assignmentsties current parametassignmentsties current parametassignmentsties current paramet**Clustering**

Week 1 Nearest Neighbor Search

* Retrieving documents

(1) Nearest Neighbor Search

arrange all articles in space based on their similarities to one another, and find the nearest one or set of nearest neighbors to the query article

①1-NN algorithm

input:

query article: **x**q

corpus of documents: **x**1, **x**2, …, **x**N

output: most similar article **x**NN

**x**NN = min\_distance(**x**q, **x**i)

**algorithm:**

Initialize Dist2NN = infinity, closest\_document = empty set

For i = 1, 2, …, N:

compute: δ = distance(document i from corpus, query document)

if δ < Dist2NN:

closest\_document = document i

Dist2NN = δ

Return most similar document closest\_document

②k-NN algorithm

input:

query article: **x**q

corpus of documents: **x**1, **x**2, …, **x**N

output: List of k similar articles **x**NN = {**x**NN1, …, **x**NNk}

For all **x**i not in **x**NN, distance(**x**i, **x**q) >= max\_distance(**x**NNj, **x**q)

**algorithm:**

Initialize Dist2kNN = sort(δ1, …,δk),

cloest\_documents = sort(**x**1,…,**x**k)

For i = k+1, …, N:

compute: δ = distance(document i, query document)

if δ < Dist2kNN[k]:

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

remove furthest house and shift queue:

closest\_document[j+1:k] = closest\_document[j:k-1]

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

set Dist2kNN[j] = δ and closest\_document[j] = **x**i

Return most similar document closest\_document

(2) Document representation

①word count document representation

bag of words model

ignore order of words

count number of instances of each word in vocabulary

→word count vector

issue with word counts- rare words

common words like ‘the’, ‘a’ does not matter much

dominate rare words like ‘football’ matter much more

②TF-IDF document representation

(Term Frequency Inverse Document Frequency)

emphasizes **import words**

appears frequently in document (**common locally**)-

Term frequency = word counts

appears rarely in corpus (**rare globally**)-

Inverse doc freq. =

trade off between local frequency vs. global rarity: **tf \* idf**

(3) Distance metrics

defining notion of closest:

① Euclidean distance

in 1D, just Euclidean distance: 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

why weighting different features

some features are more relevant than others (e.g. title, abstract)

some features vary more than others:

for feature j: 1/ (maxi(**x**i[j])- mini(**x**i[j]))

→**Scaled Euclidean distance**

ai: weights (zero weight means not including this weight during weight selection)

② Cosine similarity

Another natural inner product measure

element-wise product between **x**q and **x**i

Similarity = **x**iT**x**q =

→ Cosine similarity (normalize)

equal to the cosine of the angle between the two points on the plane

the lower the θ is, the higher the similarity becomes

define **distance = 1 - similarity**

cosine similarity is invariant to the length of documents due to normalizing

but normalizing can make dissimilar objects appear more similar

→common comprise:

just cap maximum word maximum

③ Other distance metrics

e.g. Mahalanobis, rank-based, correlation-based, Manhattan, Jaccard, Hamming

We can use different distance measures over different subsets of your features

if we want sth. invariant to length for some numerical features: use Euclidean distance, otherwise, use cosine similarity

(4) Complexity of brute-force search

given a query point, scan through each point

O(N) distance computations per 1-NN query

O(Nlogk) per k-NN query

(5) KD-tree representation

specific data structure for **efficiently** representing the data

structured organization of documents

recursively partitions points into axis aligned boxes

enables more efficient pruning of search space

works ‘well’ in ‘low-medium’ dimensions(features)

**KD-tree construction**

start with a list of d-dimensional points

split points into a groups based on a threshold(split value) of a split dimension

e.g. **x**[1] > 0.5

recurse on each group separately

e.g. for **x**[1] > 0.5, split further on **x**[2] > 0.1

and for **x**[1] <= 0.5, split further on **x**[2] > 0.4

continue splitting points at each set, create a binary tree structure

each leaf node contains a list of points

each node stores three information:

split dimension

split value

bounding box that is as small as possible while containing points

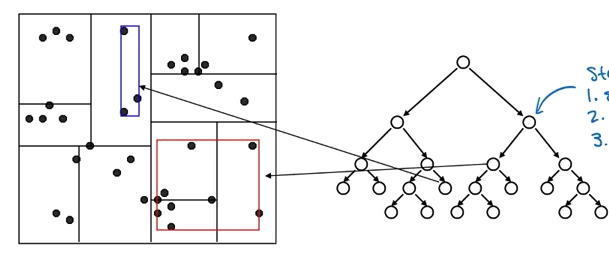
KD-tree construction choices:

Use heuristics to make splitting decisions-

which dimension do we split along?- widest dimension or alternate

which value do we split at?- median or center point of box

when do we stop?- fewer than n points left/ box hits minimum width



(6) NN search with KD-trees

traverse tree looking for nearest neighbor to query point

1.start by exploring leaf node containing query point

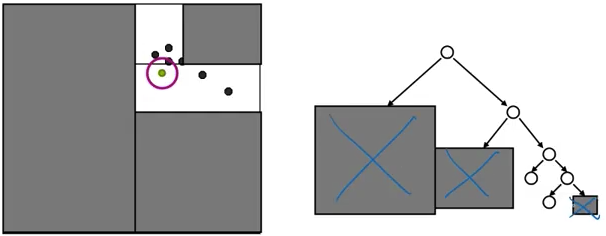
2.compute distance to each other point at leaf node

but the nearest neighbor may not in this leaf node

3.backtrack and try other branch at each node visited, and compute distance, update NN

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

if distance to bound box > distance to NN so far, then prune this box



(7) Complexity of NN search with KD-trees

for (nearly) balanced, binary trees…

construction

size: 2N-1 nodes if 1 data point at each leaf- O(N)

depth: O(logN)

median + send points left right: O(N) at every level of the tree

construction time: O(NlogN)

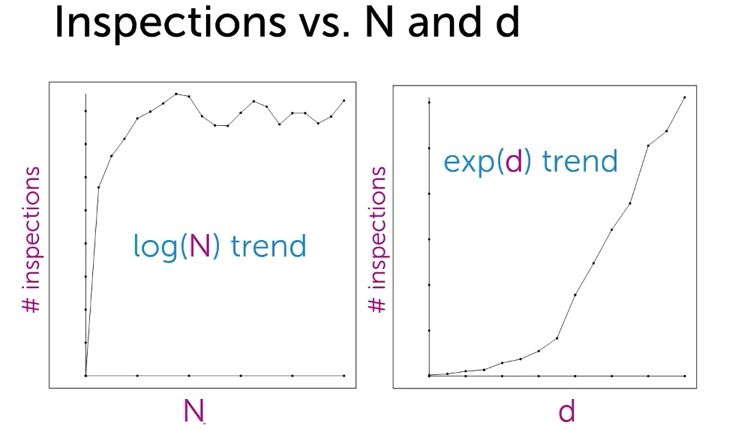
1-NN query

traverse down tree to starting point: O(logN)

maximum backtrack and traverse: O(N) in the worst case

complexity range: O(logN) to O(N) (worst case)

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



→Complexity for N queries

run N queries asking for NN to each doc

brute force 1-NN: O(N2)

kd-trees: O(NlogN) to O(N2) – potentially very large savings

8) k-NN with KD-trees

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

(9) Approximate k-NN search using KD-trees

**before:** prune when distance to bounding box > r

**now:** prune when distance to bounding box > r/alpha

alpha > 1

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

(110) Other things about KD-trees

there are tons of variants of kd-trees, thus may just be as slow as brute-force search

distance metric and data representation are still crucial to answer returned

for both KD and BF, high-dim spaces are hard

(11) Limitations of KD-trees

KD-trees has some limitations:

non-trivial to implement efficiently

problems with high-dimensional data

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 2d nodes

solution→ moving away from exact NN search

approximate neighbor finding

focus on methods that provide good probabilistic guarantees on approximation

(12) Locality Sensitive Hashing (LSH)

an alternative to KD-trees

**method:**

simply ‘bin’ data into 2 bins

make a decision boundary, and the two sides of the boundary are assigned to bin 0 and 1

only search nearest neighbor of the query point in its bin

do this using a hashtable (a dictionary, keys are bin [0, 1], values of the key are the indices of the data point that have that bin value)

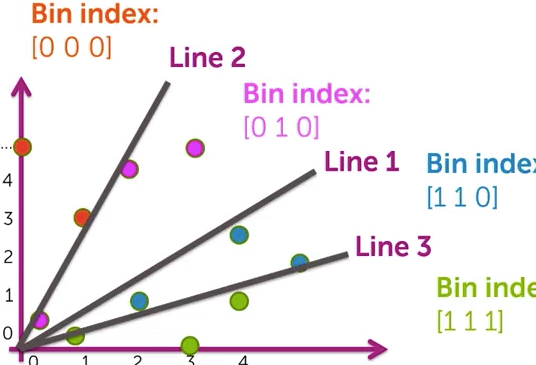
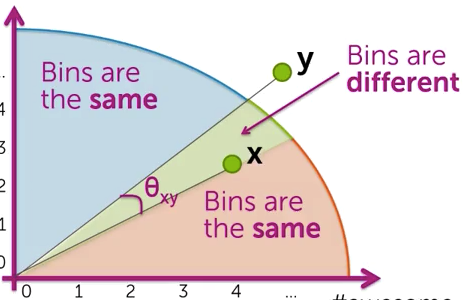
**challenges:**

①find good line to partition points

if we define line randomly

if x, y are close (angel is θxy), we want binned values to be the same

if θxy is small enough, a random line is unlikely to place x,y into separate bins



②large computational cost (bins might contain many points, so still searching over large set for each NN query)

we can reduce the number of points examined per query by defining more bins

trade off between accuracy and speed

greater chance that searching one of multiple bins, we do not find true NN

③poor quality solution (points close together get split into separate bins)

we can improve search quality by searching neighboring bins

neighboring bin: flip 1 bit (e.g. [010] to [000] and [110] on upper right plot)

but doing this we assume there is at most only one line between query and NN

algorithm: continue searching until computational budget is reached or quality of NN good enough

**LSH conclusion**

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 in high dimensions**

draw random planes/hyper planes crossing the original point

cost of binning points in d-dimensional space:

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

above calculation is one-time cost, which will be offset if many queries made

Week 2 Clustering with Kmeans

(1) Clustering

e.g. structure documents by topics

goal:

discover groups(clusters) of related articles

learn user preference (user read set of clustered documents and give feedback)

clustering is an **unsupervised** task

labels(clusters) are unknown

uncover cluster structure from input alone

input: docs as vectors **x**i

output: cluster labels zi

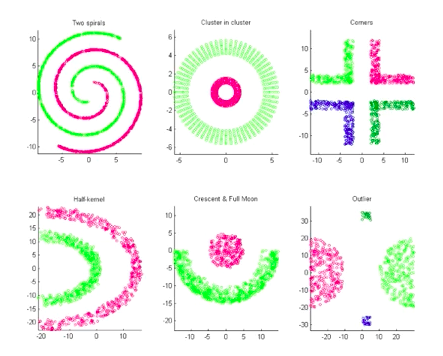
cluster defined by **center** and **shape/spread**

assign observation **x**i to cluster k if

score under cluster k is higher than under others

for simplicity, often define score as distance to cluster center (ignoring shape)

clusters can be challenging as below



(2) K-means algorithm

score = distance to cluster center (smaller is better)

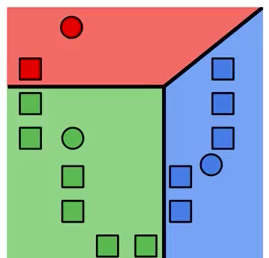
0. initialize cluster centers μ1, μ2, …,μk

1. assign observations to closest cluster center

return the index j of the cluster whose center is closest to xi

zi is the inferred label for observation i (not the given label as supervised)

to visualize, you can create a Voronoi tessellation



2. revise cluster centers as mean of assigned observations

nj: number of observations in cluster j

sum over all observation i such that zi = j (observation i is in cluster j)

3. repeat 1+2 until convergence

(3) K-means as coordinate descent

kmeans algorithm loop over step 1 and 2

we can rewrite step 1 and 2 as follows:

1.

2.

thus we need alternating minimization 1. z given μ and 2. μ given z

→coordinate descent

convergence of k-means

local optimum

has uncertainty due to different initialization

(4) Smart initialization via k-means++

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 observation **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(**x**)2

more likely to select a data point as a cluster center if that data point is far away

4. repeat step 2 and 3 until k centers have been chosen

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

(5) Assessing the quality

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

we can assess the quality of given clustering using the above measurement

lower is better, and less heterogeneous (lower heterogeneity)

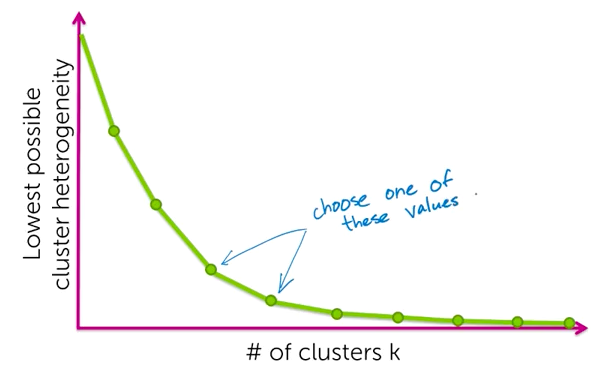
if k is larger, this quality tends to be better, but will cause overfitting

to choose k (heuristicly):

do a bunch of different restarts for every value of k

build a curve between lowest possible cluster heterogeneity and k

choose one of these values of k located at the elbow of the curve



(5) MapReduce

powerful and broadly used framework for parallel and distributed implementation of algorithms

e.g. counting words in lots of documents

①on a single processor:

count = {}

for d in documents:

for word in d:

count[word] += 1

②naïve parallel word counting (on multiple processors)

suppose we have evenly distributed data

we count occurrences in sets of documents separately

then combine them together

→data parallel

cycle through all the words in our vocabulary to sum the counts up

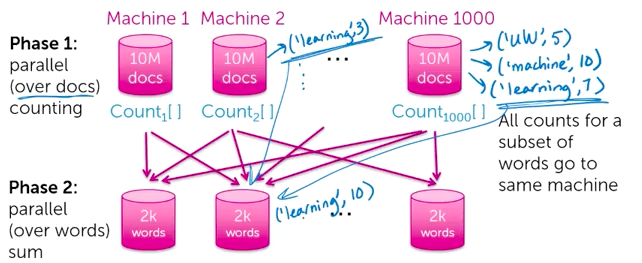
③counting words in parallel & merging tables in parallel

1. generate pairs(word, count) in parallel

2. merge counts for each word in parallel

to map words to machine using a hash function:

h(word\_index) → machine index

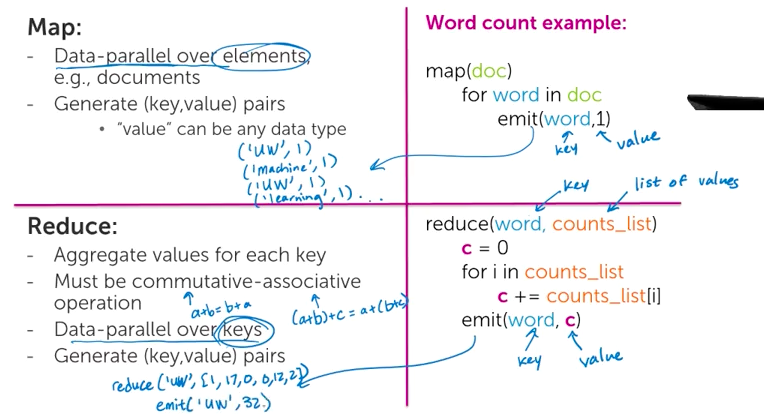


**General MapReduce abstraction**

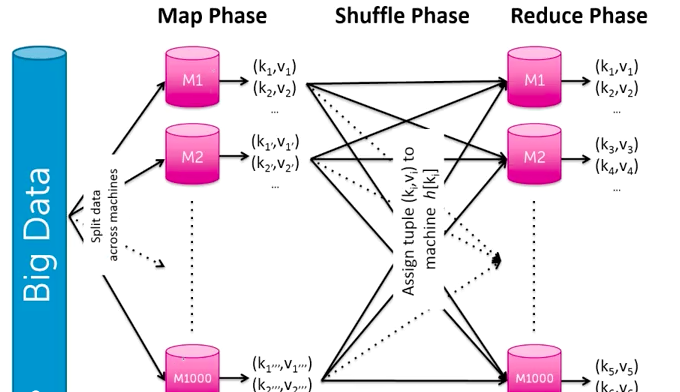
Map: computing paralleled

Reduce: aggregation values for each key

must be commutative-associative operation: a+b=b+a, (a+b)+c=a+(b+c)



**MapReduce execution overview**



improving MapReduce performace: **Combiners**

naïve implementation of MapReduce is very wasteful in communication during shuffle.

combiner: Perform reduce locally in Map phase before communicating for global reduce.

works because reduce is commutative-associative

(6) MapReduce for k-means

**scaling up k-means using MapRduce:**

1. **classify**: assign observations to closest cluster center

**Map**: for each data point, given({μj}, **x**i), emit(zi, **x**i)

map([μ1, μ2,…, μk], **x**i)

emit(zi, **x**i)

e.g. emit(2, [17, 6, 0, 1, 7, 0, 0, 5])

datapoint [17, 6, 0, 1, 7, 0, 0, 5] is assigned to cluster 2

2. **recenter**: revise cluster centers as means of assigned observations

**Reduce**: average over all points in cluster j (zi=k)

reduce(j, x\_in\_clusterj: [**x**1, **x**3, …, ])

sum = 0

count = 0

for **x** in x­­\_in\_clusterj

sum += **x**

count += 1

emit(j, sum/count)

other practival considerations for k-means with MapReduce:

k-means needs an iterative version of MapReduce (not standard formulation)

mapper needs to get data point and all centers

a lot of data

a better implementation is mapper gets may data points

(7) Other applications of clustering

clustering images

structuring web search results

brain scan and disease

group products on Amazon

discovering similar neighborhood

Week 3 Mixture Models

mixture models - model-based clustering

(1) Probabilistic clustering models

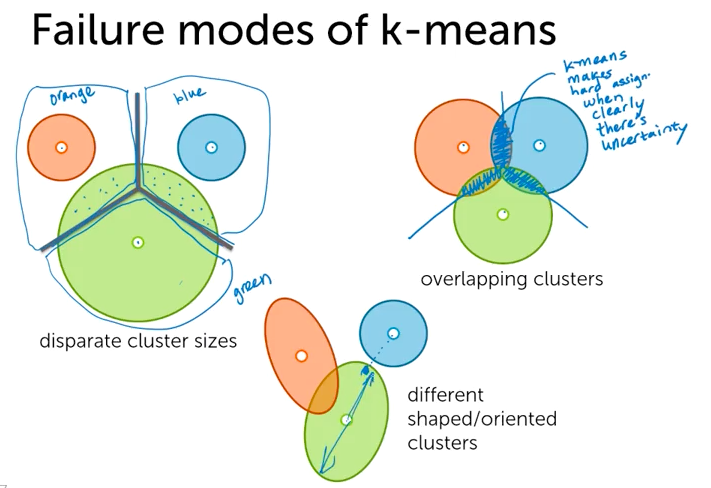
motivation:

hard assignments made by k-means don’t tell full story

we want to capture the uncertainty

k-means only consider the cluster center when assigning data point

(it assumes spherically symmetric cluster or same axis-aligned ellipses)



solution: **mixture model**

provide soft assignments of observations to clusters (uncertainty in assignment)

accounts for cluster shapes not just centers

enables learning weightings of dimensions

(2) Mixture models

example: clustering images

provide groups, but not category names

→**simple image representation**

consider average red, green, blue pixel intensities

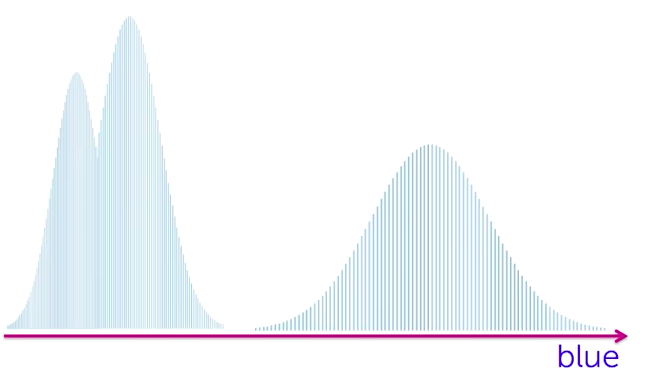
e.g. [R=0.05, G=0.7, B=0.9]

we can look at the blue intensity distribution over all images in the dataset

the following histogram indicate three underlying groups

each group follows Gaussian distribution (refer to (3))

e.g. sunset, forest, clouds



we can also look at other color dimension and clustering over all dimensions

e.g. sunset and forest images are not distinguishable in blue dimension, but they are in red dimension

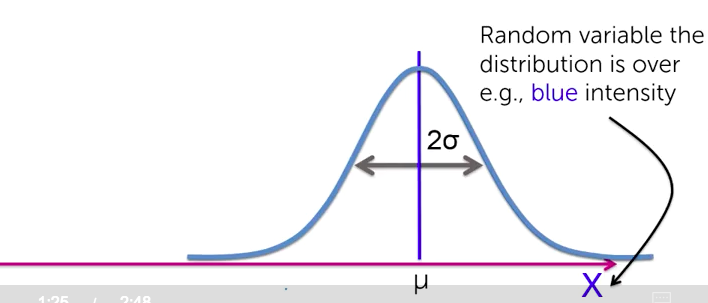
(3) Gaussian distribution (Normal distribution)

for each dimension of R,G,B vector, and each image type, we assume a **Gaussian distribution** over color intensity.

**①1D Gaussian**

fully specified by mean μ and variance σ2

notated as **N(x | μ, σ2)**



**②2D Gaussians**

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

**μ** = [μblue, μgreen], mean centers the distribution in 2D

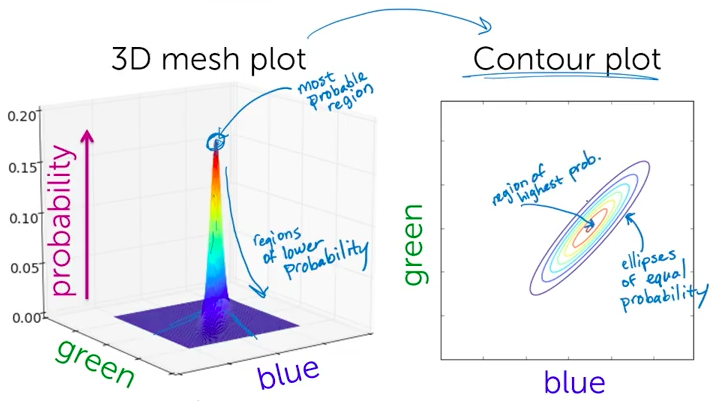
, covariance determines orientation and spread

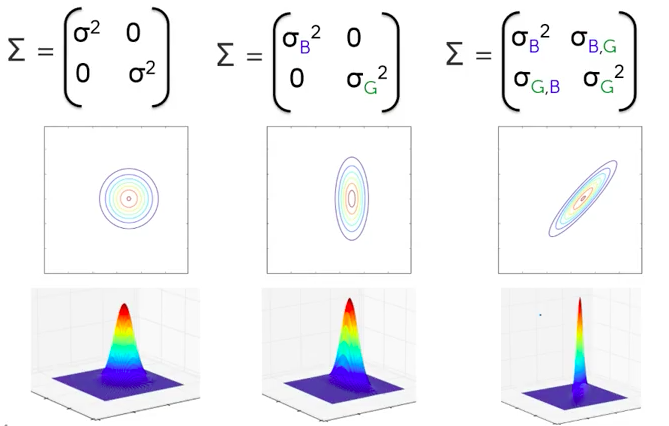
special type of covariance structures:

diagonal covariance- no correlation between blue and green

nD Gaussians notated as N(**x** | **μ**, Σ)

Σ: n\*n matrix





(4) Mixture of Gaussians

we assume an entire Gaussian with correlation structure for all three dimensions- RGB

take each category-specific distribution and simple/weighted average them together, the resulting density is going to be an average of each one of them.

simple average suppose each one category appears in equal proportion

weighted average- if one category is more likely to appear, then higher weight

associate a weight πk (between(0,1)) with each Gaussian component

e.g. π = [π1=0.47, π2=0.26, π3=0.27]

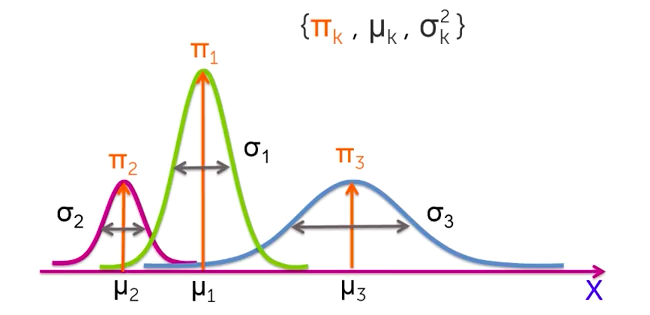
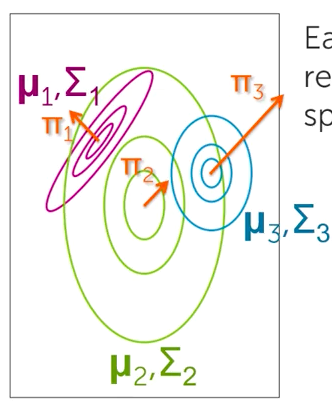
the probability that an image belongs to cluster k: p(zi=k)= πk

given observation **x**i is from cluster k, the likelihood of seeing **x**i:

P(xi | zi=k, μk, Σk) = N(xi |μk, Σk)

e.g. in the cloud category, how likely are we seeing this picture

each mixture component represents a unique cluster specified by {πk, μk, σk2} in 1D and {πk, μk, Σk} in higher dimension

(3) Document clustering with mixtures of Gaussians

**x**i = tf-idf vector

embedding our documents to the space with vocabulary size v dimensions

each cluster has {πk, μk, Σk}

full covariance matrix Σk has V2 dims and V(V+1)/2 unique parameters to learn

we can restrict to diagonal covariance- only look at axis aligned ellipses

we can learn weights on dimensions and learn cluster-specific weights on dimensions

(4) Three conditions for inferring assignments/ cluster parameters

input: a set of unlabeled, observed data

output: a set of soft assignments per data point (an allocation of that data point to each one of the clusters)

①Condition 1: what if we knew the cluster parameters {πk, μk, Σk}

compute responsibility vector for each observation i:

rik: the responsibility that cluster k takes for observation i

p(zi=k): probability of assignment to cluster k

given model parameters and observed value xi

πk: initial probability of being from cluster k

N(…): how likely is the observed value **x**i under this cluster assignment

Bayesian inference: prior \* likelihood

→ given cluster parameters, soft assignments are plausible

②Condition 2: imagine we knew the cluster (hard) assignments zi

imagine we know the cluster assignments

estimation problem decouples across clusters

what we have: **x**i and their cluster assignments

step1: split the data based on the hard assignment of cluster

step2: maximum likelihood estimation (MLE)

estimate {πk, μk, Σk} given data assigned to cluster k

find parameters that maximize the score(likelihood)

mean estimation: (the same as sample mean)

covariance estimation: (as sample estimation)

cluster proportion estimation: (this is true for any distribution)

→ given hard assignments, estimating cluster parameters is straightforward

③Condition 3: what can we do with just soft assignments rij

Instead of having a full observation **x**i in cluster k, we just have soft assignment rik

MLE from soft assignments

just like in boosting with weighted observations

we assign a set of weights that correspond to the responsibility on each obs

compute total weight in each cluster (effective # of observations in each cluster)

form a cluster specific data table (e.g. cluster 1 weights table etc.)

compute cluster parameter estimates with weights on each row operation

→ given soft assignment, we can easily estimate cluster parameters

\*\* for hard assignments, we can view rik as only taking 1 or 0 as 1 indicating i in k



(5) Inferring soft assignments with expectation maximization (EM)

**Two steps:**

1. E-step

estimate cluster responsibilities given current parameter estimates

2. M-step

maximize likelihood over parameters given current responsibilities

(calculating as condition 3 above)

**EM algorithm:**

initialize cluster parameters

compute responsibilities rik-hat

maximize the likelihood given our soft assignments rik from last iteration

then recomputed responsibilities based on new cluster parameters

loop… until convergence

we will have uncertainty in cluster even in final assignments

EM is a coordinate-ascent algorithm

can equate E- and M-steps with alternating maximizations of an objective function

converges to a local mode

**key concerns:**

1. initialization:

many ways to initialize and it’s important for convergence rates and quality of local mode found

e.g. choose k observations at random to define k “centroids”, assign other observations to nearest centroid to form initial parameter estimates

e.g. pick centers sequentially to provide good coverage of data like in k-means++

e.g. initialize from k-means solution

e.g. grow mixture model by splitting(and sometimes removing) clusters until k clusters are formed

2. overfitting of MLE:

maximizing likelihood can overfit to data

e.g. at k=5 example, with one obs assigned to cluster1, and all others to cluster2

e.g. for doc assignment, all docs in certain cluster agree on count of a specific word: μk=0, σk2=0.

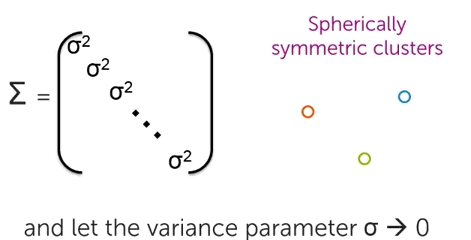


simple fix for overfitting: don’t let variances get to 0, add small amount of diagonal of covariance estimate.

alternatively, we can take Bayesian approach and place prior on parameters. this will smooth the cluster proportion and means and the covariances.

3. relationship to k-means

for spherically symmetric clusters (Σ matrix with only diagonal), if we let the variance parameter σ gets to 0, then the clustering result turned out to be k-means results (in E-step, we get responsibilities just 0 or 1, and in M-step, data points will be assigned just based on its distance to centers)



Week 4 Latent Dirichlet Allocation

(1) Mixed membership models

clustering model specifies each doc belongs to 1 certain topic

mixed membership model: want to discover a set of memberships

(2) An alternative document clustering model- Bag of words

instead of using a single tf-idf representation, we apply bag-of-words representation

bag-of-words:

an unordered set of words with multiple occurrences of a unique word if that word appears multiple times in the document

formally called a ‘multiset’

prior probability that doc i is from topic k:

p(zi=k) = πk

likelihood:

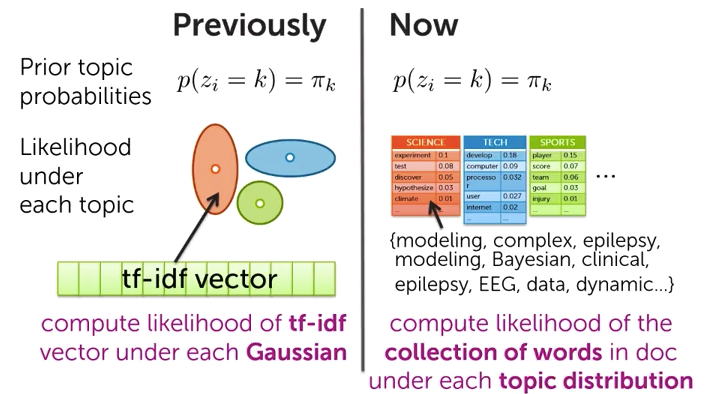
we have a topic probability vector over words

e.g. SCIENCE topic: experiment:0.1, text:0.08, clinical:0.01, epilepsy:0.002, …

every topic is going to be associated with a probability distribution over words in the vocabulary

we use the vector to score doc i over every topic

lastly, choose the topic using the prior and the likelihood



we can drive EM algorithm on it:

have multinomial likelihood of word counts (mw successes of word w)

result: mixture of multinomial model

(3) Latent Dirichlet Allocation (LDA)

LDA is a mixed membership model:

1. introduce the same set of topic-specific vocabulary distribution

2. one topic indicator ziw per word in doc i

w th word of doc i

instead of assigning the entire document, we assign each word to a topic

3. each word gets scored under its topic ziw

all words assigned to “tech” topic in a doc are get scored together

4. get distribution on prevalence of topics in document

for each doc, we now have a score for each topic

πi = [πi1 πi2 … πik]

πik: the score of doc i on topic k

LDA inputs:

set of words per doc for each doc in corpus

LDA outputs:

corpus-wide topic vocab distributions

topic assignments per word

topic proportions per doc

interpreting LDA outputs:

examine coherence of learned topics

-what are top words per topics

-do they form meaningful groups

-use to post-facto label topics(e.g. science, tech, sports)

doc-specific topic proportions can be used to

-relate documents

-study user topic preferences

-assign docs to multiple categories

(4) Gibbs sampling- an inference algorithm for LDA

EM algorithm is a poor solution because of the sheer number of parameters that we have to learn

Normally LDA is specified as a **Bayesian model**

account for uncertainty in parameters when making predictions

naturally regularizes parameter estimates in contrast to MLE

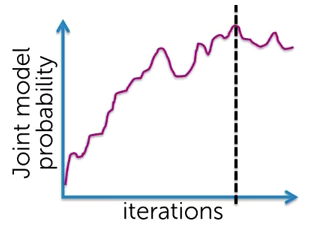
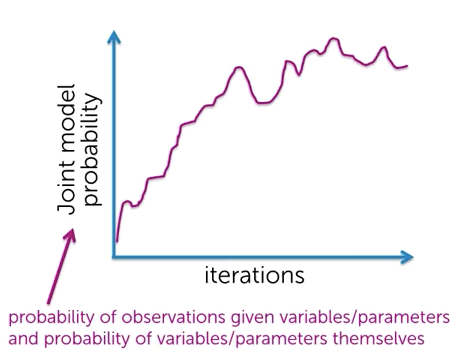
**Gibbs sampling**- a Bayesian inference approach, iterative random hard assignment

benefits:

typically intuitive updates

very straightforward to implement

This is not an optimization algorithm, it is not promised that the joint model probability will always goes up. This is because that it is a randomized algorithm, we’re actually intending to explore the space of possible solutions, rather than converge just to a single point. And it will eventually provide ‘correct’ Bayesian estimates.



Gibbs sampling predictions:

make prediction for each snapshot of randomly assigned variables/parameters (full iteration)

average predictions for final result

parameter or assignment estimate:

look at snapshot of randomly assigned variables/parameters that maximizes “joint model probability”- maximum a posterior parameter estimate.

Assignment variables and model parameters are treated similarly in Gibbs sampling

**Standard Gibbs sampling steps**

iteratively draw variable/parameter from conditional distribution having fixed:

all other variables/parameters

values randomly selected in previous rounds

changes from iter to iter

observations

always the same values

in LDA, we have topic vocabulary distributions and document specific assignment variables of words to topics and the document specific topic proportion factors

**step1:**

randomly reassign all ziw based on

-doc topic proportions

-topic vocab distributions

draw randomly from responsibility vector [riw1 riw2 … riwK]

riwK: probability of assigning word w in doc i to topic K

riwK = πik (prior) \* p(w|ziw=K)(likelihook) / Σj=1k πij\*p(w|ziw=j)

**step2:**

randomly reassign doc topic proportions based on assignments ziw in current doc

we also need to regularize by the Bayesian prior

**step3:**

repeat for all docs

we repeat sampling the word assignment variables and the topic proportions for each document in our entire dataset

**step4:**

randomly reassign topic vocab distributions based on assignments ziw in entire corpus

e.g. for the word EEG, we can think of how many times was the word EEG assigned to topic one, and we can use that information to inform us of how probable EEG is under topic one.

**repeat steps 1-4 until max iter reached**

ps: we can parallel this process since ziw does not rely on other word assignment

(5) Collapsed Gibbs sampling in LDA

based on special structure of LDA model, we can just sample indicator variables ziw

no need to sample other parameters (corpus-wide topic vocab distributions and per-doc topic proportions)

this often leads to much better performance because examining uncertainty in smaller space

just randomly reassign ziw based on current assignments zjv of all other words in document and corpus

never draw topic vocab distributions or doc topic proportions

from best sample (maximizing joint collapse probability) of {ziw}, we can infer

1. topics from conditional distribution (need corpus-wide info)

2. embedding new documents

fix topics based on training set collapsed sampling

run uncollapsed sampler on new docs only

Week 5 Hierarchical Clustering

(1) Why hierarchical clustering

avoid choosing number of 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

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

(2) Two mean types

Divisive (top-down)

start with all data in one big cluster and recursively split

e.g. recursive k-means

Agglomerative (bottom-up)

start with each data point as its own cluster

merge clusters until all points are in one big cluster

e.g. single linkage

(3) Divisive clustering

we take each cluster at a given granularity, dig down and specify it as a set of clusters itself

one very straightforward approach is to recursively apply k-means algorithm

split into k clusters, then for each cluster, split it further

divisive choice to be made:

which algorithm to recurse

how many clusters per split (k)

when to split v.s. 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

(4) Agglomerative clustering

**e.g. single linkage**

1. initialize each point to be its own cluster

2. define distance between clusters to be:

distance(C1,C2) = min d(xi, xj) with xi in C1 and xj in C2

3. merge the two closest clusters

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

through this process, we get clusters of clusters, i.e. clusters defined at different granularities

(5) Dendrogram

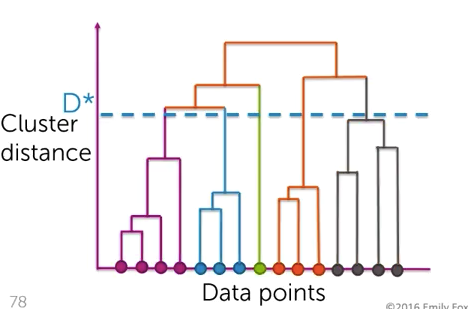
e.g. dendrogram for agglomerative clustering

x-axis shows data points (carefully ordered)

y-axis shows distance between pair of clusters

path shows all clusters to which a point belongs and the order in which clusters merge

we can set a D\*, and cut along the y-axis of the dendrogram, the every branch that crosses D\* becomes a separate cluster.



(6) Agglomerative clustering details

choices to be made

distance metric: d(xi, xj)

linkage function e.g. min d(xi, xj)

where and how to cut dendrogram

more on cutting dendrogram

for visualization, smaller number of 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 ‘incomsistency’ rather than distance

no cutting method is incorrect, some are just more useful

computational considerations

computing all pairs of distance is expensive- brute force with O(N2log(N))

smart implementations use triangle inequality to rule out pairs

best known algorithm is O(N2)

statistical issues

chaining: distant points clustered together if there is a chain of pairwise close points between

other linkage functions can be more robust, but restrict the shapes of clusters that can be found

e.g. complete linkage- max pairwise distance between clusters

e.g. ward criterion- min within-cluster variance at each merge

(7) Hidden Markov models (HMMs)

so far, we only looked at clustering unordered data

data index does not influence clustering

what if we have time series data?

e.g. repeated patterns of dance transitions of bees

→hidden Markov model (HMM)

as in mixture model, every observation xt is associated with cluster assignment variable zt, each cluster has a distribution over observed values

difference from mixture model: probability of (zt=k) depends on previous cluster assignment zt-1

**Inference in HMMs**

learn MLE of HMM parameters using EM algorithm = Baum Welch

infer MLE of state sequence given fixed model parameters using dynamic programming = Viterbi algorithm

infer soft assignments of state sequence using dynamic programming = forward-backward algorithm