images

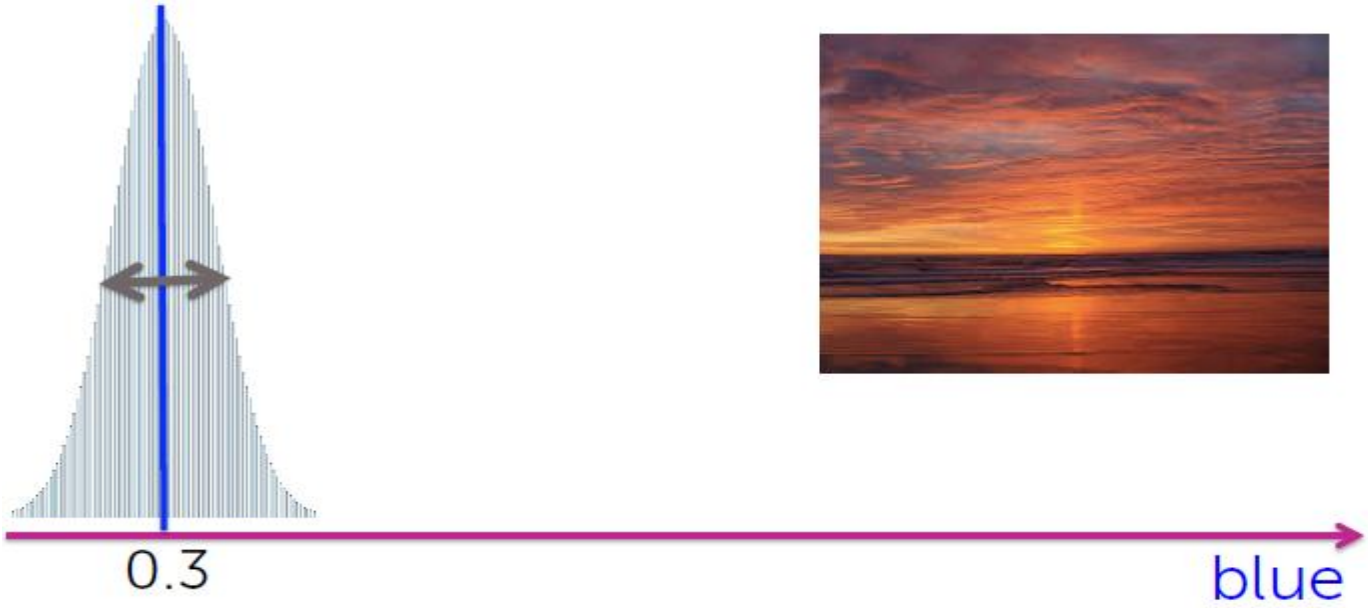
Let's look at just the blue dimension



Application: clustering images

Distribution over all sunset images

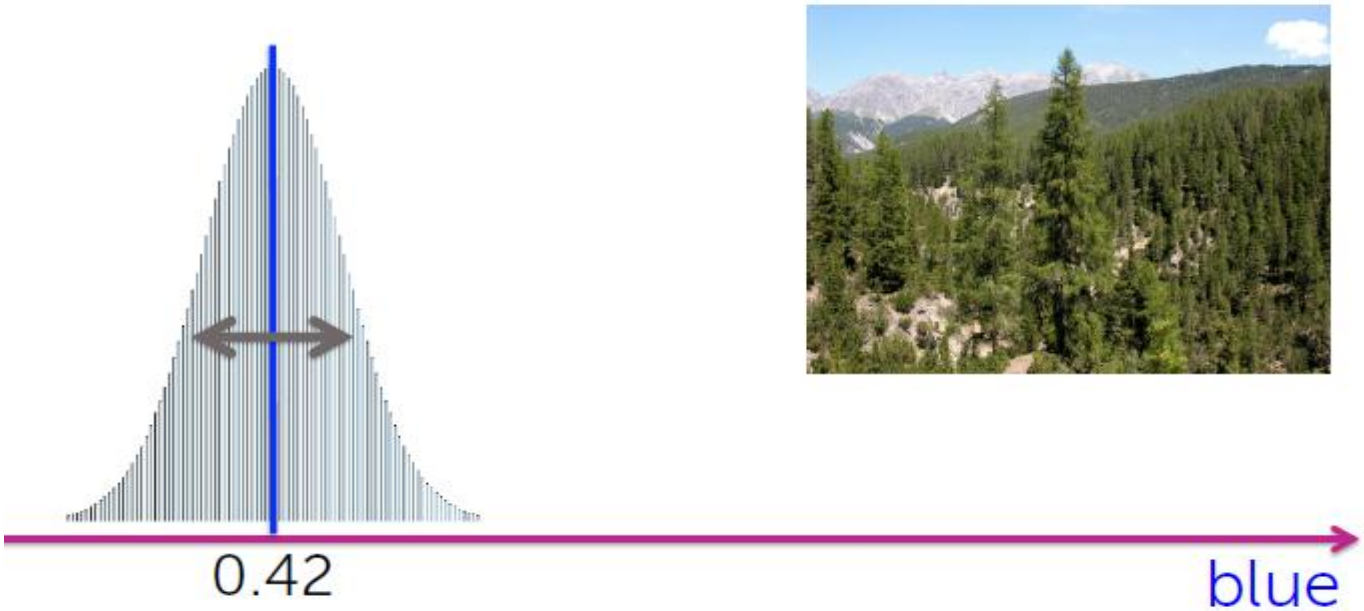
Let's look at just the blue dimension



Application: clustering images

Distribution over all forest images

Let's look at just the blue dimension



Application: clustering images

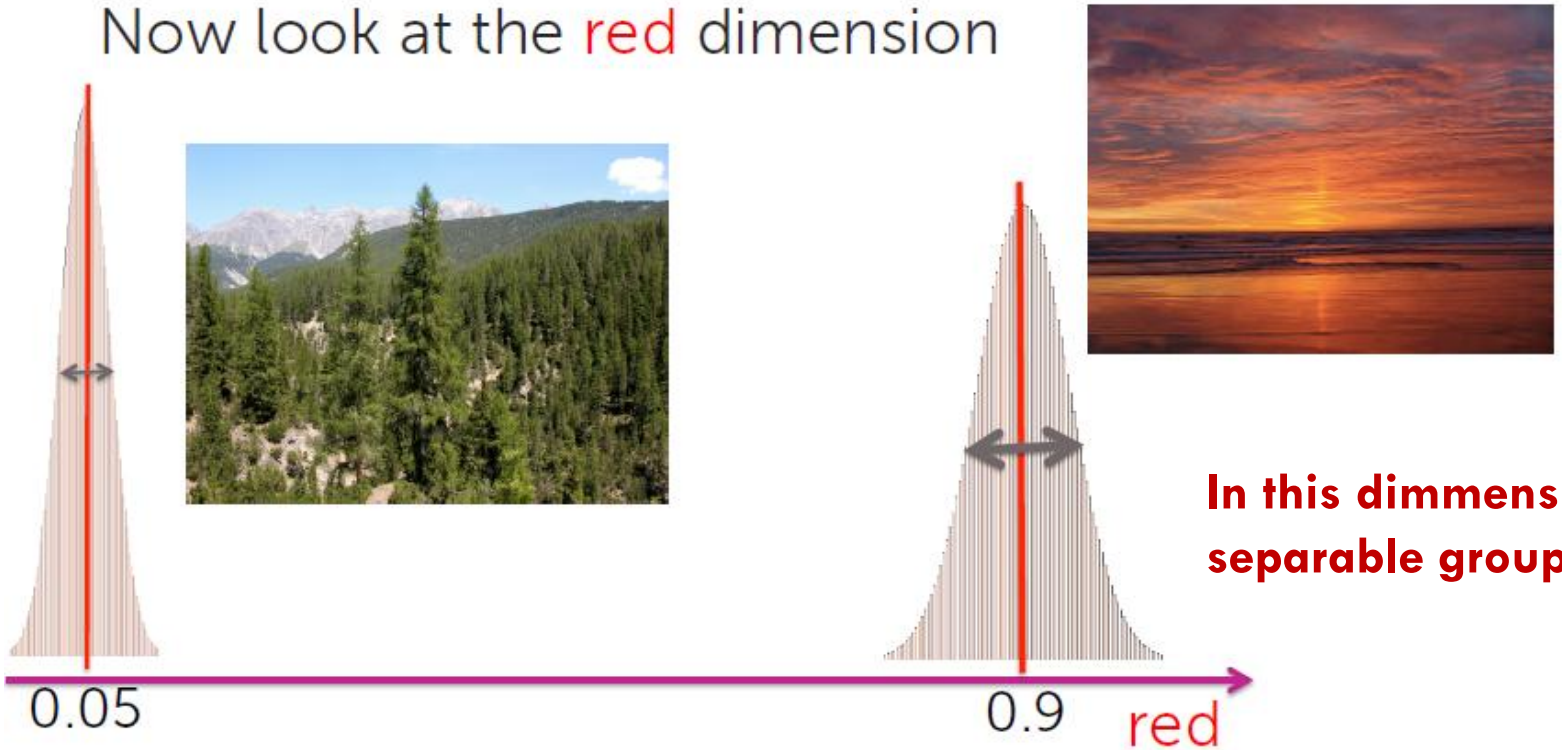
Distribution over all images

We see that they are grouping!
But not easy to distinguish between groups

Application: clustering images

Can be distinguished along other dim

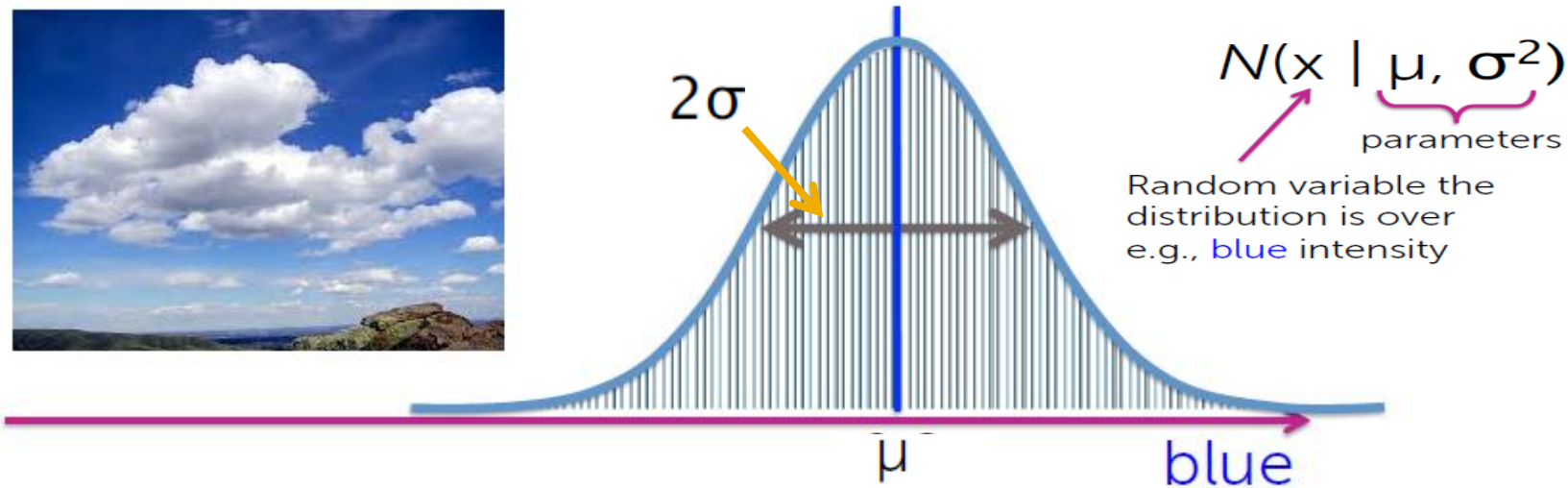
Now look at the red dimension



In this dimmension separable groups!

Model for a given image type

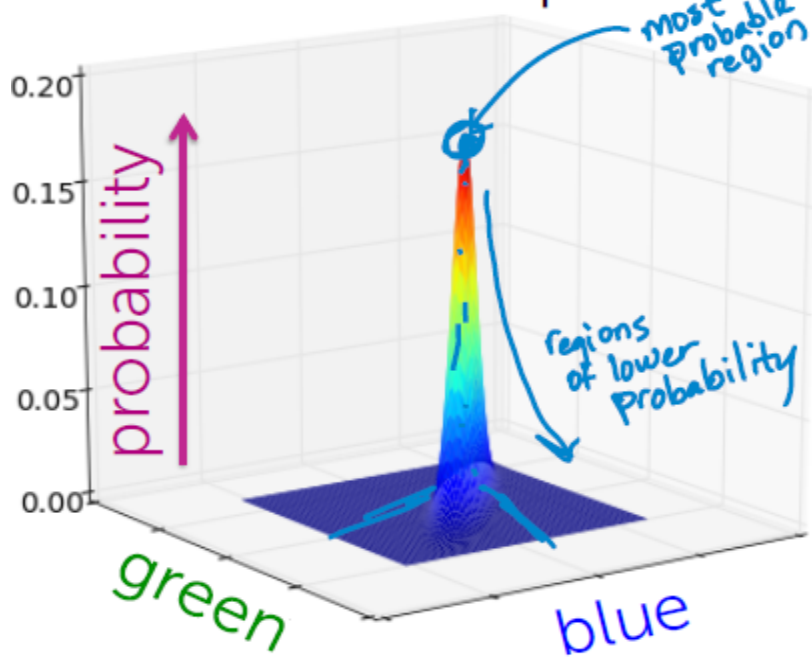
For **each dimension** of the [R, G, B] vector, and **each image type**, assume a Gaussian distribution over color intensity



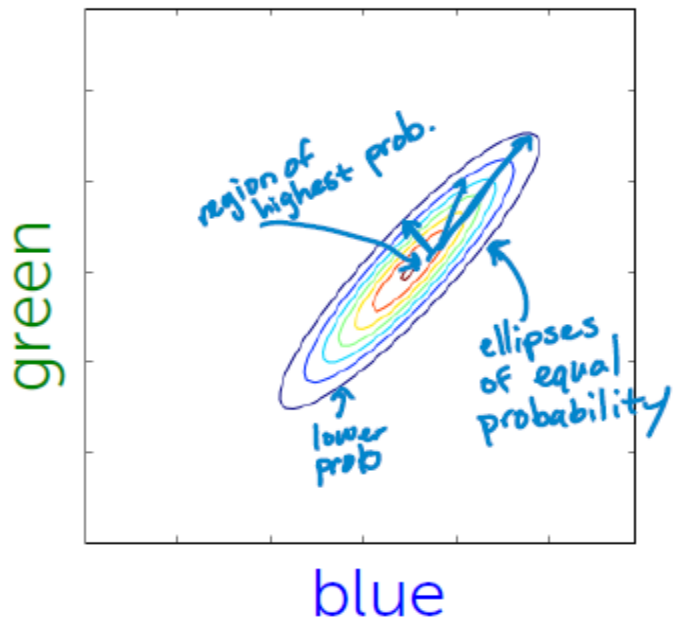
Model for a given image type

2D Gaussians – Bird's eye view

3D mesh plot



Contour plot



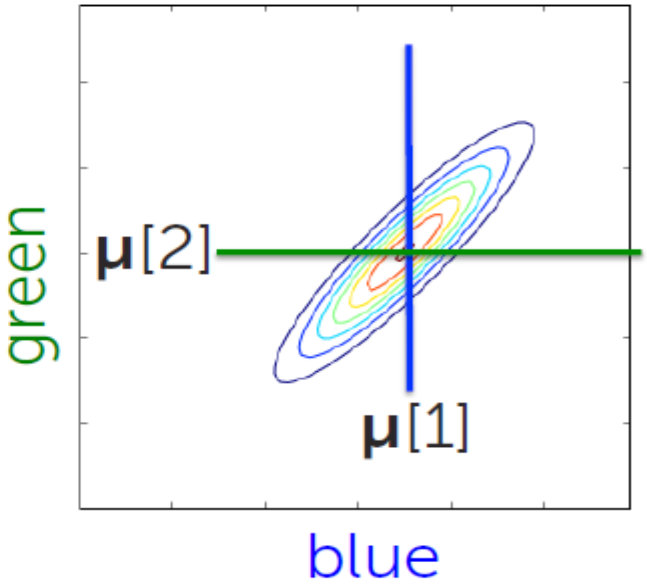
Application: clustering images

2D Gaussians – Parameters

Fully specified by **mean μ** and **covariance Σ**

$\boldsymbol{\mu} = [\mu_{blue}, \mu_{green}]$

mean centers the distribution in 2D



Application: clustering images

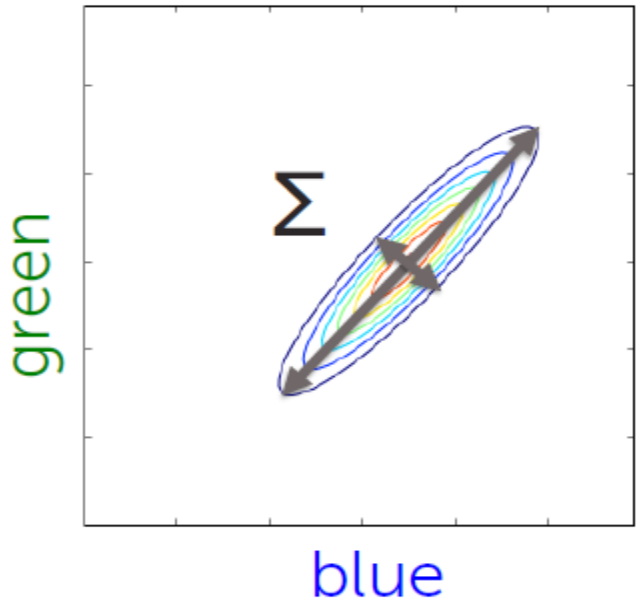
2D Gaussians – Parameters

Fully specified by **mean** μ and **covariance** Σ

$$\boldsymbol{\mu} = [\mu_{blue}, \mu_{green}]$$

$$\Sigma = \begin{pmatrix} \sigma_{blue}^2 & \sigma_{blue,green} \\ \sigma_{green,blue} & \sigma_{green}^2 \end{pmatrix}$$

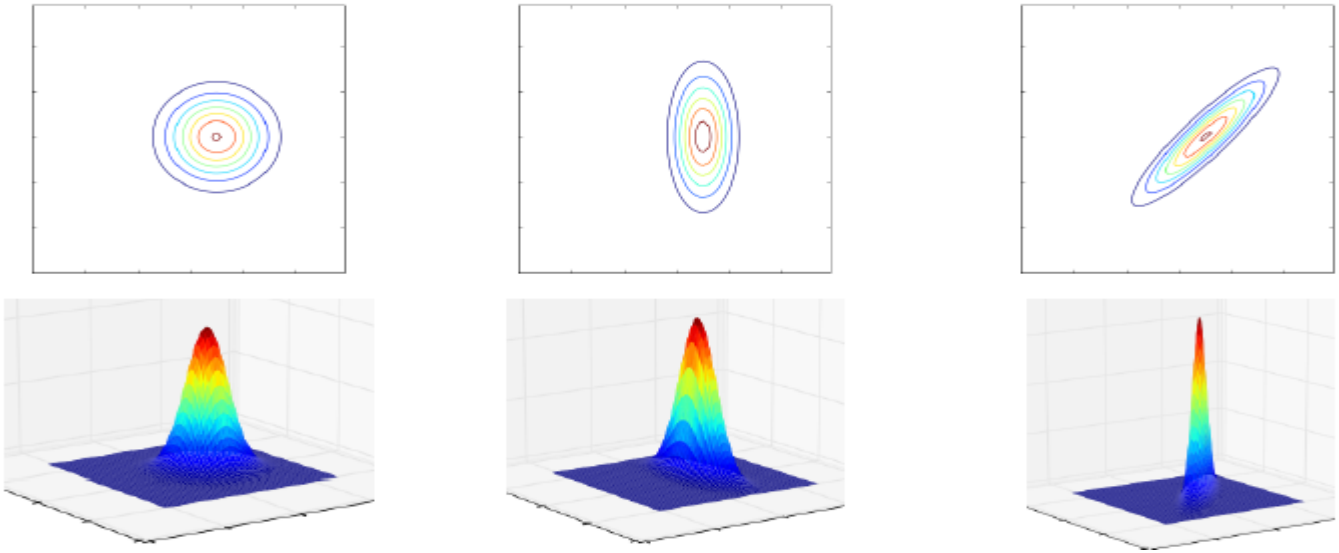
covariance determines orientation + spread



Application: clustering images

Covariance structures

$$\Sigma = \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \sigma_B{}^2 & 0 \\ 0 & \sigma_G{}^2 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \sigma_B{}^2 & \sigma_{B,G} \\ \sigma_{G,B} & \sigma_G{}^2 \end{pmatrix}$$

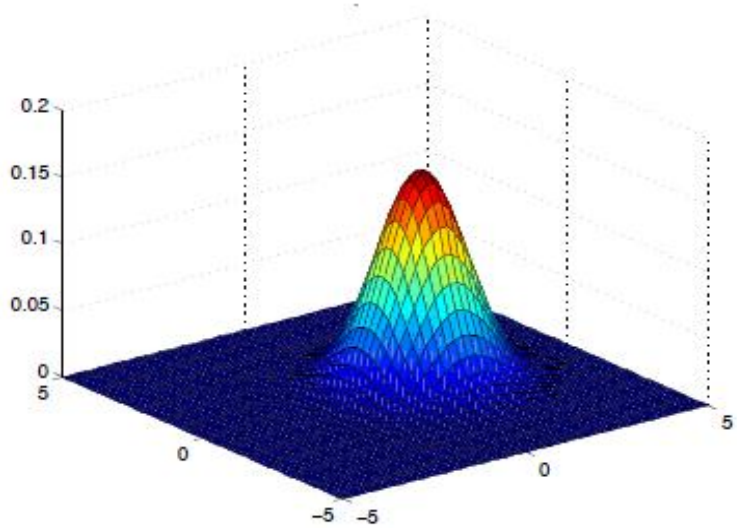


Application: clustering images

Notating a multivariate Gaussian

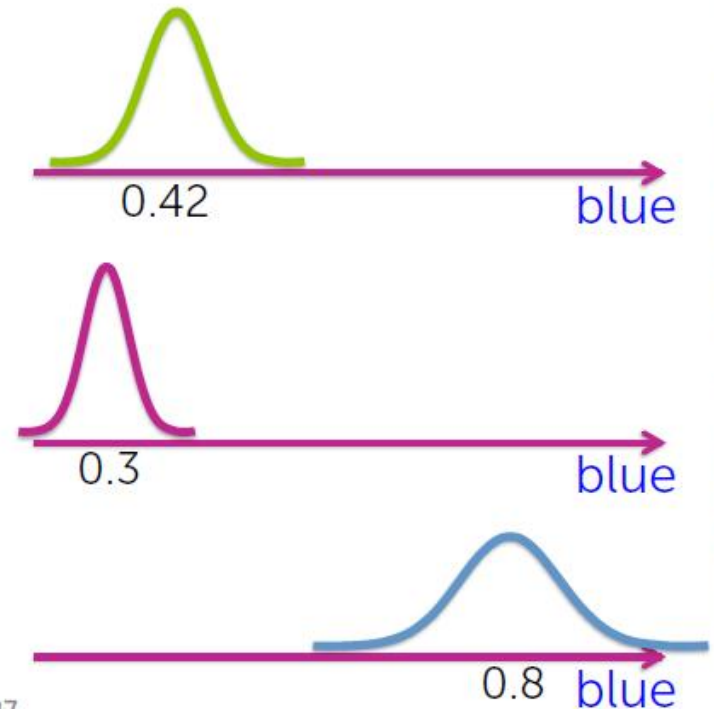
$$N(\mathbf{x} \mid \underbrace{\boldsymbol{\mu}, \Sigma}_{\text{parameters}})$$

Random vector
e.g., [R, G, B] intensities



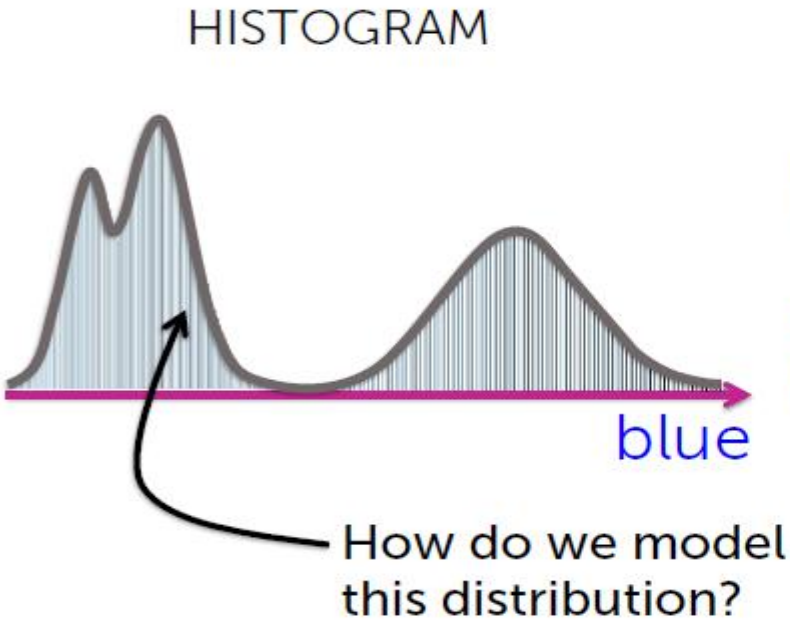
Mixture of Gaussians

Model as Gaussian per category/cluster



Mixture of Gaussians

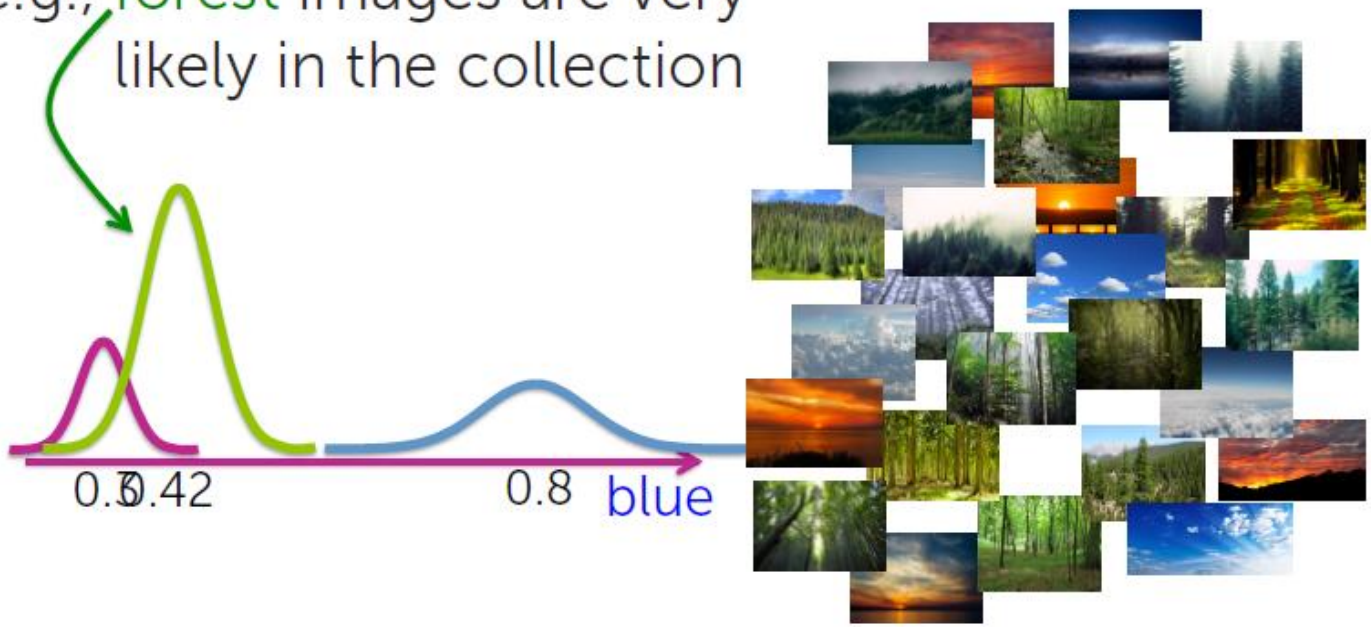
Jumble of unlabeled images



Mixture of Gaussians

What if image types not equally represented?

e.g., forest images are very likely in the collection



Mixture of Gaussians

Combination of weighted Gaussians

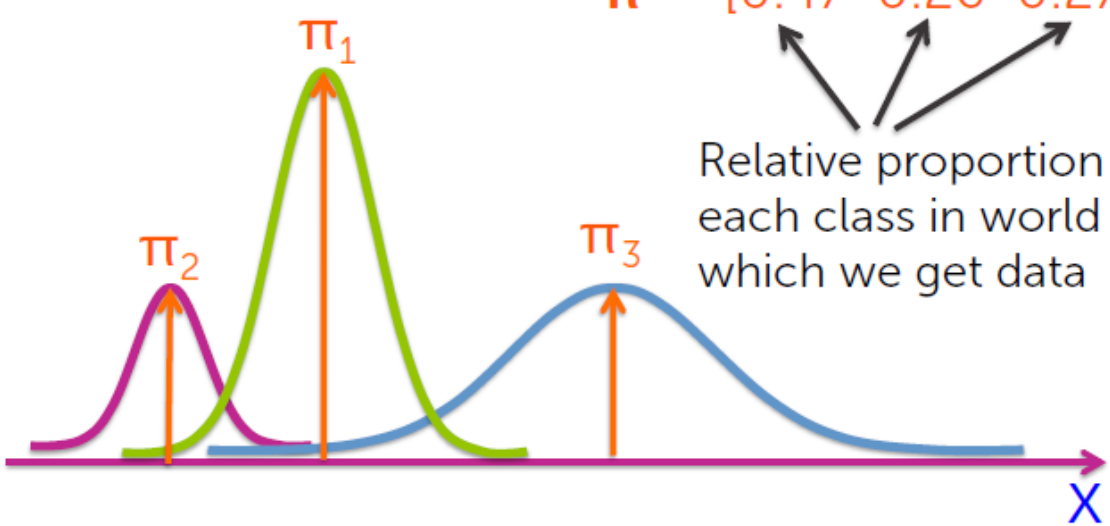
Associate a weight π_k with each Gaussian component

$$\boldsymbol{\pi} = [\overset{\pi_1}{0.47} \quad \overset{\pi_2}{0.26} \quad \overset{\pi_3}{0.27}]$$

Relative proportion of each class in world from which we get data

$$0 \leq \pi_k \leq 1$$

$$\sum_{k=1}^{K} \pi_k = 1$$

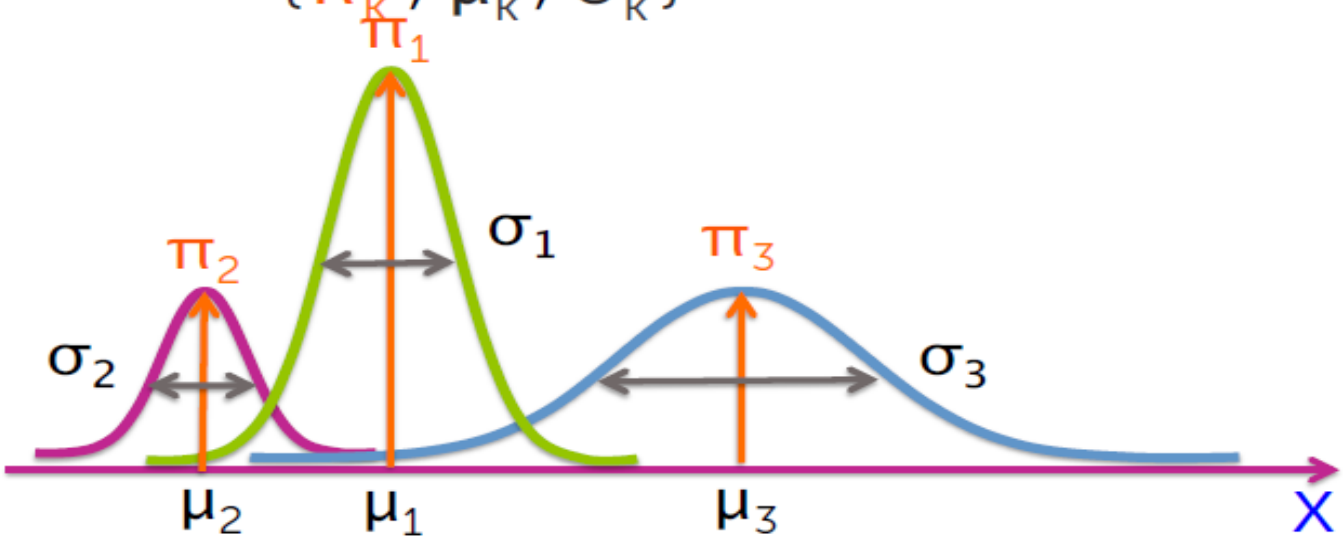


Mixture of Gaussians

Mixture of Gaussians (1D)

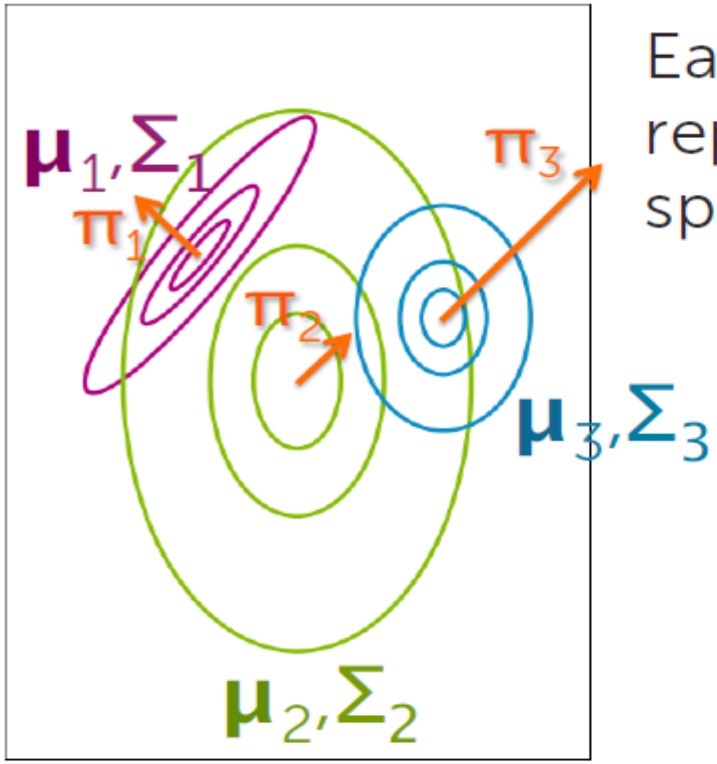
Each mixture component represents a unique cluster specified by:

$\{\pi_k, \mu_k, \sigma_k\}$



Mixture of Gaussians

Mixture of Gaussians (general)



Each mixture component represents a unique cluster specified by:

$$\{\pi_k, \mu_k, \Sigma_k\}$$

Mixture of Gaussians

According to the model...

Without observing the image content, what's the probability it's from cluster k? (e.g., prob. of seeing "clouds" image)

cluster assignment for obs. x_i

$$p(z_i = k) = \pi_k$$

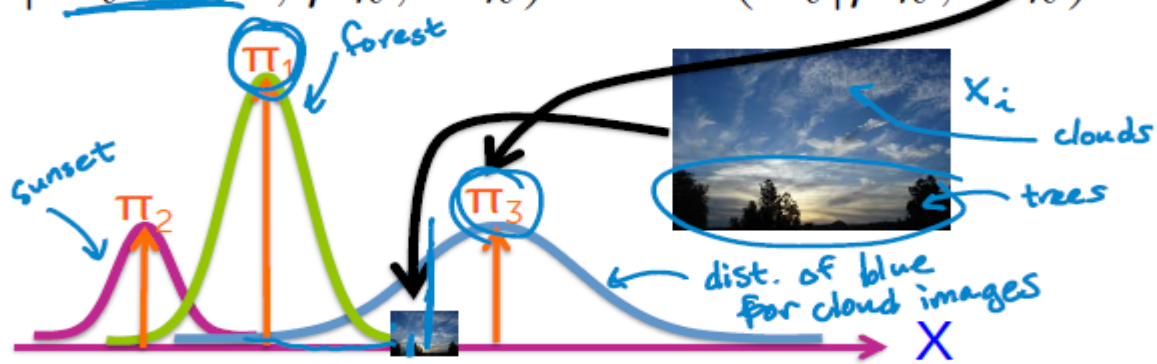
prior

Given observation $\mathbf{x}_i$ is from cluster k, what's the likelihood of seeing $\mathbf{x}_i$? (e.g., just look at distribution for "clouds")

$$p(x_i \mid z_i = k, \mu_k, \Sigma_k) = N(x_i | \mu_k, \Sigma_k)$$

likelihood

$[RGB]_i$



Application: clustering documents

Discover groups of related documents

Application: clustering documents

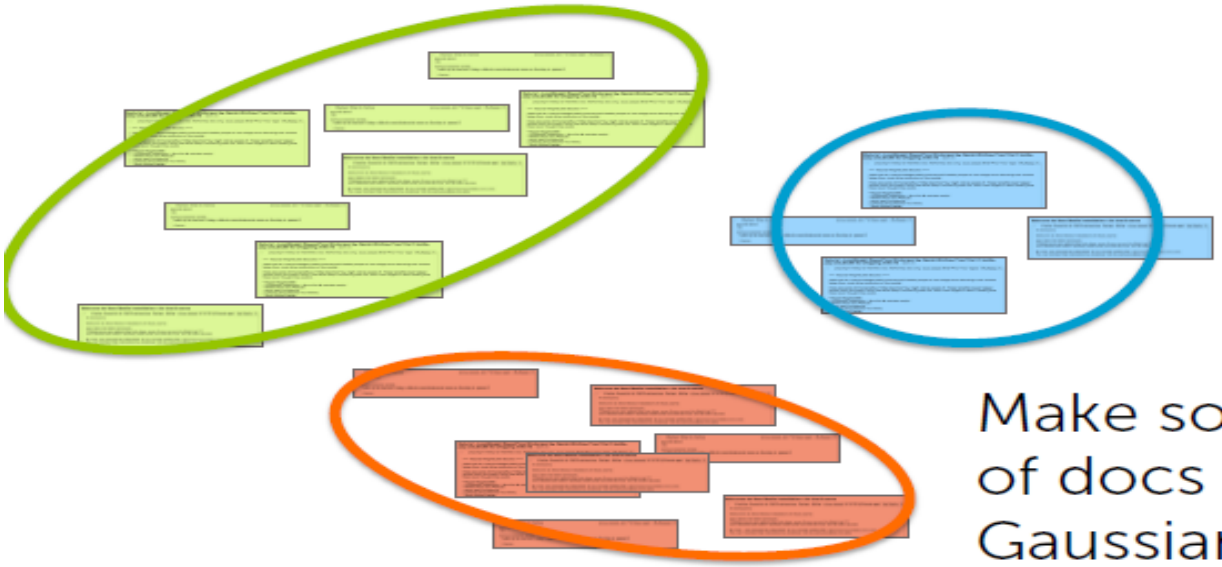
Document representation

$\mathbf{x}_i$ = tf-idf vector

Application: clustering documents

Mixture of Gaussians for clustering documents

Space of all documents
(really lives in $\mathbf{R}^V$ for vocab size V)



Make soft assignments of docs to each Gaussian

Application: clustering documents

Counting parameters

Each cluster has $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k\}$

In 2D:

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{1,2} \\ \sigma_{2,1} & \sigma_2^2 \end{pmatrix}$$

Application: clustering documents

Counting parameters

Each cluster has $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k\}$

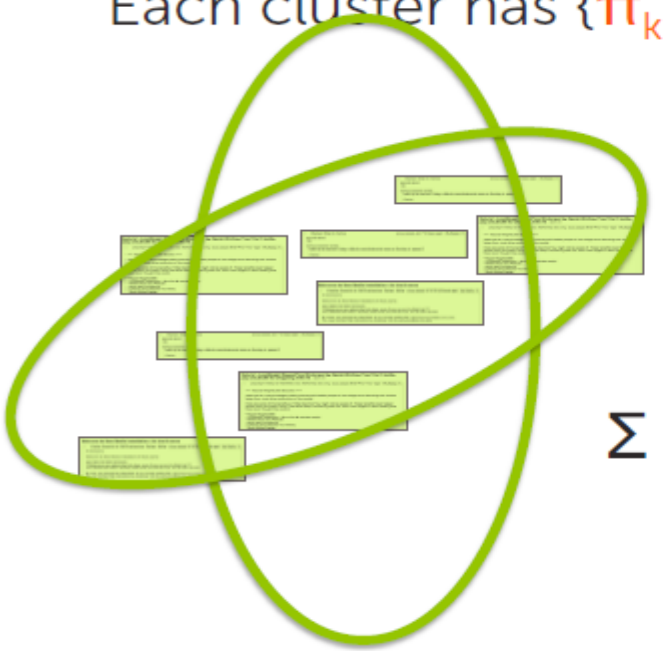
In V (vocab size) dims:

$$\Sigma = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \frac{V(V+1)}{2} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

Application: clustering documents

Restricting to diagonal covariance

Each cluster has $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k \text{ diagonal}\}$

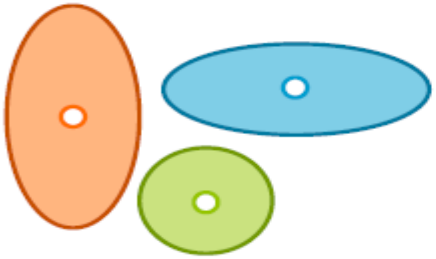


V params

$$\Sigma = \begin{pmatrix} \sigma_1^2 & & & 0 \\ & \sigma_2^2 & & \\ & & \sigma_3^2 & \\ 0 & & & \ddots \\ & & & \sigma_V^2 \end{pmatrix}$$

Application: clustering documents

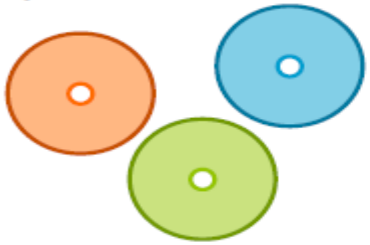
Restrictive assumption, but...

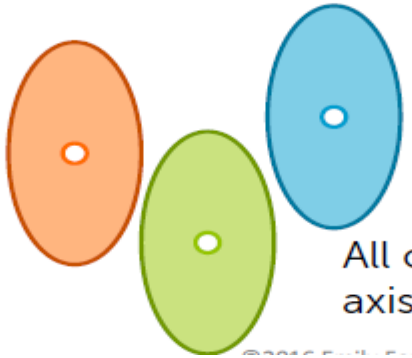


- Can **learn** weights on dimensions (e.g., weights on words in vocab)
- Can learn **cluster-specific** weights on dimensions

Still more flexible than k-means

Spherically symmetric clusters





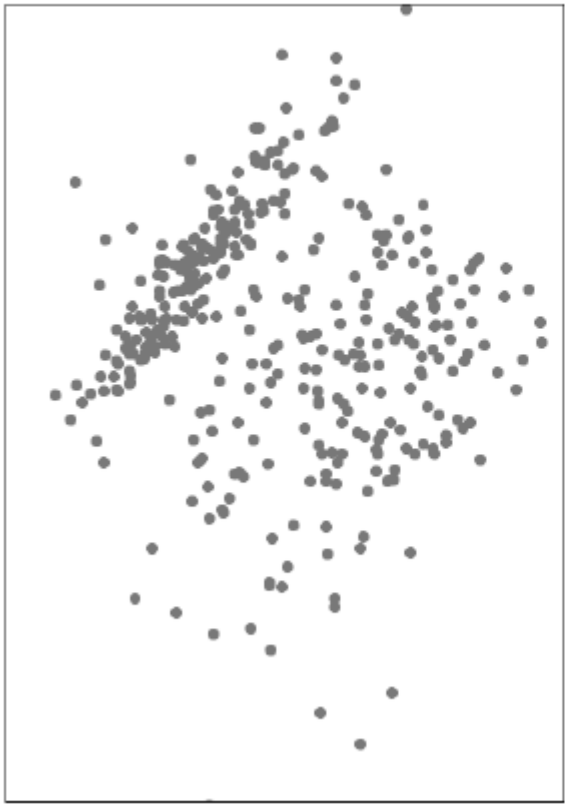
Specify weights...

All clusters have same axis-aligned ellipses

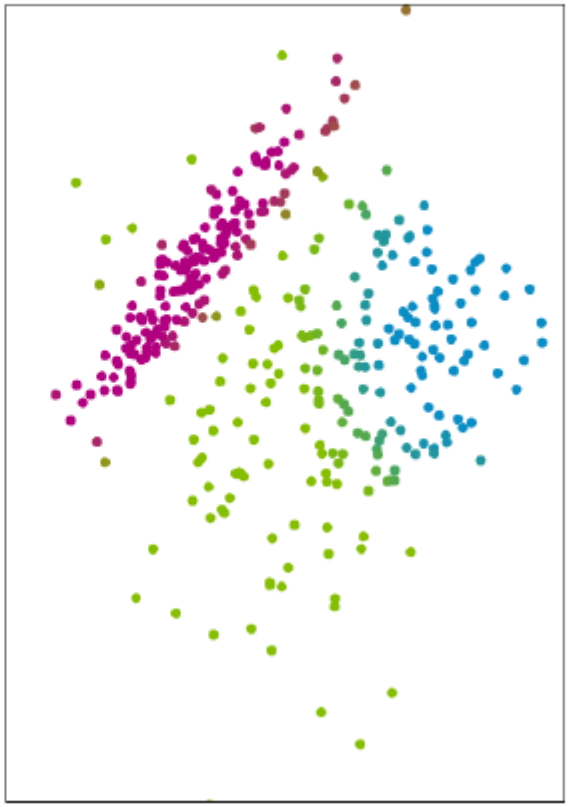
Inferring soft assignments with expectation maximization (EM)

Inferring cluster labels

Data

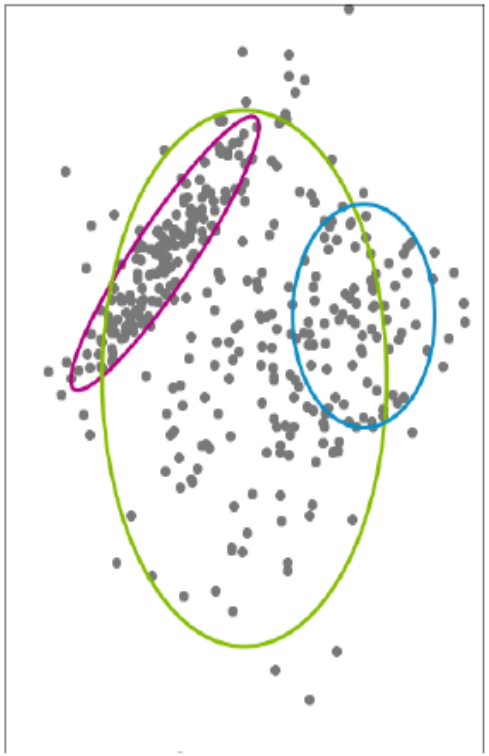


Desired soft assignments



What if we knew the cluster parameters $\{\pi_k, \mu_k, \Sigma_k\}$?

Compute responsibilities



$r_i = [r_{i1}\ r_{i2} \dots r_{iK}]$ # clusters

fixed values defining the distribution

Responsibility cluster k takes for observation i

"given"

$$r_{ik} = p(z_i = k \mid \{\pi_j, \mu_j, \Sigma_j\}_{j=1}^{K}, x_i)$$

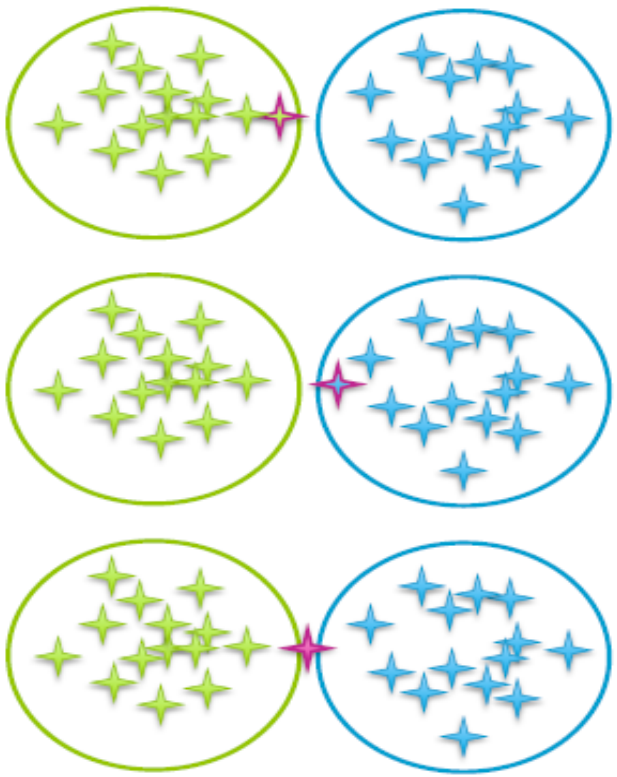
random variable

probability of assignment to cluster k

given model parameters and observed value

What if we knew the cluster parameters $\{\pi_k, \mu_k, \Sigma_k\}$?

Responsibilities in pictures



Green cluster takes more responsibility

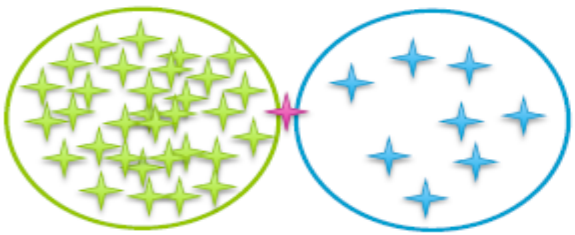
Blue cluster takes more responsibility

Uncertain... split responsibility

What if we knew the cluster parameters $\{\pi_k, \mu_k, \Sigma_k\}$?

Responsibilities in pictures

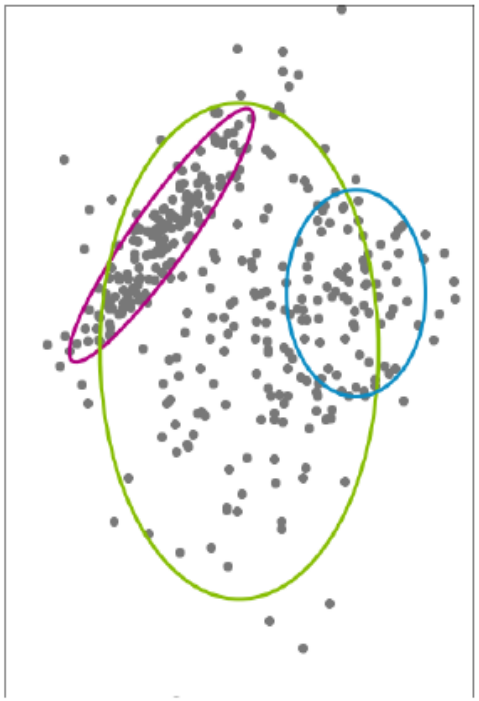
Need to weight by cluster probabilities, not just cluster shapes



Still uncertain,
but green cluster seems more probable...
takes more responsibility

What if we knew the cluster parameters $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k\}$?

Responsibilities in equations

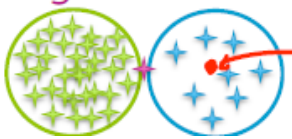


Responsibility cluster k takes for observation i

$$r_{ik} = \pi_k \; N(x_i \mid \mu_k, \Sigma_k)$$

Initial probability of being from cluster k

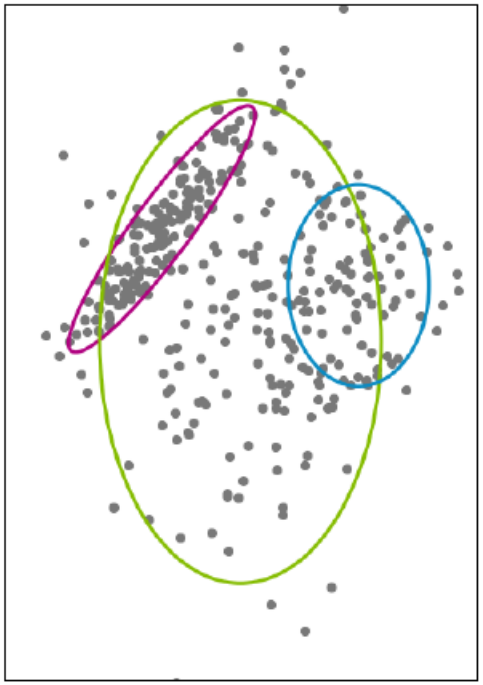
How likely is the observed value $\mathbf{x}_i$ under this cluster assignment?



very unlikely under the green cluster, even though the prior on green is higher

What if we knew the cluster parameters $\{\pi_k, \mu_k, \Sigma_k\}$?

Responsibilities in equations



Responsibility cluster k takes for observation i

$$r_{ik} = \frac{\pi_k \; N(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_i \mid \mu_j, \Sigma_j)}$$

Normalized over all possible cluster assignments

What if we knew the cluster parameters $\{\pi_k, \mu_k, \Sigma_k\}$?

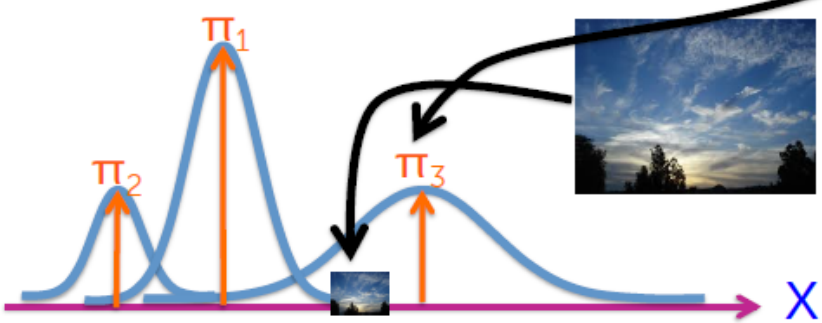
Recall: According to the model...

Without observing the image content, what's the probability it's from cluster k? (e.g., prob. of seeing "clouds" image)

$$p(z_i = k) = \pi_k$$

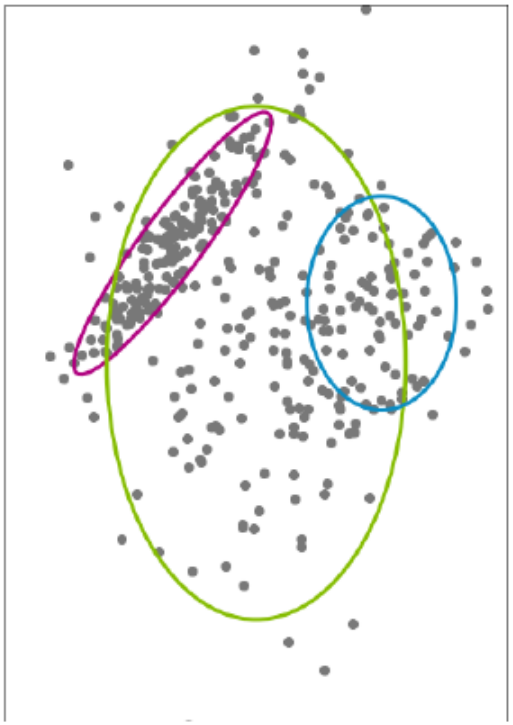
Given observation $\mathbf{x}_i$ is from cluster k, what's the likelihood of seeing $\mathbf{x}_i$? (e.g., just look at distribution for "clouds")

$$p(x_i \mid z_i = k, \mu_k, \Sigma_k) = N(x_i | \mu_k, \Sigma_k)$$



What if we knew the cluster parameters $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k\}$?

Part 1: Summary

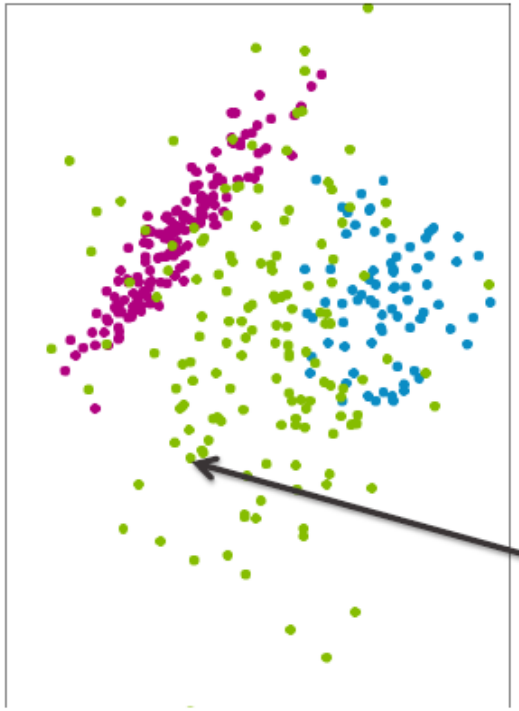


Desired soft assignments (responsibilities) are **easy** to compute when cluster parameters $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k\}$ are known

But, we don't know these!

Imagine we knew the cluster (hard) assignments z_i

Estimating cluster parameters



Imagine we know the cluster assignments

Estimation problem decouples across clusters

Is green point informative of fuchsia cluster parameters?

NO!

Imagine we knew the cluster (hard) assignments z_i

Data table decoupling over clusters

R	G	B	Cluster
$x_1[1]$	$x_1[2]$	$x_1[3]$	3
$x_2[1]$	$x_2[2]$	$x_2[3]$	3
$x_3[1]$	$x_3[2]$	$x_3[3]$	3
$x_4[1]$	$x_4[2]$	$x_4[3]$	1
$x_5[1]$	$x_5[2]$	$x_5[3]$	2
$x_6[1]$	$x_6[2]$	$x_6[3]$	2

Then split into separate tables and consider them independently.

Imagine we knew the cluster (hard) assignments z_i

Maximum likelihood estimation

R	G	B	Cluster
$x_1[1]$	$x_1[2]$	$x_1[3]$	3
$x_2[1]$	$x_2[2]$	$x_2[3]$	3
$x_3[1]$	$x_3[2]$	$x_3[3]$	3

Estimate $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k\}$ given data assigned to cluster k

maximum likelihood estimation (MLE)

Find parameters that maximize the score, or *likelihood*, of data

Imagine we knew the cluster (hard) assignments z_i

Mean/covariance MLE

sum these vectors

R	G	B	Cluster
$x_1[1]$	$x_1[2]$	$x_1[3]$	3
$x_2[1]$	$x_2[2]$	$x_2[3]$	3
$x_3[1]$	$x_3[2]$	$x_3[3]$	3

divide by 3 (the total # of obs.)

denotes "estimate"

$$\hat{\mu}_k = \frac{1}{N_k} \sum_{i \text{ in } k} x_i$$

← average data points in cluster k

# of obs. in cluster

$$\hat{\Sigma}_k = \frac{1}{N_k} \sum_{i \text{ in } k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$$

Scalar case: $\hat{\sigma}_k^2 = \frac{1}{N_k} \sum_{i \text{ in } k} (x_i - \hat{\mu}_k)^2$

Imagine we knew the cluster (hard) assignments z_i

Cluster proportion MLE

R	G	B	Cluster
$x_4[1]$	$x_4[2]$	$x_4[3]$	1

R	G	B	Cluster
$x_5[1]$	$x_5[2]$	$x_5[3]$	2
$x_6[1]$	$x_6[2]$	$x_6[3]$	2

R	G	B	Cluster
$x_1[1]$	$x_1[2]$	$x_1[3]$	3
$x_2[1]$	$x_2[2]$	$x_2[3]$	3
$x_3[1]$	$x_3[2]$	$x_3[3]$	3

obs in cluster k

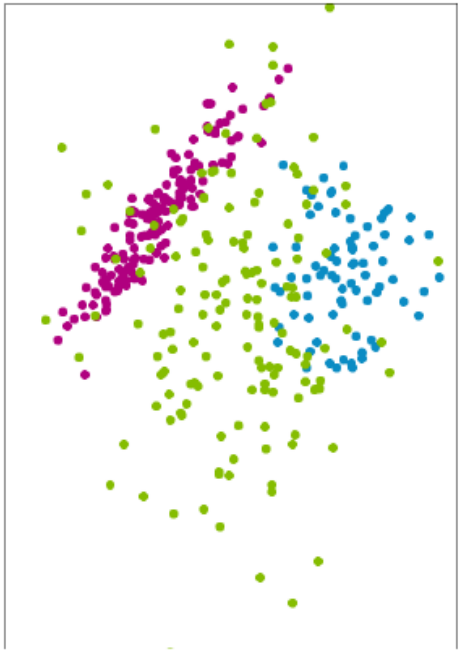
$$\hat{\pi}_k = \frac{N_k}{N}$$

total # of obs

True for general mixtures of i.i.d. data, not just Gaussian clusters

Imagine we knew the cluster (hard) assignments z_i

Part 2a : Summary



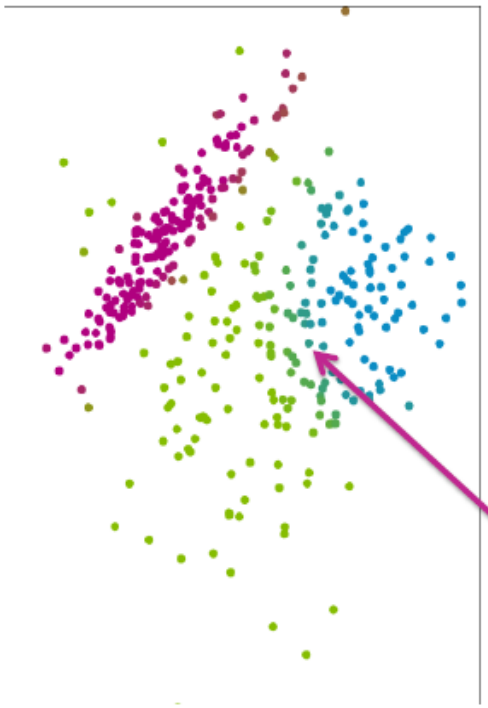
needed to compute soft assignments

Cluster parameters are simple to compute if we know the cluster assignments

But, we don't know these!

What can we do with just soft assignments r_{ij}?

Estimating cluster parameters from soft assignments



Instead of having a full observation $\mathbf{x}_i$ in cluster k, just allocate a portion r_{ik}

$\mathbf{x}_i$ divided across all clusters, as determined by r_{ik}

What can we do with just soft assignments r_{ij}?

Maximum likelihood estimation from soft assignments

Just like in boosting with weighted observations...

R	G	B	r_{i1}	r_{i2}	r_{i3}
$x_1[1]$	$x_1[2]$	$x_1[3]$	0.30	0.18	0.52
$x_2[1]$	$x_2[2]$	$x_2[3]$	0.01	0.26	0.73
$x_3[1]$	$x_3[2]$	$x_3[3]$	0.002	0.008	0.99
$x_4[1]$	$x_4[2]$	$x_4[3]$	0.75	0.10	0.15
$x_5[1]$	$x_5[2]$	$x_5[3]$	0.05	0.93	0.02
$x_6[1]$	$x_6[2]$	$x_6[3]$	0.13	0.86	0.01

52% chance this obs is in cluster 3

Total weight in cluster: **1.242** **2.8** **2.42**
(effective # of obs)

What can we do with just soft assignments r_{ij}?

Maximum likelihood estimation from soft assignments

R	G	B	Cluster 1 weights
$x_1[1]$	$x_1[2]$	$x_1[3]$	0.30
$x_2[1]$			
$x_3[1]$			
$x_4[1]$			
$x_5[1]$			
$x_6[1]$			

R	G	B	Cluster 2 weights
$x_1[1]$	$x_1[2]$	$x_1[3]$	0.18
$x_2[1]$			
$x_3[1]$			
$x_4[1]$			
$x_5[1]$			
$x_6[1]$			

R	G	B	Cluster 3 weights
$x_1[1]$	$x_1[2]$	$x_1[3]$	0.52
$x_2[1]$	$x_2[2]$	$x_2[3]$	0.73
$x_3[1]$	$x_3[2]$	$x_3[3]$	0.99
$x_4[1]$	$x_4[2]$	$x_4[3]$	0.15
$x_5[1]$	$x_5[2]$	$x_5[3]$	0.02
$x_6[1]$	$x_6[2]$	$x_6[3]$	0.01

What can we do with just soft assignments r_{ij}?

Cluster-specific location/shape MLE

R	G	B	Cluster 1 weights
$x_1[1]$	$x_1[2]$	$x_1[3]$	0.30
$x_2[1]$	$x_2[2]$	$x_2[3]$	0.01
$x_3[1]$	$x_3[2]$	$x_3[3]$	0.002
$x_4[1]$	$x_4[2]$	$x_4[3]$	0.75
$x_5[1]$	$x_5[2]$	$x_5[3]$	0.05
$x_6[1]$	$x_6[2]$	$x_6[3]$	0.13

$$\hat{\mu}_k = \frac{1}{N_k^{\text{soft}}} \sum_{i=1}^{N} r_{ik} x_i$$

$$\hat{\Sigma}_k = \frac{1}{N_k^{\text{soft}}} \sum_{i=1}^{N} r_{ik} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$$

$$N_k^{\text{soft}} = \sum_{i=1}^{N} r_{ik}$$

Total weight in cluster k = effective # obs

Compute cluster parameter estimates with weights on each row operation

What can we do with just soft assignments r_{ij}?

MLE of cluster proportions $\hat{\pi}_k$

r_{i1}	r_{i2}	r_{i3}
0.30	0.18	0.52
0.01	0.26	0.73
0.002	0.008	0.99
0.75	0.10	0.15
0.05	0.93	0.02
0.13	0.86	0.01

Total weight in cluster:

1.242	**2.8**	**2.42**

Total weight in dataset: 6

datapoints N

$$\hat{\pi}_k = \frac{N_k^{\text{soft}}}{N}$$

$$N_k^{\text{soft}} = \sum_{i=1}^{N} r_{ik}$$

Total weight in cluster k = effective # obs

Estimate cluster proportions from relative weights

What can we do with just soft assignments r_{ij}?

Defaults to hard assignment case when r_{ij} in {0,1}

Hard assignments have:

$$r_{ik} = \begin{cases} 1 & i \text{ in } k \\ 0 & \text{otherwise} \end{cases}$$

R	G	B	r_{i1}	r_{i2}	r_{i3}
$x_1[1]$	$x_1[2]$	$x_1[3]$	0	0	1
$x_2[1]$	$x_2[2]$	$x_2[3]$	0	0	1
$x_3[1]$	$x_3[2]$	$x_3[3]$	0	0	1
$x_4[1]$	$x_4[2]$	$x_4[3]$	1	0	0
$x_5[1]$	$x_5[2]$	$x_5[3]$	0	1	0
$x_6[1]$	$x_6[2]$	$x_6[3]$	0	1	0
Total weight in cluster:			1	2	3

One-hot encoding of cluster assignment

What can we do with just soft assignments r_{ij}?

Equating the estimates...

$$\hat{\pi}_k = \frac{N_k^{\text{soft}}}{N}$$

$$\boxed{N_k^{\text{soft}} = \sum_{i=1}^{N} r_{ik}}$$

if {0,1} just count obs i in cluster k if $r_{ik}=1$ $= N_k$ ✓

$$\hat{\mu}_k = \frac{1}{N_k^{\text{soft}}} \sum_{i=1}^{N} r_{ik} x_i$$

only add x_i if i in k ($r_{ik}=1$) $\rightarrow \frac{1}{N_k} \sum_{i \text{ in } k} x_i$ ✓

$$\hat{\Sigma}_k = \frac{1}{N_k^{\text{soft}}} \sum_{i=1}^{N} r_{ik} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$$ ✓

same as above

$$= \frac{1}{N_k} \sum_{i \text{ in } k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$$

What can we do with just soft assignments r_{ij}?

Part 2b: Summary

Still straightforward to compute cluster parameter estimates from soft assignments

Expectation maximization (ME)

An iterative algorithm

Motivates an iterative algorithm:

1. **E-step:** estimate cluster responsibilities given current parameter estimates

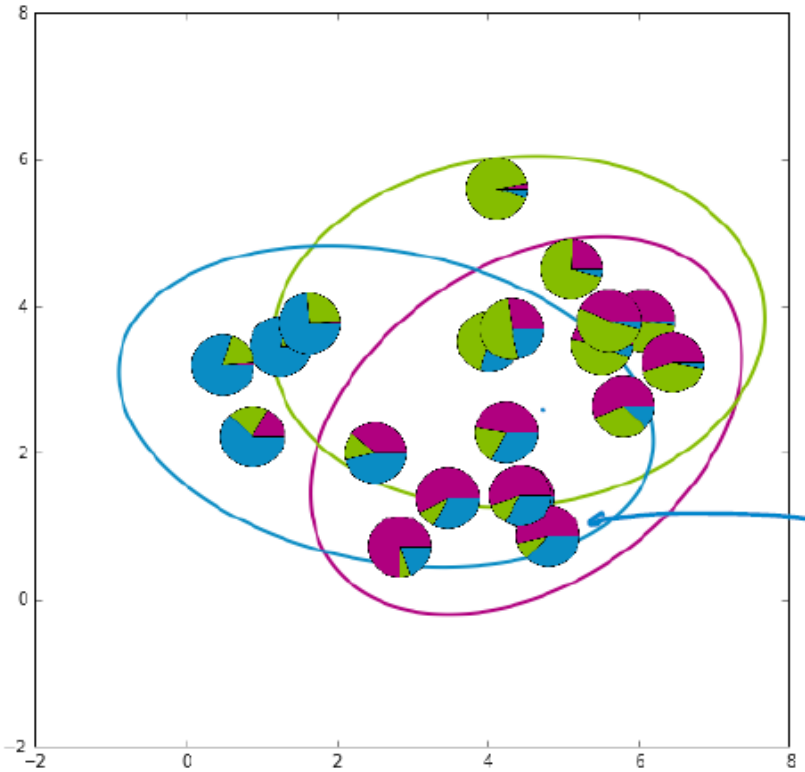
$$\hat{r}_{ik} = \frac{\hat{\pi}_k N(x_i \mid \hat{\mu}_k, \hat{\Sigma}_k)}{\sum_{j=1}^{K} \hat{\pi}_j N(x_i \mid \hat{\mu}_j, \hat{\Sigma}_j)}$$

2. **M-step:** maximize likelihood over parameters given current responsibilities

$$\hat{\pi}_k, \hat{\mu}_k, \hat{\Sigma}_k \mid \{\hat{r}_{ik}, x_i\}$$

Expectation maximization (ME)

EM for mixtures of Gaussians in pictures – initialization



Initialize

iter counter

$\{\pi_k^{(0)}, \hat{\mu}_k^{(0)}, \hat{\Sigma}_k^{(0)}\}$

Then compute

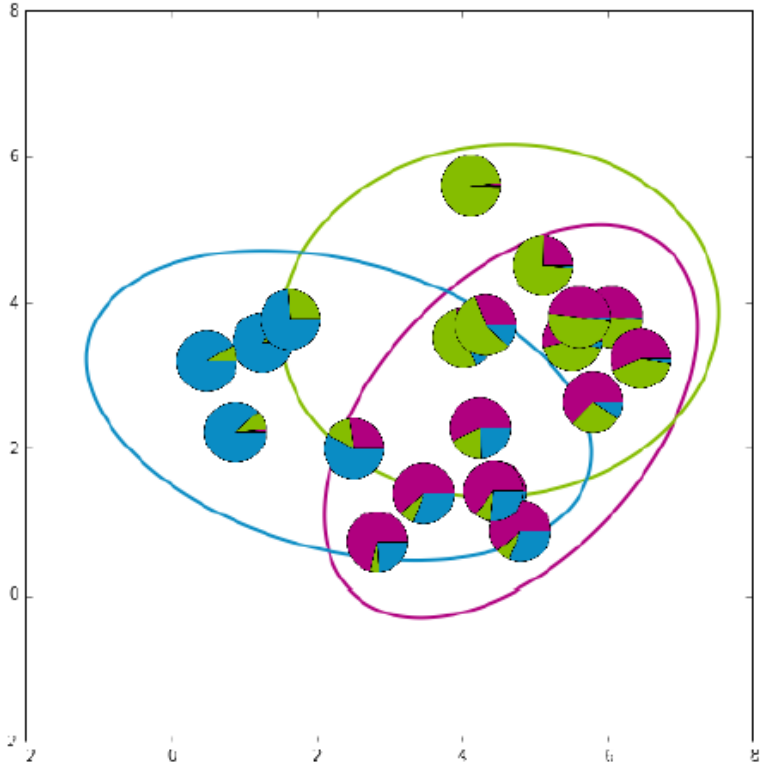
$\hat{r}_{ik}^{(1)}$

fucksia blue green

$\hat{r}_i^{(1)} = [0.52 \quad 0.4 \quad 0.08]$

Expectation maximization (ME)

EM for mixtures of Gaussians in pictures – after 1st iteration



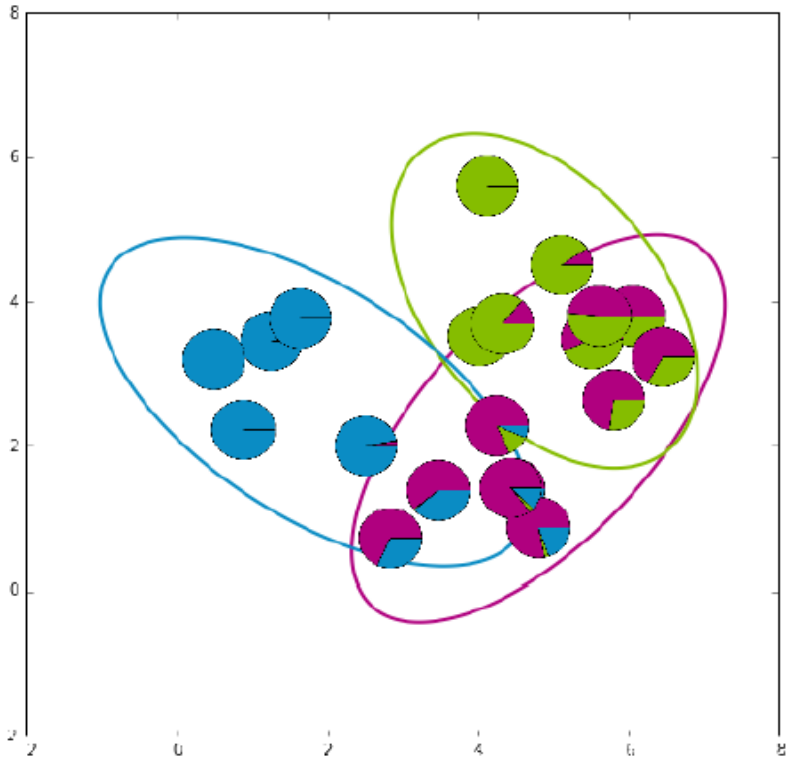
Maximize likelihood given soft assign. $r_{ik}^{(1)}$

$\rightarrow \{\hat{\pi}_k^{(1)}, \hat{\mu}_k^{(1)}, \hat{\Sigma}_k^{(1)}\}$

Then recompute responsibilities $\hat{r}_{ik}^{(2)}$

Expectation maximization (ME)

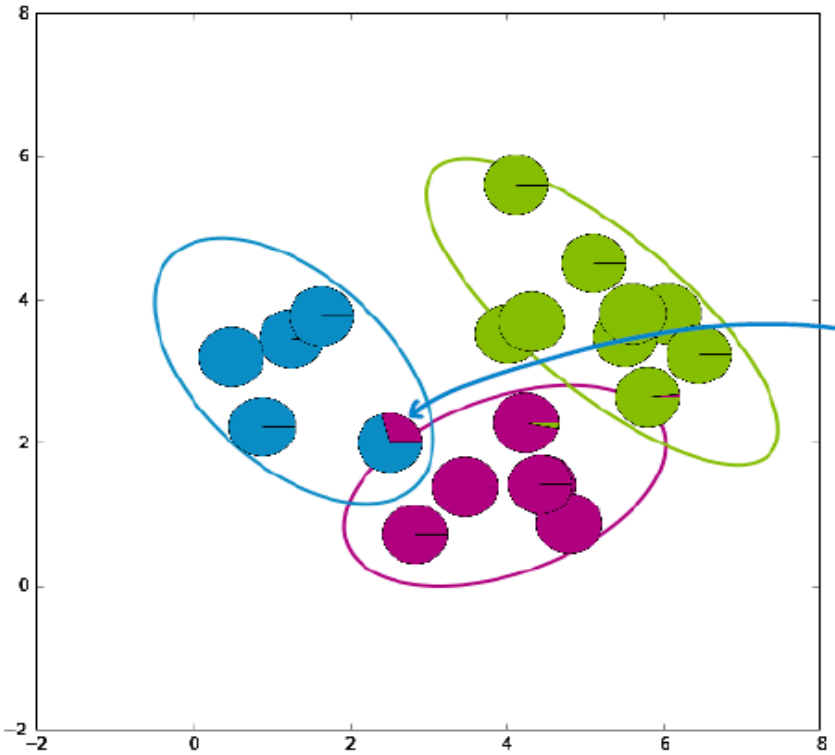
EM for mixtures of Gaussians in pictures – after 2nd iteration



rinse
+
repeat
until convergence

Expectation maximization (ME)

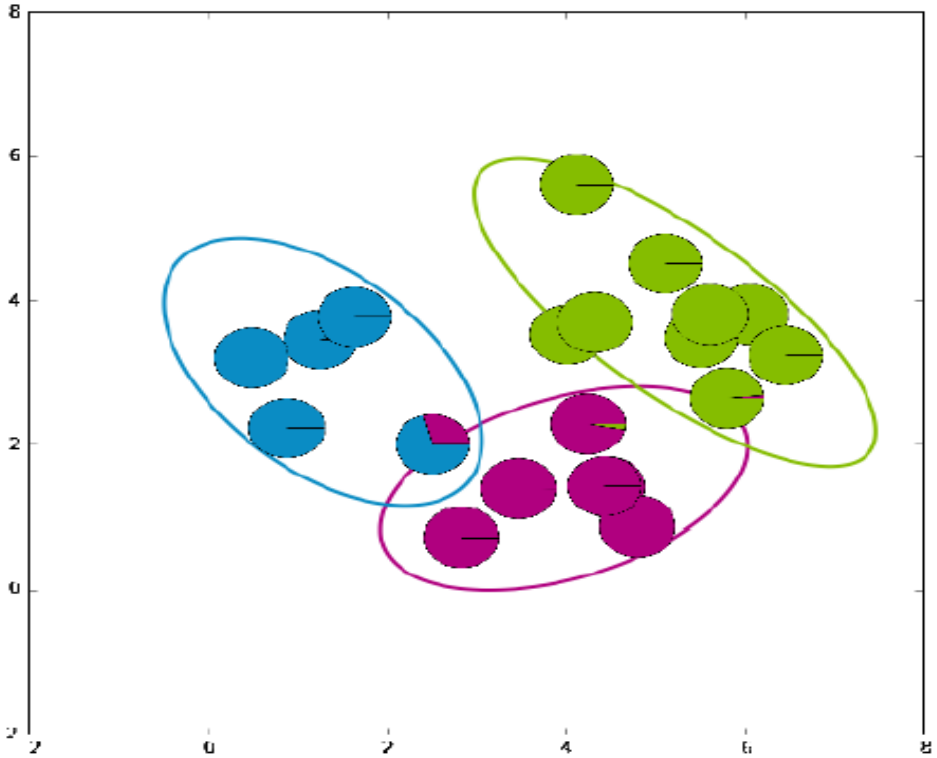
EM for mixtures of Gaussians in pictures – converged solution



clearly see uncertainty in assignment of obs. to blue or fuchsia cluster, even in final assignments.

Expectation maximization (ME)

EM for mixtures of Gaussians in pictures - replay



Expectation maximization (ME)

Convergence of EM

- EM is a coordinate-ascent algorithm
  - Can equate E-and M-steps with alternating maximizations of an objective function
- Convergences to a local mode

- We will assess via (log) likelihood of data under current parameter and responsibility estimates

Expectation maximization (ME)

Initialization

- Many ways to initialize the EM algorithm
- Important for convergence rates and quality of local mode found
- Examples:
  - Choose K observations at random to define K "centroids". Assign other observations to nearest centriod to form initial parameter estimates.
  - Pick centers sequentially to provide good coverage of data like in k-means++
  - Initialize from k-means solution
  - Grow mixture model by splitting (and sometimes removing) clusters until K clusters are formed

Expectation maximization (ME)

Overfitting of MLE

Maximizing likelihood can overfit to data

Imagine at K=2 example with one obs assigned to cluster 1 and others assigned to cluster 2

- What parameter values maximize likelihood?

Set center equal to point and shrink variance to 0

Likelihood goes to ∞ !

Expectation maximization (ME)

Overfitting in high dims

Doc-clustering example:
Imagine only 1 doc assigned to cluster k has word w (or all docs in cluster agree on count of word w)

Likelihood maximized by setting $\boldsymbol{\mu}_k[w] = \mathbf{x}_i[w]$ and $\sigma^2_{w,k} = 0$

Likelihood of any doc with different count on word w being in cluster k is 0!

Expectation maximization (ME)

Simple regularization of M-step for mixtures of Gaussians

Simple fix: Don't let variances → 0!

Add small amount to diagonal of covariance estimate

Alternatively, take Bayesian approach and place prior on parameters.

Similar idea, but all parameter estimates are "smoothed" via cluster pseudo-observations.

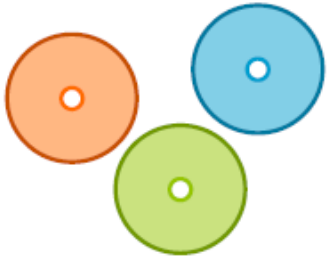
Expectation maximization (ME)

Relationship to k-means

Consider Gaussian mixture model with

$$\Sigma = \begin{pmatrix} \sigma^2 & & & \\ & \sigma^2 & & \\ & & \sigma^2 & \\ & & & \ddots \end{pmatrix}$$

Spherically symmetric clusters



and let the variance parameter $\sigma \rightarrow 0$

Datapoint gets fully assigned to nearest center, just as in k-means

- Spherical clusters with equal variances, so relative likelihoods just function of distance to cluster center
- As variances→0, likelihood ratio becomes 0 or 1
- Responsibilities weigh in cluster proportions, but dominated by likelihood disparity

$$\hat{r}_{ik} = \frac{\hat{\pi}_k N(x_i \mid \hat{\mu}_k, \sigma^2 I)}{\sum_{j=1}^{K} \hat{\pi}_j N(x_i \mid \hat{\mu}_j, \sigma^2 I)}$$

Expectation maximization (ME)

Infinitesimally small variance EM = k-means

1. **E-step:** estimate cluster responsibilities given current parameter estimates

$$\hat{r}_{ik} = \frac{\hat{\pi}_k N(x_i \mid \hat{\mu}_k, \sigma^2 I)}{\sum_{j=1}^{K} \hat{\pi}_j N(x_i \mid \hat{\mu}_j, \sigma^2 I)} \in \{0, 1\}$$

Infinitesimally small

Decision based on distance to nearest cluster center

2. **M-step:** maximize likelihood over parameters given current responsibilities (hard assignments!)

$$\hat{\pi}_k, \hat{\mu}_k \mid \{\hat{r}_{ik}, x_i\}$$

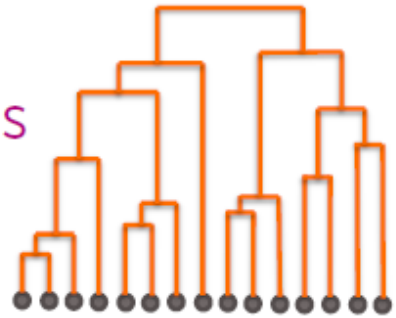
What you can do now …

- Interpret a probabilistic model-based approach to clustering using mixture models
- Describe model parameters
- Motivate the utility of soft assignments and describe what they represent
- Discuss issues related to how the number of parameters grow with the number of dimensions
  - Interpret diagonal covariance versions of mixtures of Gaussians
- Compare and contrast mixtures of Gaussians and k-means
- Implement an EM algorithm for inferring soft assignments and cluster parameters
  - Determine an initialization strategy
  - Implement a variant that helps avoid overfitting issues

Hierarchical clustering

Why hierarchical clustering

- Avoid choosing # clusters beforehand
- Dendrograms help visualize different clustering granularities
  - No need to rerun algorithm

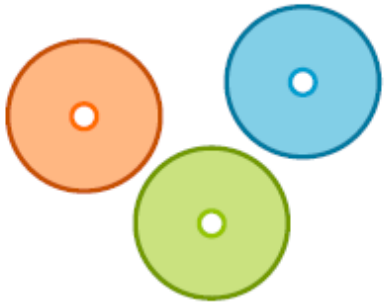


- Most algorithms allow user to choose any distance metric
  - k-means restricted us to Euclidean distance

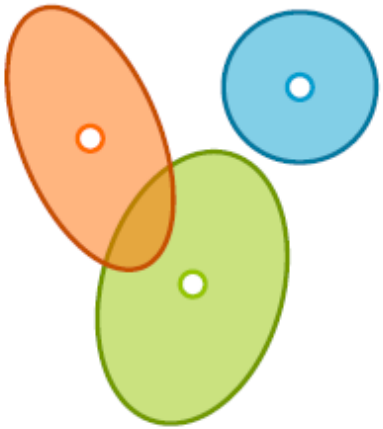
Why hierarchical clustering

Can often find more complex shapes than k-means or Gaussian mixture models

k-means: spherical clusters



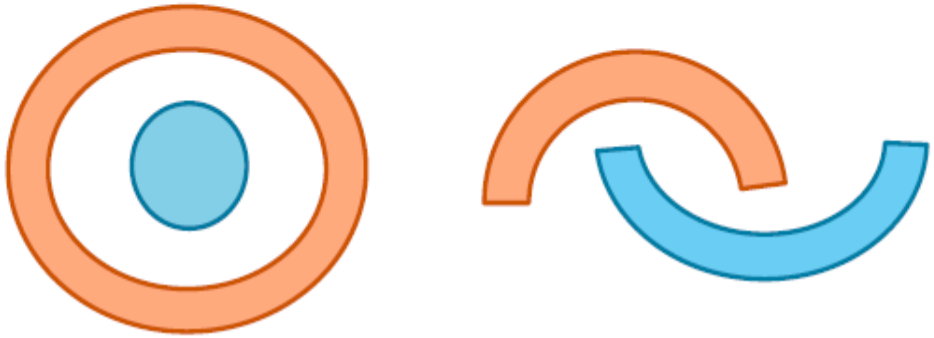
Gaussian mixtures: ellipsoids



Why hierarchical clustering

Can often find more complex shapes than k-means or Gaussian mixture models

What about these?



Two main types of algorithms

Divisive, *a.k.a top-down:* Start with all data in one big cluster and recursively split.

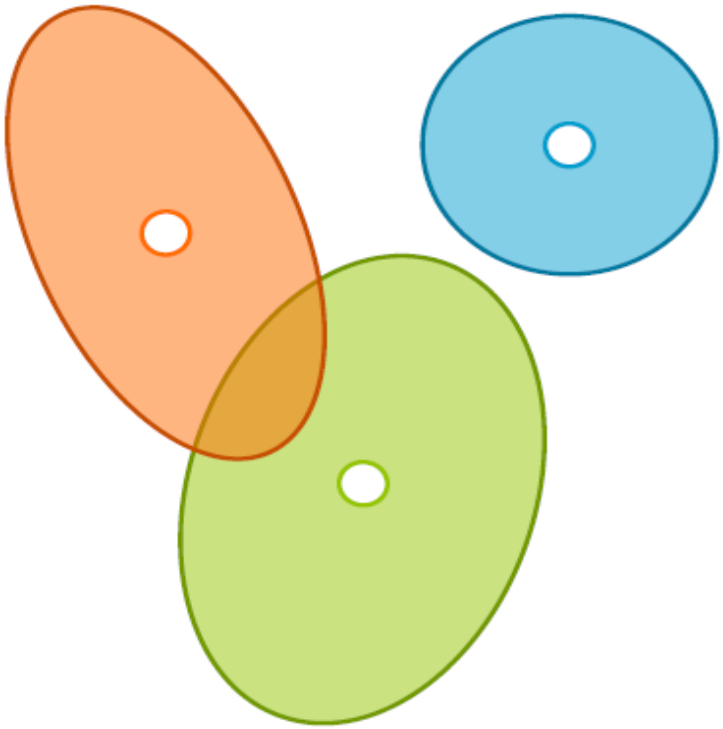
- Example: recursive k-means

Agglomerative *a.k.a. bottom-up*: Start with each data point as its own cluster. Merge clusters until all points are in one big cluster.

- Example: single linkage

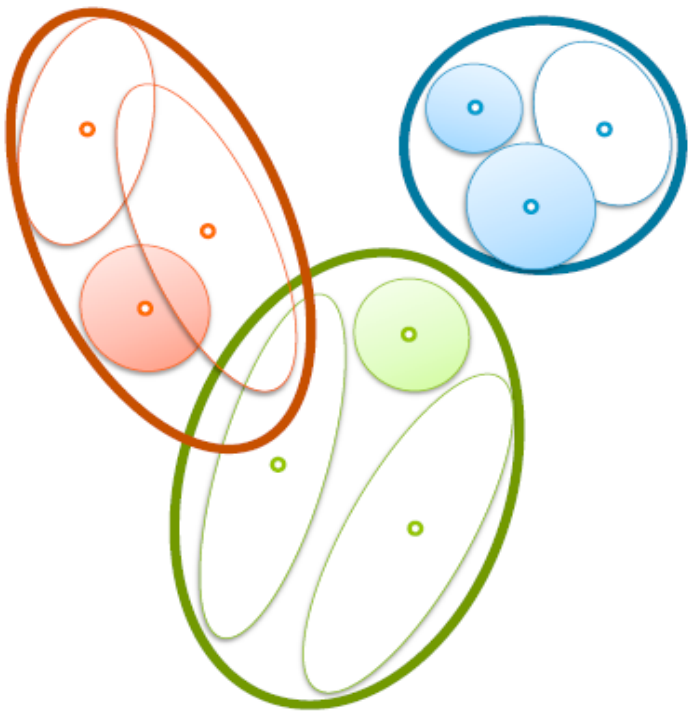
Divisive clustering

Divisive in pictures – level 1

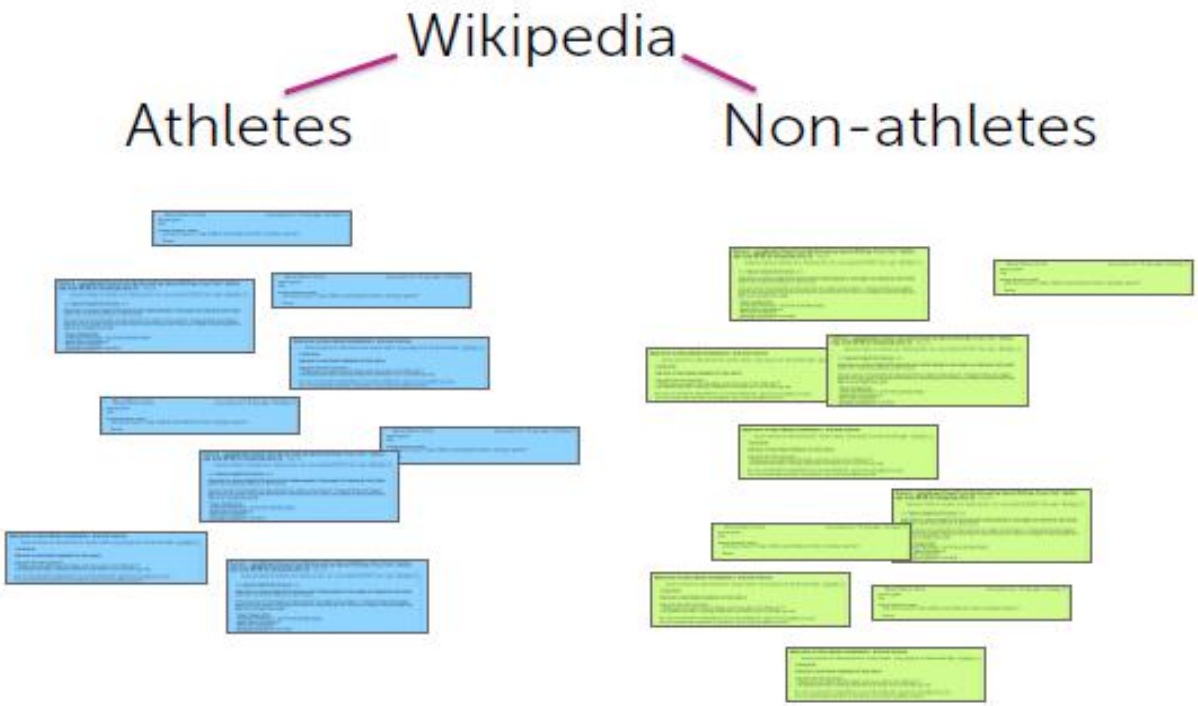


Divisive clustering

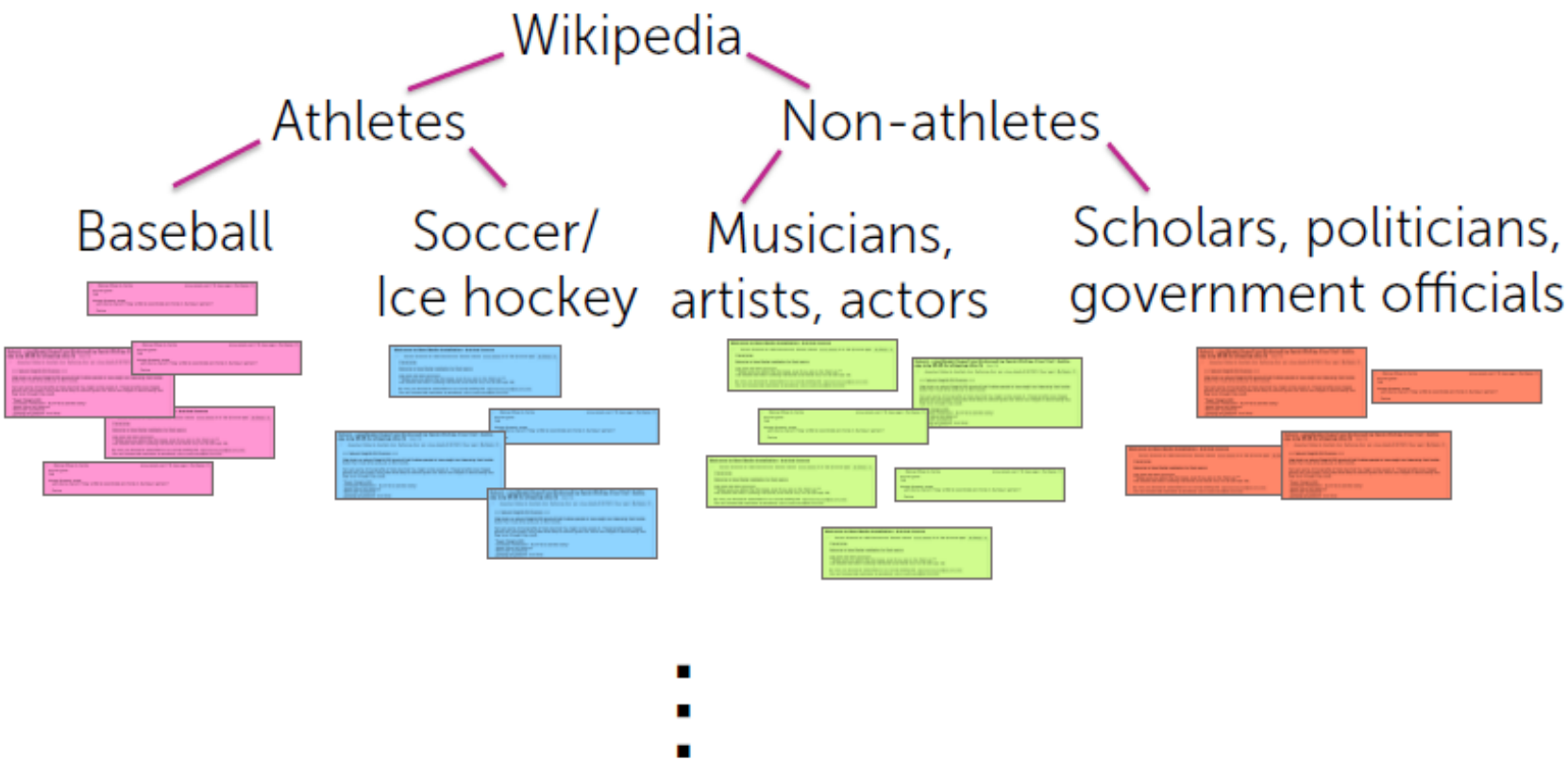
Divisive in pictures – level 2



Divisive: Recursive k-means



Divisive: Recursive k-means

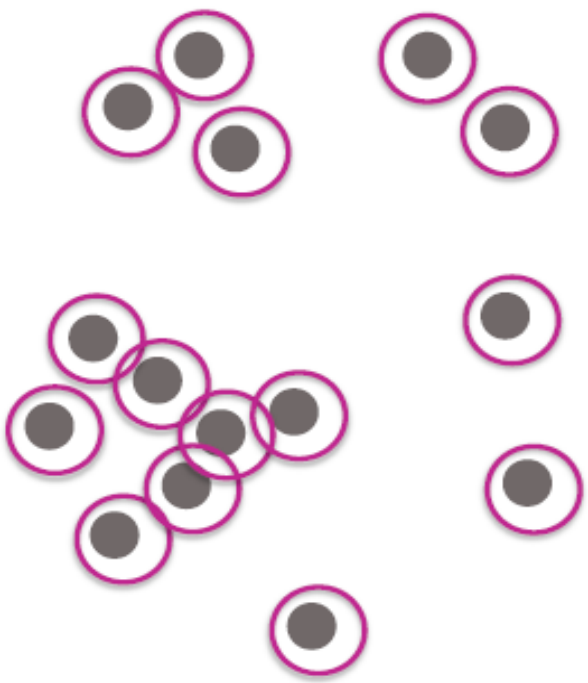


Divisive: choices to be made

- Which algorithm to recurse
- How many clusters per split
- When to split vs. stop
  - Max cluster size:
    number of points in cluster falls below threshold
  - Max cluster radius:
    distance to furthest point falls below threshold
  - Specified # clusters:
    split until pre-specified # clusters is reached

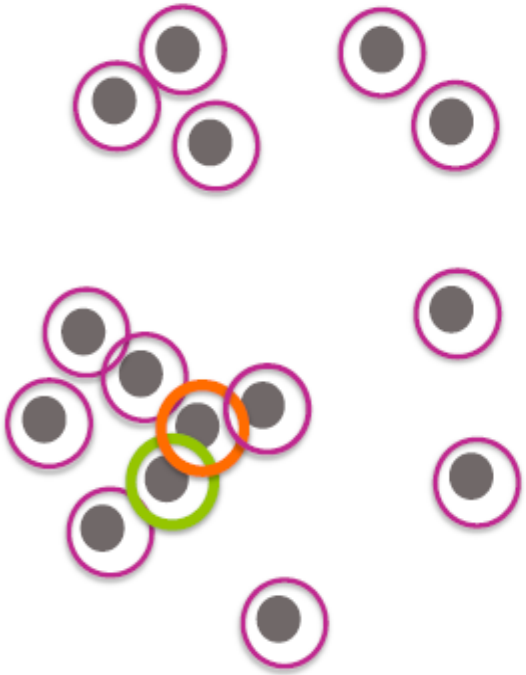
Aglomerative: Single linkage

1. Initialize each point to be its own cluster



Aglomerative: Single linkage

2. Define distance between clusters to be:



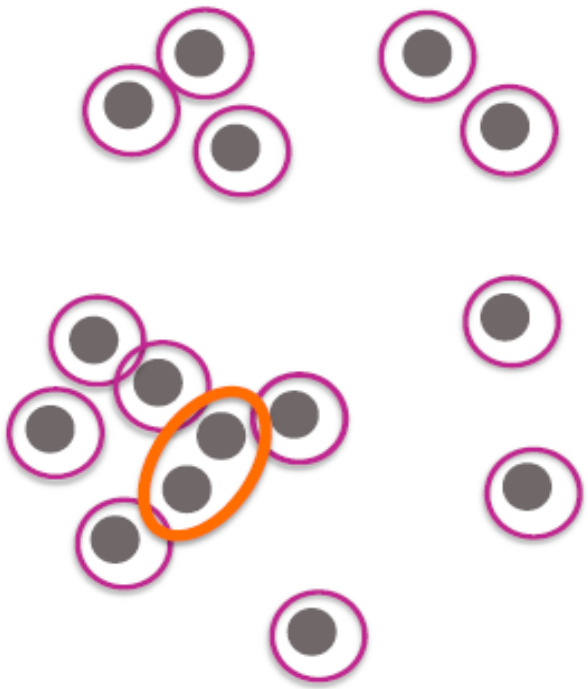
$$\text{distance}(C_1, C_2) = \min_{\substack{x_i \text{ in } C_1, \\ x_j \text{ in } C_2}} d(x_i, x_j)$$

specified pairwise distance function

Linkage criteria

Aglomerative: Single linkage

3. Merge the two closest clusters

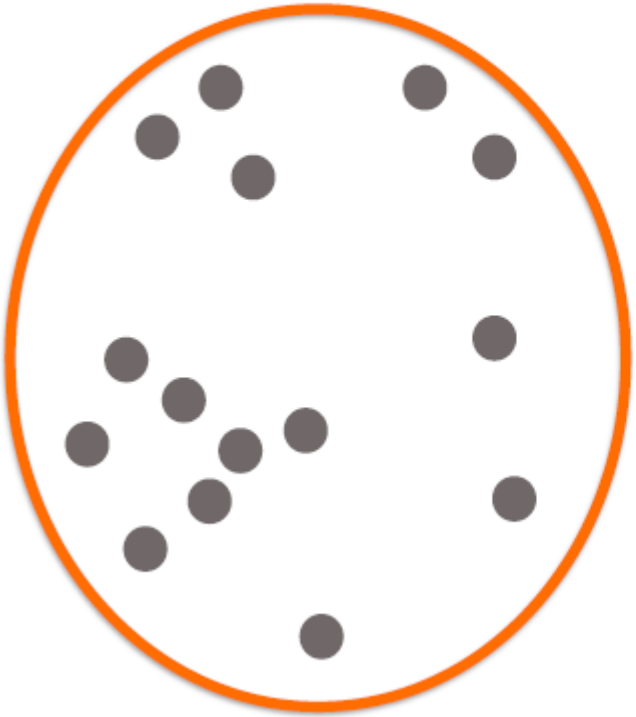


Aglomerative: Single linkage

4. Repeat step 3 until all points are in one cluster

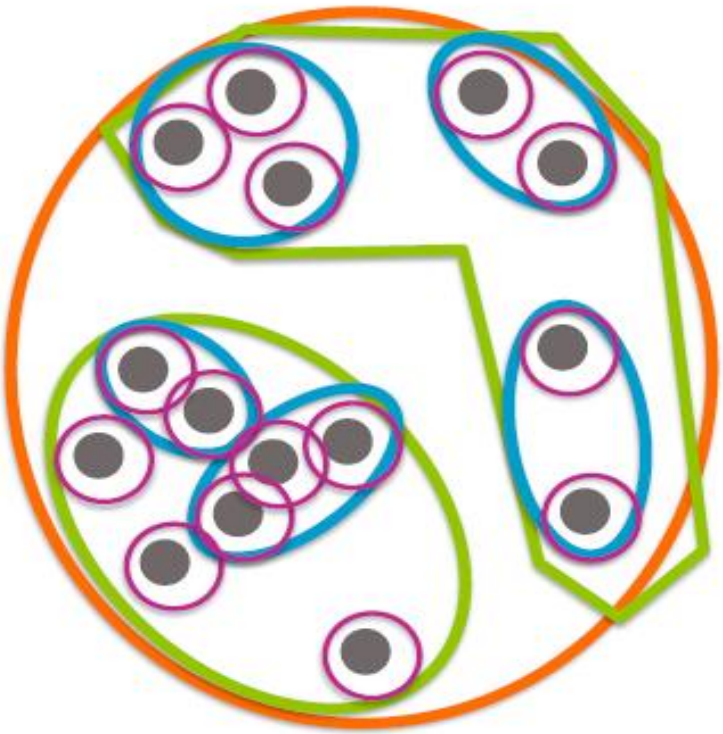
Aglomerative: Single linkage

4. Repeat step 3 until all points are in one cluster



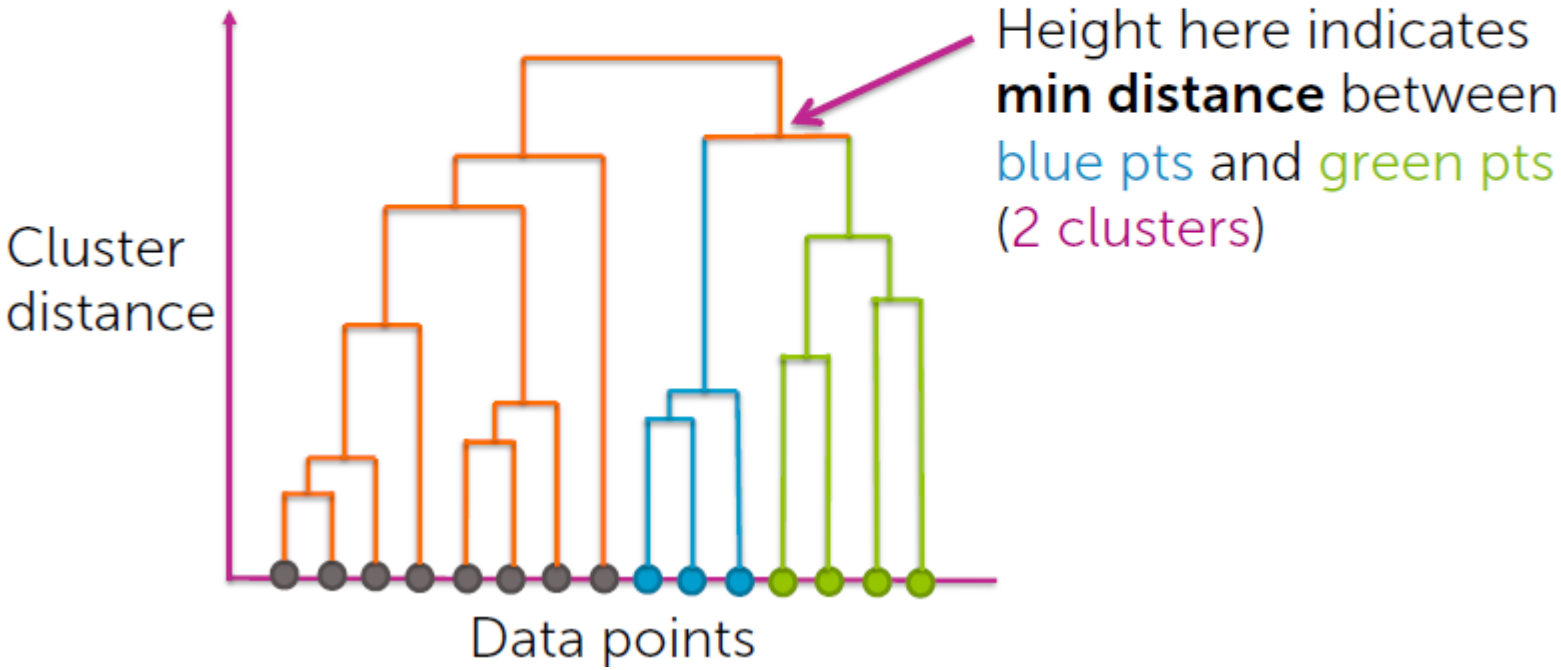
Cluster of clusters

Just like our picture for divisive clustering...



The dendrogram

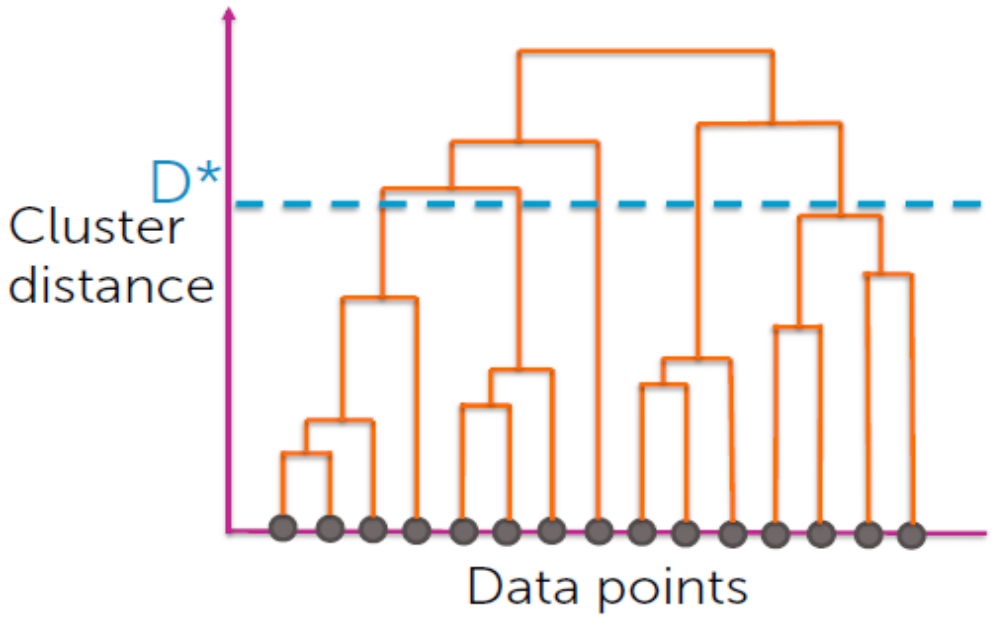
- x axis shows data points (carefully ordered)
- y-axis shows distance between pair of clusters



Extracting a partition

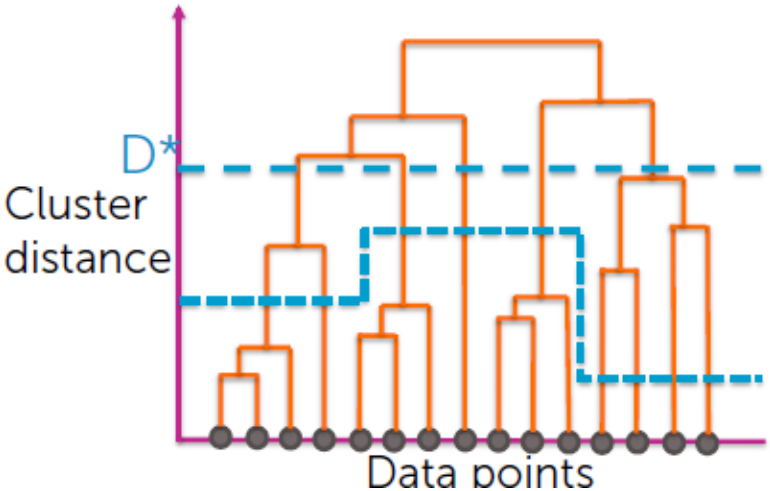
Choose a distance D* at which to cut dendogram

Every branch that crosses D* becomes a separate cluster



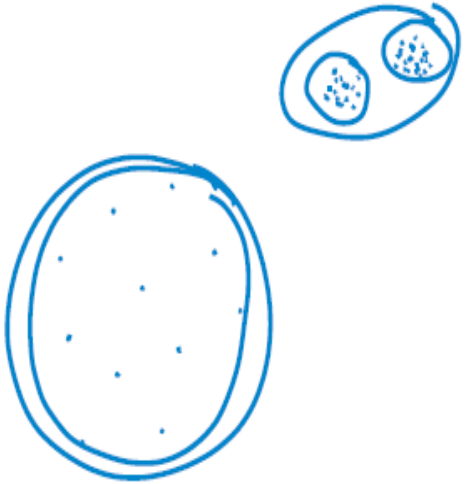
Agglomerative: choices to be made

- Distance metric: $d(\mathbf{x}_i, \mathbf{x}_j)$
- Linkage function: e.g., $\min_{\substack{\mathbf{x}_i \text{ in } C_1, \\ \mathbf{x}_j \text{ in } C_2}} d(\mathbf{x}_i, \mathbf{x}_j)$
- Where and how to cut dendrogram



More on cutting dendrogram

- For visualization, smaller # clusters is preferable
- For tasks like outlier detection, cut based on:
  - Distance threshold
  - Inconsistency coefficient
    - Compare height of merge to average merge heights below
    - If top merge is substantially higher, then it is joining two subsets that are relatively far apart compared to the members of each subset internally
    - Still have to choose a threshold to cut at, but now in terms of "inconsistency" rather than distance
- No cutting method is "incorrect", some are just more useful than others



Computational considerations

- Computing all pairs of distances is expensive
  - Brute force algorithm is $O(N^2 \log(N))$

datapoints

- Smart implementations use triangle inequality to rule out candidate pairs

- Best known algorithm is $O(N^2)$