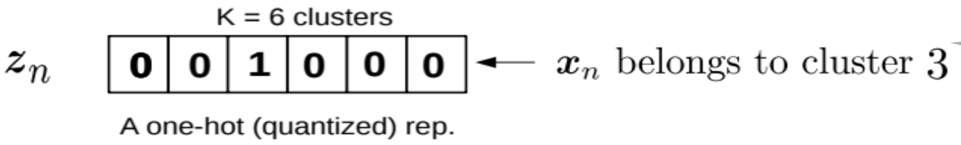
Unsupervised Learning: Clustering

CS771: Introduction to Machine Learning

Unsupervised Learning

- It's about learning interesting/useful structures in the data (unsupervisedly!)
- There is no supervision (no labels/responses), only inputs $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$
- Some examples of unsupervised learning
  - Clustering: Grouping similar inputs together (and dissimilar ones far apart)
  - Dimensionality Reduction: Reducing the dimensionality of inputs
  - Estimating the probability density of inputs (which distribution $p(\boldsymbol{x}|\theta)$ "generated" the inputs)
- Most unsup. learning algos also learn a new feature representation of inputs, e.g.,
  - Clustering gives a one-hot "quantized" representation $\boldsymbol{z}_n$ for each input $\boldsymbol{x}_n$

Some clustering algos learn a soft/probabilistic clustering in which $\boldsymbol{z}_n$ will be be probability vector that sums to 1 (will see later)



K = 6 clusters

z_n $\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}$ ← x_n belongs to cluster 3

A one-hot (quantized) rep.

Assuming each input belongs deterministically to a single cluster

  - Dim-red gives $\boldsymbol{z}_n$, a lower-dim representation of $\boldsymbol{x}_n$

Unsupervised Learning as Representation Learning

- Unsupervised Learning can be seen as learning a function f s.t.

$$\boldsymbol{x}_n \approx f(\boldsymbol{z}_n) \qquad \forall n$$

- $\boldsymbol{z}_n$ denotes a new representation of $\boldsymbol{x}_n$ (note: $\boldsymbol{z}_n$ can have lower/higher dim than $\boldsymbol{x}_n$)
- Different unsupervised learning algos differ in terms of
  - The nature of the function f
  - The nature of the representation $\boldsymbol{z}_n$
- The goal is to learn f and $\boldsymbol{z}_n$ by minimizing some distortion function

How well we can reconstruct $\boldsymbol{x}_n$ from f and $\boldsymbol{z}_n$

$$(\hat{f}, \widehat{\mathbf{Z}}) = \text{argmin}_{f,\mathbf{Z}} \sum_{n=1}^{N} \ell(\boldsymbol{x}_n, f(\boldsymbol{z}_n))$$

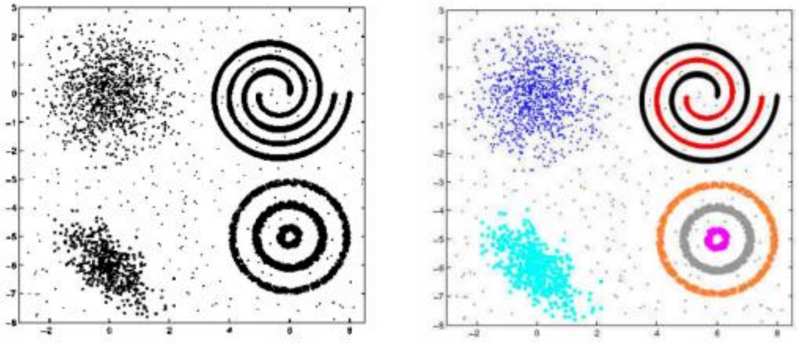
Can additionally also have regularizers/constraints on f and $\boldsymbol{z}_n$

Clustering

In some cases, we may not know the right number of clusters in the data and may want to learn that (technique exists for doing this but beyond the scope)

- Given: N unlabeled inputs $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N$; desired no. of partitions K
- Goal: Group the examples into K “homogeneous” partitions

In addition to partitioning these N inputs, we may also want to predict which partition a new test point belongs to



(a) Input data

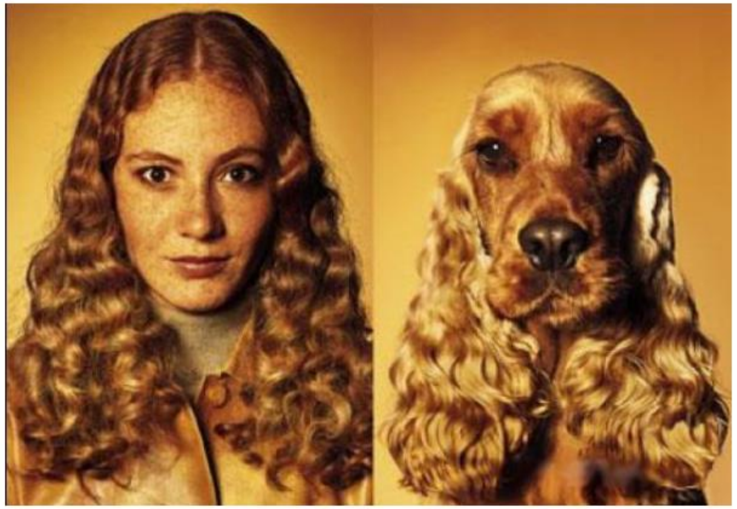
(b) Desired clustering

Picture courtesy: “Data Clustering: 50 Years Beyond K-Means”, A.K. Jain (2008)

- Loosely speaking, it is classification without ground truth labels of training data
- A good clustering is one that achieves
  - High within-cluster similarity
  - Low inter-cluster similarity

Similarity can be Subjective

- Clustering only looks at similarities b/w inputs, since no labels are given
- Without labels, similarity can be hard to define

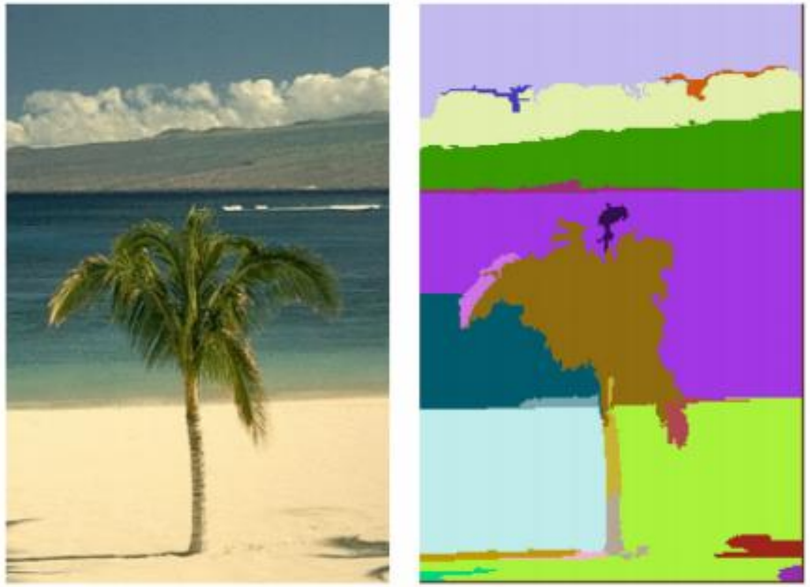


- Thus using the right distance/similarity is very important in clustering
- In some sense, related to asking: “Clustering based on what”?

Picture courtesy: http://www.guy-sports.com/humor/videos/powerpoint presentation dogs.htm

Clustering: Some Examples

- Document/Image/Webpage Clustering
- Image Segmentation (clustering pixels)



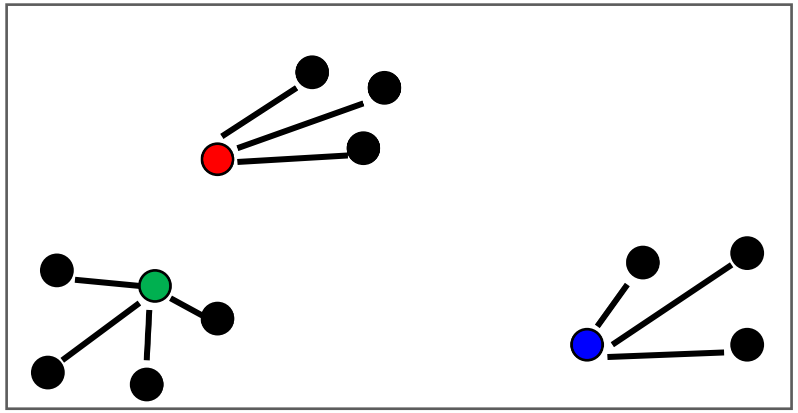
- Clustering web-search results
- Clustering (people) nodes in (social) networks/graphs
- .. and many more..

Picture courtesy: http://people.cs.uchicago.edu/~pff/segment/

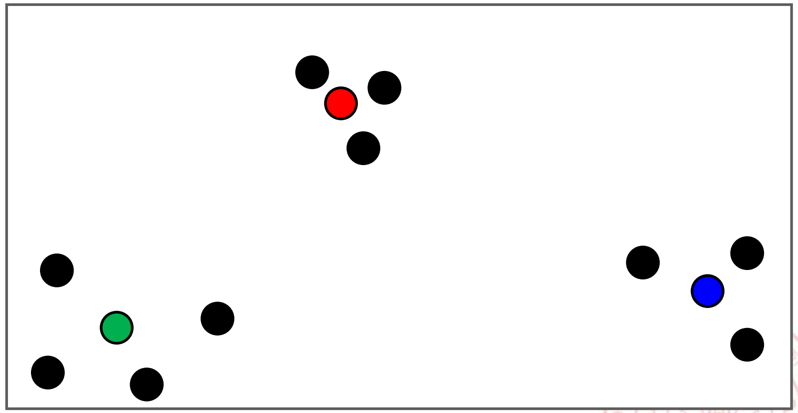
K-means Clustering

- Based on the following two steps (applied in alternating fashion until not converged)

1. Assign each input to the current closest mean



1. Recompute (update) the means



- At the very beginning, the means needs to be initialized (at random locations or using other schemes that we will see later)

K-means Clustering

Similar to LwP but in LwP the labels are known so the means can be computed in a single step without iterating multiple times

Good initialization matters (some methods exist to pick good initialization, e.g., K-means++)

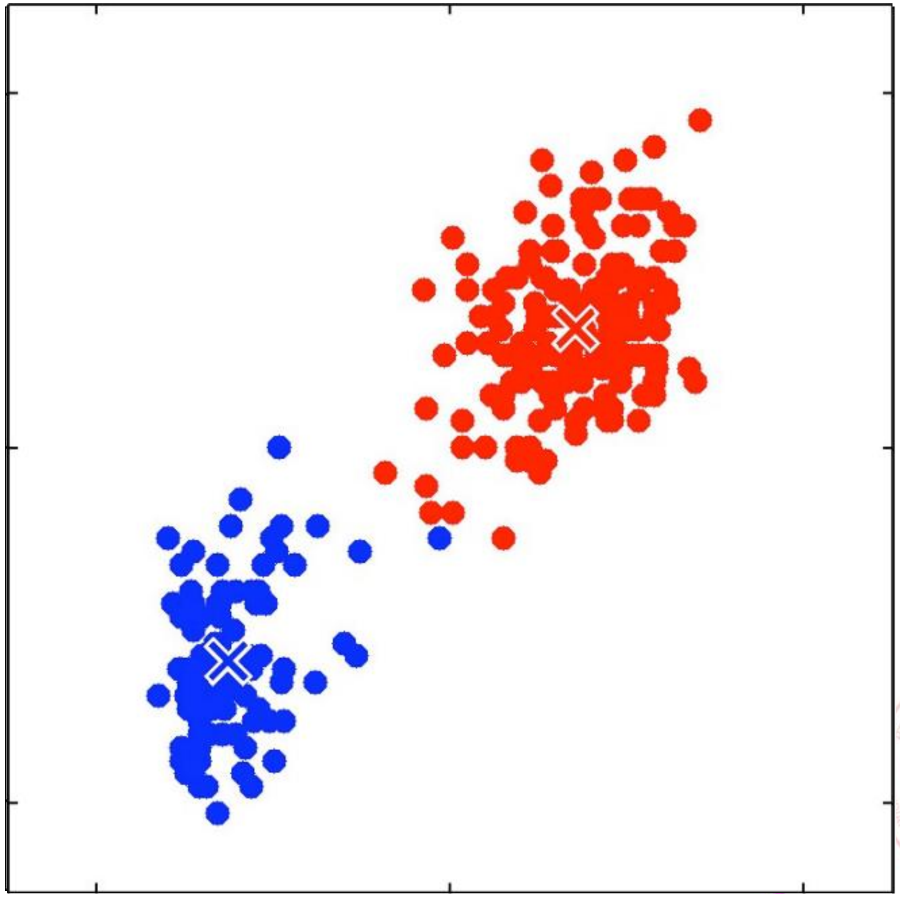
1. Randomly initialize K locations (the means)

2. Using the current means, predict the cluster id for each input (closest mean)

3. Recompute the means using the predicted cluster id of each input

4. Go to step 2 if not converged

K-means clustering with $K = 2$



K-means (a.k.a. Lloyd's) Algorithm: Formally

- **Input:** N inputs $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$; $\boldsymbol{x}_n \in \mathbb{R}^D$; desired no. of partitions K
- **Desired Output:** Cluster ids of these N inputs and K cluster means $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$

K-means Algorithm

1. Initialize K cluster means $\mu_1, \ldots, \mu_K$
2. For $n = 1, \ldots, N$, assign each point $\mathbf{x}_n$ to the closest cluster

$$z_n = \arg\min_{k \in \{1,\ldots,K\}} ||\mathbf{x}_n - \mu_k||^2$$

3. Suppose $\mathcal{C}_k = \{\mathbf{x}_n : z_n = k\}$. Re-compute the means

$$\mu_k = \text{mean}(\mathcal{C}_k), \quad k = 1, \ldots, K$$

4. Go to step 2 if not yet converged

$z_n = k$ means $z_{nk} = 1$ in one-hot representation of z_n

K-means algo can also be seen as doing a compression by "quantization": Representing each of the N inputs by one of the $K < N$ means

Can be fixed by modeling each cluster by a probability distribution, such as Gaussian (e.g., Gaussian Mixture Model; will see later)

This basic K-means models each cluster by a single mean μ_k. Ignores size/shape of clusters

What Loss Function is *K*-means Optimizing?

- Define the distortion or "loss" for the cluster assignment of a single input $\boldsymbol{x}_n$

No labels here. We are measuring how much error we will incur if we assign x_n to its nearest cluster mean

$$\ell(\boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{z}_n) = \sum_{k=1}^{K} z_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$$

$\boldsymbol{z}_n = [z_{n1}, z_{n2}, \dots, z_{nK}]$ denotes a length K one-hot encoding of $\boldsymbol{x}_n$ and $z_{nk} = 1$ if x_n is assigned to cluster k

- Notation:
  - $\boldsymbol{X}$ is $N \times D$ matrix of inputs, $\boldsymbol{Z}$ is $N \times K$ matrix denoting cluster ids (each row is a one-hot $\boldsymbol{z}_n$)
  - $\boldsymbol{\mu}$ is $K \times D$ (each row contains the mean $\boldsymbol{\mu}_k$ of cluster $\boldsymbol{k}$)
- Total distortion over all inputs defines the K-means "loss function"

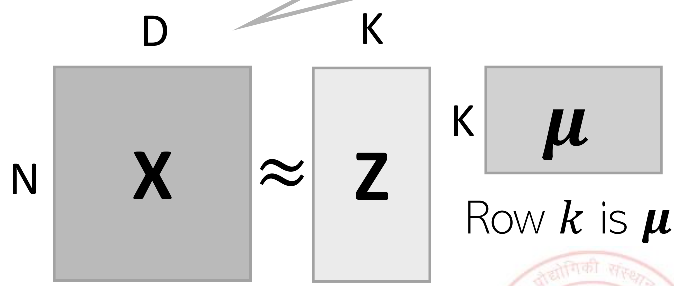
$$\ell(\boldsymbol{X}, \boldsymbol{\mu}, \boldsymbol{Z}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$$

$$= \|\boldsymbol{X} - \boldsymbol{Z}\boldsymbol{\mu}\|_F^2$$

K-means problem viewed as a matrix factorization problem

$\|.\|_F^2$ denotes the matrix Frobenius norm (squared norm for matrices)

$\boldsymbol{X}$ approximated by the product of $\boldsymbol{Z}$ and $\boldsymbol{\mu}$



Row $\boldsymbol{k}$ is $\boldsymbol{\mu}_k$

Row $\boldsymbol{n}$ is $\boldsymbol{z}_n$ (one-hot vector)

- The K-means problem is to minimize this objective w.r.t. $\boldsymbol{\mu}$ and $\boldsymbol{Z}$
  - Alternating optimization on this loss would give the K-means (Lloyd's) algorithm we saw earlier!

Optimizing the *K*-means Loss Function

- The K-means problem is

$$\{\hat{\mathbf{Z}}, \hat{\boldsymbol{\mu}}\} = \arg\min_{\mathbf{Z},\boldsymbol{\mu}} \mathcal{L}(\mathbf{X}, \mathbf{Z}, \boldsymbol{\mu}) = \arg\min_{\mathbf{Z},\boldsymbol{\mu}} \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\mathbf{x}_n - \mu_k||^2$$

- Can't optimize it jointly for $\boldsymbol{Z}$ and $\boldsymbol{\mu}$. Let's try alternating optimization for $\boldsymbol{Z}$ and $\boldsymbol{\mu}$

Alternating Optimization for *K*-means Problem

1. Fix $\boldsymbol{\mu}$ as $\hat{\boldsymbol{\mu}}$ and find the optimal $\mathbf{Z}$ as

Given the current estimates of the K means $\mu_1, \mu_2, \ldots, \mu_K$, find the optimal cluster ids $\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_N$ for all the inputs

$$\hat{\mathbf{Z}} = \arg\min_{\mathbf{Z}} \mathcal{L}(\mathbf{X}, \mathbf{Z}, \hat{\boldsymbol{\mu}}) \quad \text{(still not easy - next slide)}$$

2. Fix $\mathbf{Z}$ as $\hat{\mathbf{Z}}$ and find the optimal $\boldsymbol{\mu}$ as

Given the current estimates of the optimal cluster ids $\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_N$,compute the K means $\mu_1, \mu_2, \ldots, \mu_K$

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} \mathcal{L}(\mathbf{X}, \hat{\mathbf{Z}}, \boldsymbol{\mu})$$

3. Go to step 1 if not yet converged

Solving for Z

- Solving for $\boldsymbol{Z}$ with $\boldsymbol{\mu}$ fixed at $\widehat{\boldsymbol{\mu}}$

$$\hat{\mathbf{Z}} = \arg\min_{\mathbf{Z}} \mathcal{L}(\mathbf{X}, \mathbf{Z}, \hat{\mu}) = \arg\min_{\mathbf{Z}} \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\mathbf{x}_n - \hat{\mu}_k||^2$$

- Still not easy. $\boldsymbol{Z}$ is discrete and above is an NP-hard problem
  - Combinatorial optimization: K^N possibilities for $\boldsymbol{Z}$ ($N \times K$ matrix with one-hot rows)

- Greedy approach: Optimize $\boldsymbol{Z}$ one row (z_n) at a time keeping all others $\boldsymbol{z_n}$'s (and the cluster means $\mu_1, \mu_2, \ldots, \mu_K$) fixed

$$\hat{\mathbf{z}}_n = \arg\min_{\mathbf{z}_n} \sum_{k=1}^{K} z_{nk} ||\mathbf{x}_n - \hat{\mu}_k||^2 = \arg\min_{\mathbf{z}_n} ||\mathbf{x}_n - \hat{\mu}_{z_n}||^2$$

- Easy to see that this is minimized by assigning $\boldsymbol{x_n}$ to the closest mean
  - This is exactly what the K-means (Lloyd's) algo does!

Solving for $\boldsymbol{\mu}$

- Solving for $\boldsymbol{\mu}$ with $\boldsymbol{Z}$ fixed at $\widehat{\boldsymbol{Z}}$

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} \mathcal{L}(\mathbf{X}, \hat{\mathbf{Z}}, \boldsymbol{\mu}) = \arg\min_{\boldsymbol{\mu}} \sum_{k=1}^{K} \sum_{n:\hat{z}_n=k} ||\mathbf{x}_n - \mu_k||^2$$

- Not difficult to solve (each $\boldsymbol{\mu}_k$ is a real-valued vector, can optimize easily)

$$\hat{\mu}_k = \arg\min_{\mu_k} \sum_{n:\hat{z}_n=k} ||\mathbf{x}_n - \mu_k||^2$$

- Note that each $\boldsymbol{\mu}_k$ can be optimized for independently
- (Verify) This is minimized by setting $\hat{\boldsymbol{\mu}}_k$ to be mean of points currently in cluster k
  - This is exactly what the K-means (Lloyd's) algo does!

Convergence of *K*-means algorithm

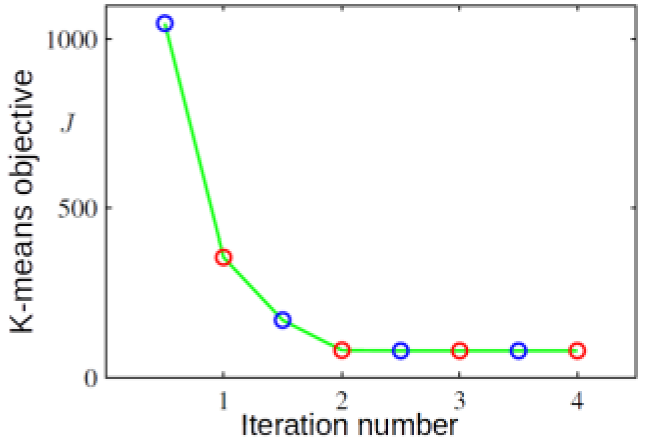
- Each step (updating $\boldsymbol{Z}$ or $\boldsymbol{\mu}$) can never increase the K-means loss
- When we update $\boldsymbol{Z}$ from $\boldsymbol{Z}^{(t-1)}$ to $\boldsymbol{Z}^{(t)}$

$$\mathcal{L}(\mathbf{X}, \mathbf{Z}^{(t)}, \boldsymbol{\mu}^{(t-1)}) \leq \mathcal{L}(\mathbf{X}, \mathbf{Z}^{(t-1)}, \boldsymbol{\mu}^{(t-1)}) \quad \text{because} \quad \mathbf{Z}^{(t)} = \arg\min_{\mathbf{Z}} \mathcal{L}(\mathbf{X}, \mathbf{Z}, \boldsymbol{\mu}^{(t-1)})$$

- When we update $\boldsymbol{\mu}$ from $\boldsymbol{\mu}^{(t-1)}$ to $\boldsymbol{\mu}^{(t)}$

$$\mathcal{L}(\mathbf{X}, \mathbf{Z}^{(t)}, \boldsymbol{\mu}^{(t)}) \leq \mathcal{L}(\mathbf{X}, \mathbf{Z}^{(t)}, \boldsymbol{\mu}^{(t-1)}) \quad \text{because} \quad \boldsymbol{\mu}^{(t)} = \arg\min_{\boldsymbol{\mu}} \mathcal{L}(\mathbf{X}, \mathbf{Z}^{(t)}, \boldsymbol{\mu})$$

- Thus the K-means algorithm monotonically decreases the objective



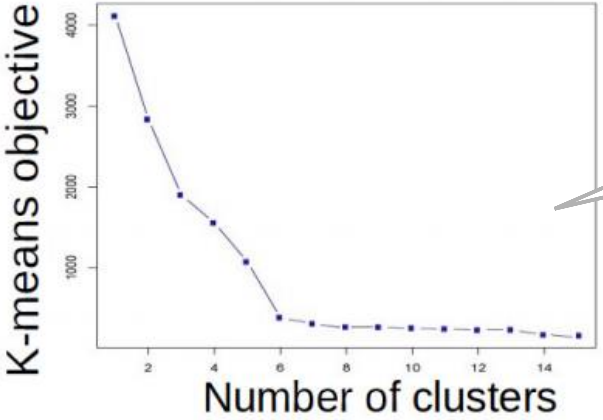
(blue: after Z updated, red: after μ updated)

K-means: Choosing *K*

- One way to select K for the K-means algorithm is to try different values of K, plot the K-means objective versus K, and look at the "elbow-point"

If we use $K = N$ then K-means algo can achieve smallest possible loss (zero!). Think why.

However, if we use $K = N$ it isn't a useful clustering



K=6 is the elbow point

- Can also information criterion such as AIC (Akaike Information Criterion)

$$AIC = 2\mathcal{L}(\hat{\mu}, \mathbf{X}, \hat{\mathbf{Z}}) + KD$$

KD is the Total number of parameters (note that we have K means, each is a vector of size D

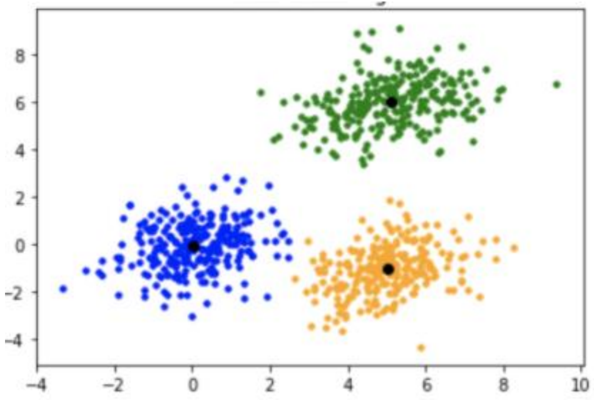
and choose K which gives the smallest AIC (small loss + large K values penalized)

- More advanced approaches, such as nonparametric Bayesian methods (Dirichlet Process mixture models also used, not within K-means but with other clustering algos)

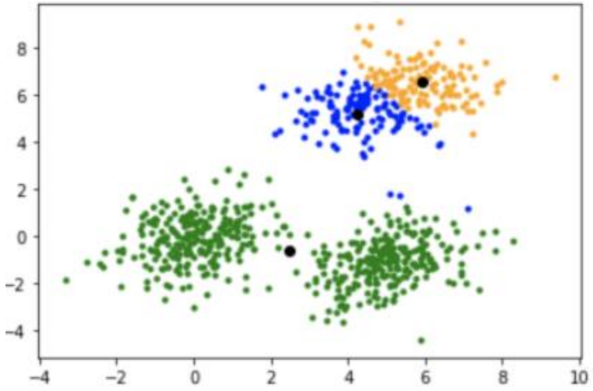
K-means++

- K-means results can be sensitive to initialization

Desired clustering



Poor initialization: bad clustering



- K-means++ (Arthur and Vassilvitskii, 2007) an improvement over K-means
  - Only difference is the way we initialize the cluster centers (rest of it is just K-means)
  - Basic idea: Initialize cluster centers such that they are reasonably far from each other
  - Note: In K-means++, the cluster centers are chosen to be K of the data points themselves

K-means++

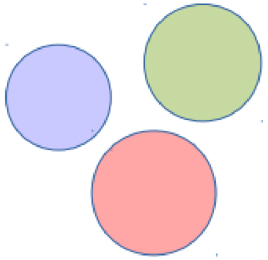
- K-means++ works as follows
  - Choose the first cluster mean uniformly randomly to be one of the data points
  - The subsequent $K-1$ cluster means are chosen as follows
    1. For each unselected point $\boldsymbol{x}$, compute its smallest distance $D(\boldsymbol{x})$ from already initialized means
    2. Select the next cluster mean unif. rand. to be one of the unselected points based on probability prop. to $D(\boldsymbol{x})^2$
    3. Repeat 1 and 2 until the $K-1$ cluster means are initialized
  - Now run standard K-means with these initial cluster means

Thus farthest points are most likely to be selected as cluster means

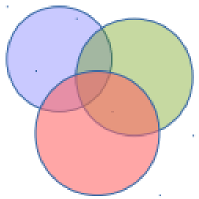
- K-means++ initialization scheme sort of ensures that the initial cluster means are located in different clusters

K-means: Hard vs Soft Clustering

- *K*-means makes hard assignments of points to clusters
    - Hard assignment: A point either completely belongs to a cluster or doesn't belong at all



Hard-assignment okay



Hard-assignment tricky

A more principled extension of K-means for doing soft-clustering is via probabilistic mixture models such as the Gaussian Mixture Model

    - When clusters overlap, soft assignment is preferable (i.e., probability of being assigned to each cluster: say $K = 3$ and for some point $\boldsymbol{x}_n$, $p_1 = 0.7, p_2 = 0.2, p_3 = 0.1$)
- A heuristic to get soft assignments: Transform distances from clusters into prob.

$$\sum_{k=1}^{K} \gamma_{nk} = 1$$

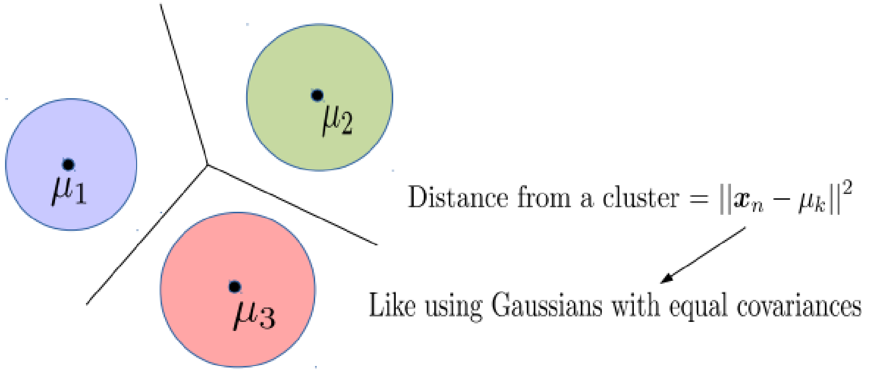
$$\gamma_{nk} = \frac{\exp(-||\boldsymbol{x}_n - \mu_k||^2)}{\sum_{\ell=1}^{K} \exp(-||\boldsymbol{x}_n - \mu_\ell||^2)} \quad \text{(prob. that } \boldsymbol{x}_n \text{ belongs to cluster } k\text{)}$$

- Each cluster mean updates changes: $\mu_k = \frac{\sum_{n=1}^{N} \gamma_{nk} \boldsymbol{x}_n}{\sum_{n=1}^{N} \gamma_{nk}}$ (all points contribute fractionally)

K-means: Decision Boundaries and Cluster Sizes/Shapes

- K-mean assumes that the decision boundary between any two clusters is linear
- Reason: The K-means loss function implies assumes equal-sized, spherical clusters

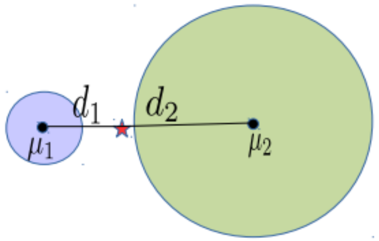
Reason: Use of Euclidean distances

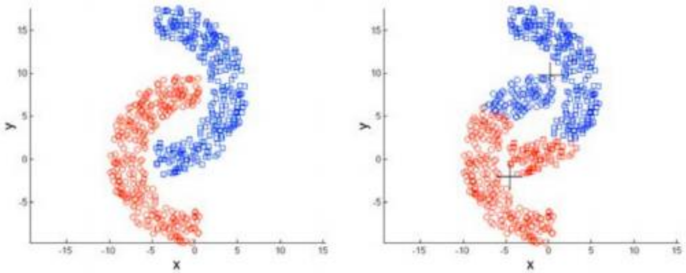


Distance from a cluster $= ||\boldsymbol{x}_n - \mu_k||^2$

Like using Gaussians with equal covariances

- May do badly if clusters are not roughly equi-sized and convex-shaped





Kernel K-means

Helps learn non-spherical clusters and nonlinear cluster boundaries

- Basic idea: Replace the Eucl. distances in K-means by the kernelized versions

Kernelized distance between input $\boldsymbol{x}_n$ and mean of cluster k

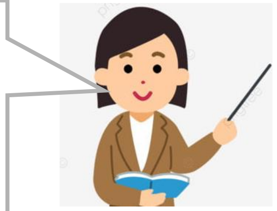
$$||\phi(\boldsymbol{x}_n) - \phi(\boldsymbol{\mu}_k)||^2 = ||\phi(\boldsymbol{x}_n)||^2 + ||\phi(\boldsymbol{\mu}_k)||^2 - 2\phi(\boldsymbol{x}_n)^\top\phi(\boldsymbol{\mu}_k)$$

- Here $\boldsymbol{k}(.,.)$ denotes the kernel function and $\boldsymbol{\phi}$ is its (implicit) feature map
- Note: $\boldsymbol{\phi}(\mu_k)$ is the mean of $\boldsymbol{\phi}$ mappings of the data points assigned to cluster $\boldsymbol{k}$

Not the same as the $\boldsymbol{\phi}$ mapping of the mean of the data points assigned to cluster $\boldsymbol{k}$

$$\phi(\boldsymbol{\mu}_k) = \frac{1}{|\mathcal{C}_k|}\sum_{n:z_n=k}\phi(\boldsymbol{x}_n)$$

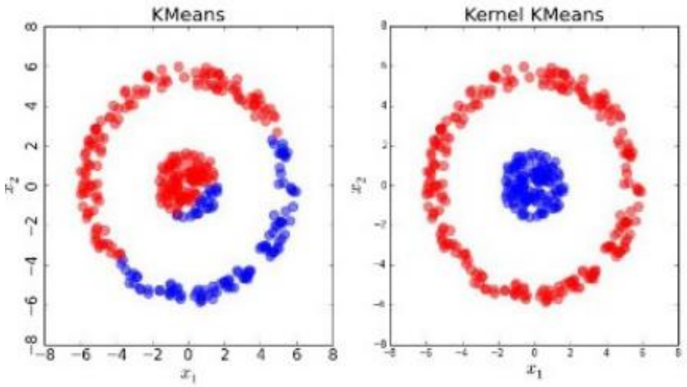
Can also used landmarks or kernel random features idea to get new features and run standard k-means on those



$$||\phi(\boldsymbol{\mu}_k)||^2 = \phi(\boldsymbol{\mu}_k)^\top\phi(\boldsymbol{\mu}_k)$$

$$= \frac{1}{|\mathcal{C}_k|^2}\sum_{n:z_n=k}\sum_{n:z_m=k}k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

$$\phi(\boldsymbol{x}_n)^\top\phi(\boldsymbol{\mu}_k) = \frac{1}{|\mathcal{C}_k|}\sum_{m:z_m=k}k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$



Note: Apart from kernels, it is also possible to use other distance functions in K-means. Bregman Divergence* is such a family of distances (Euclidean and Mahalanobis are special cases)

*Clustering with Bregman Divergences (Banerjee et al, 2005)

Hierarchical Clustering

- Can be done in two ways: Agglomerative or Divisive

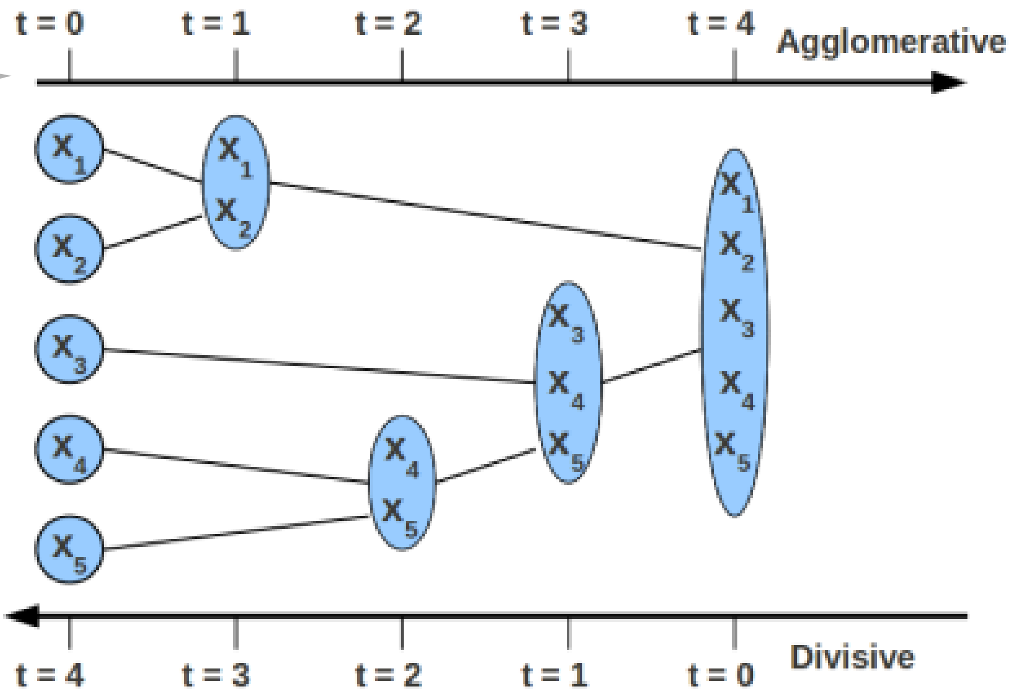
Similarity between two clusters (or two set of points) is needed in HC algos (e.g., this can be average pairwise similarity between the inputs in the two clusters)

Agglomerative: Start with each point being in a singleton cluster

At each step, greedily merge two most "similar" sub-clusters

Stop when there is a single cluster containing all the points

Learns a dendrogram-like structure with inputs at the leaf nodes. Can then choose how many clusters we want



Keep recursing until the desired number of clusters found

At each step, break a cluster into (at least) two smaller homogeneous sub-clusters

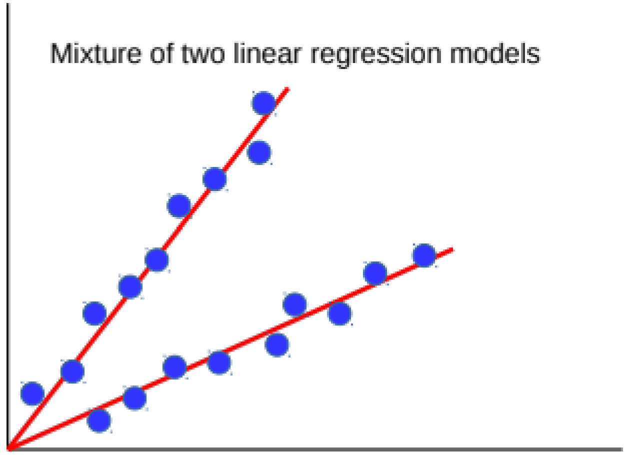
Divisive: Start with all points being in a single cluster

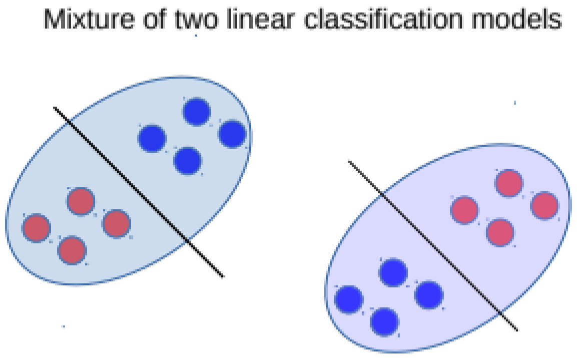
Tricky because no labels (unlike Decision Trees)

- Agglomerative is more popular and simpler than divisive (the latter usually needs complicated heuristics to decide cluster splitting).
- Neither uses any loss function

Clustering can help supervised learning, too

- Often "difficult" sup. learning problems can be seen as mixture of simpler models
- Example: Nonlinear regression or nonlinear classification as mixture of linear models





- Don't know which point should be modeled by which linear model ⇒ Clustering
- Can therefore solve such problems as follows
  - Initialize each linear model somehow (maybe randomly)
  - Cluster the data by assigning each point to its "closest" linear model (one that gives lower error)
  - (Re-)Learn a linear model for each cluster's data. Go to step 2 if not converged.

Such an approach is also an example of divide and conquer and is also known as "mixture of experts" (will see it more formally when we discuss latent variable models)

Evaluating Clustering Algorithms

- Clustering algos are in general harder to evaluate since we rarely know the ground truth clustering (since clustering is unsupervised)

- If ground truth labels not available, use output of clustering for some other task
  - For example, use cluster assignment $\boldsymbol{z}_n$ (hard or soft) as a new feature representation
  - Performance on some task using this new rep. is a measure of goodness of clustering

- If ground truth labels are available, can compare them with clustering based labels
  - Not straightforward to compute accuracy since the label identities may not be the same, e.g.,

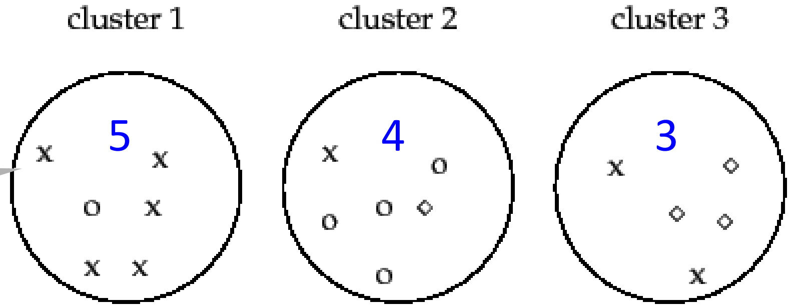
    Ground truth = [1 1 1 0 0 0] Clustering = [0 0 0 1 1 1]

    (Perfect clustering but zero "accuracy" if we just do a direct match)
  - There are various metrics that take into account the above fact
    - Purity, Rand Index, F-score, Normalized Mutual Information, distortion or loss on test data etc

Evaluating Clustering Algorithms

- Purity: Looks at how many points in each cluster belong to the majority class in that cluster

3 classes (x, o ,△, assuming known ground truth labels)



Sum and divide by total number of points

Purity = (5+4+3)/17 ≈ 0.71

Close to 0 for bad clustering, 1 for perfect clustering

Also a bad metric if number of clusters is very large – each cluster will be kind of pure anyway

- Rand Index (RI): Can also look at what fractions of pairs of points with same (resp. different) label are assigned to same (resp. different) cluster

F_β score is also popular

$$P = \frac{\text{TP}}{\text{TP}+\text{FP}} \qquad R = \frac{\text{TP}}{\text{TP}+\text{FN}} \qquad F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

Precision

Recall

True Positive: No. of pairs with same true label and same cluster

True Negative: No. of pairs with diff true label and diff clusters

$$\text{RI} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{FN}+\text{TN}}$$

False Positive: No. of pairs with diff true label and same cluster

False Negative: No. of pairs with same true label and diff cluster

Overlapping Clustering

Kind of unsupervised version of multi-label classification (just like standard clustering is like unsupervised multi-class classification)

Example: Clustering people based on the interests they may have (a person may have multiple interests; thus may belong to more than one cluster simultaneously)

- Have seen hard clustering and soft clustering
- In hard clustering, z_n is a one-hot vector
- In soft clustering, z_n is a vector of probabilities
- Overlapping Clustering: A point can simultaneously belong to multiple clusters
  - This is different from soft-clustering
  - z_n would be a binary vector, rather than a one hot or probability vector, e.g.,

$$z_n = [1\ 0\ 0\ 1\ 0]$$

K=5 clusters with point x_n belonging (in whole, not in terms of probabilities) to clusters 1 and 4

- In general, more difficult than hard/soft clustering (for N data points and K clusters, the size of the space of possible solutions is not K^N but 2^{NK} - exp in both N and K)
- K-means has extensions* for doing overlapping clustering. There also exist latent variable models for doing overlapping clustering

*An extended version of the k-means method for overlapping clustering (Cleuziou, 2008); Non-exhaustive, Overlapping k-means (Whang et al, 2015)