Clustering & Unsupervised Learning

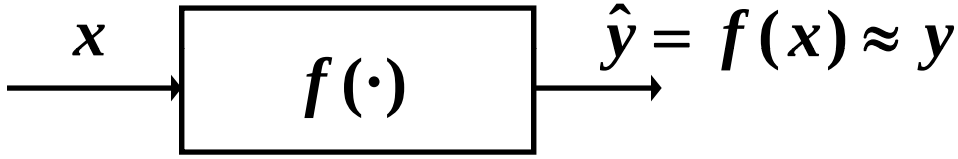
Nuno Vasconcelos

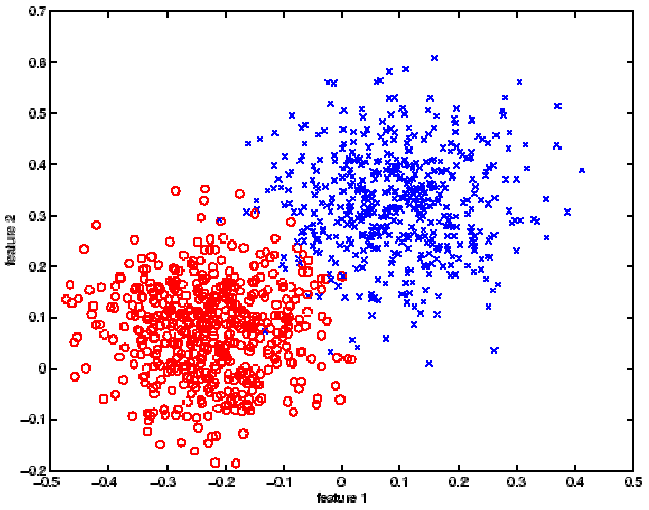
(Ken Kreutz-Delgado)

UCSD

Statistical Learning

- Goal: Given a relationship between a feature vector x and a vector y, and iid data samples (x_i, y_i), find an approximating function $f(x) \approx y$

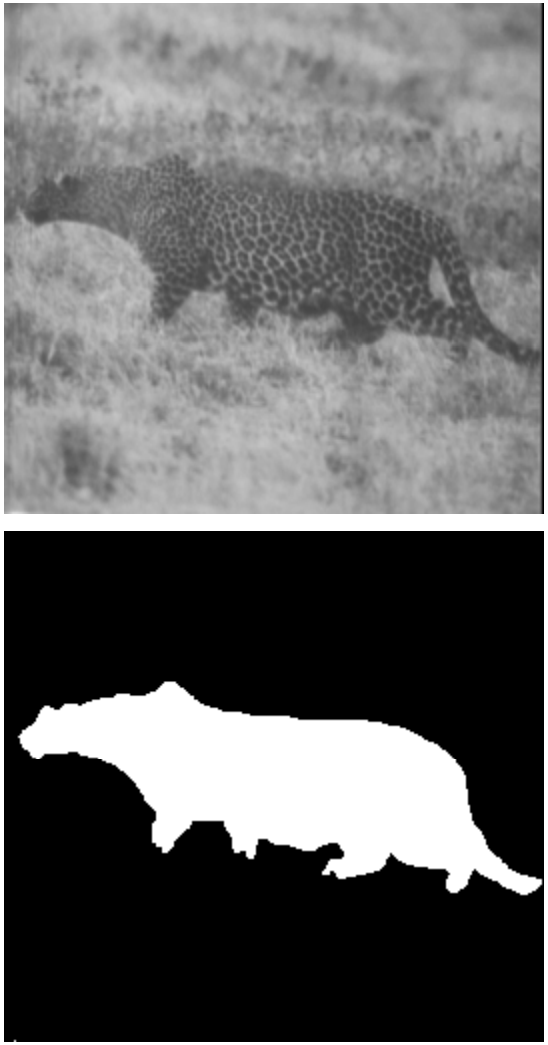




- This is called training or learning.
- Two major types of learning:
  - Unsupervised Classification (aka Clustering) : only X is known.
  - Supervised Classification or Regression: both X and target value Y are known during training, only X is known at test time.

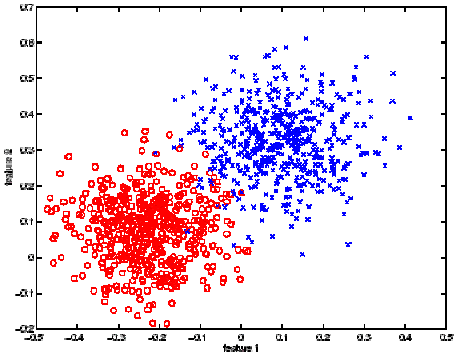
Unsupervised Learning – Clustering

- Why learning without supervision?
  - In many problems labels are not available or are impossible or expensive to get.
  - E.g. in the hand-written digits example, a human sat in front of the computer for hours to label all those examples.
  - For other problems the classes to be labeled depend on the application.
  - A good example is image segmentation:
    - if you want to know if this is an image of the wild or of a big city, there is probably no need to segment.
    - If you want to know if there is an animal in the image, then you would segment.
      - Unfortunately, the segmentation mask is usually not available



Review of Supervised Classification

Although our focus on clustering, let us start by reviewing supervised classification:



- To implement the optimal decision rule for a supervised classification problem, we need to
  - Collect a labeled iid training data set
    $\mathcal{D} = \{(x_1,y_1) , \ldots , (x_n,y_n)\}$
    where x_i is a vector of observations
    and y_i is the associated class label,

  and then

  Learn a probability model for each class

  - This involves estimating $P_{X|Y}(x|i)$ and $P_Y(i)$ for each class i

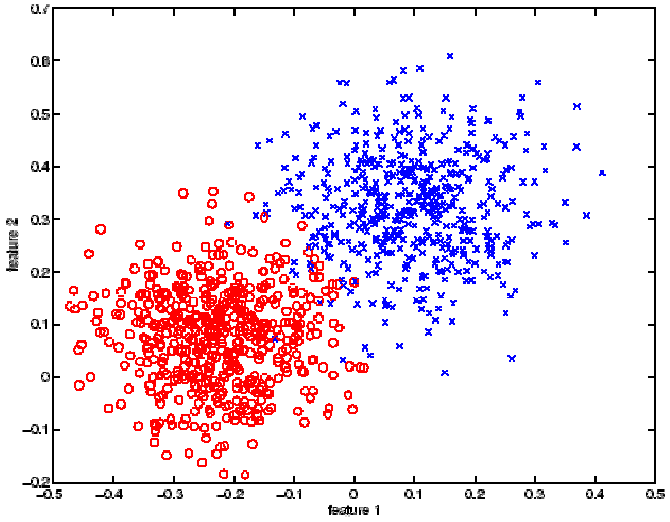
Supervised Classification

- This can be done by Maximum Likelihood Estimation

  MLE has two steps:

  1) Choose a parametric model for each class pdf:

$$P_{X|Y}(x \mid i;\theta_i) \quad \theta_i \in \Theta_i$$



  2) Select the parameters of class i to be the ones that maximize the probability of the iid data from that class:

$$\hat{\theta}_i = \arg\max_{\theta_i \in \Theta_i} P_{X|Y}\left(\mathrm{D}^{(i)} \mid i;\theta_i\right)$$

$$= \arg\max_{\theta_i \in \Theta_i} \log P_{X|Y}\left(\mathrm{D}^{(i)} \mid i;\theta_i\right)$$

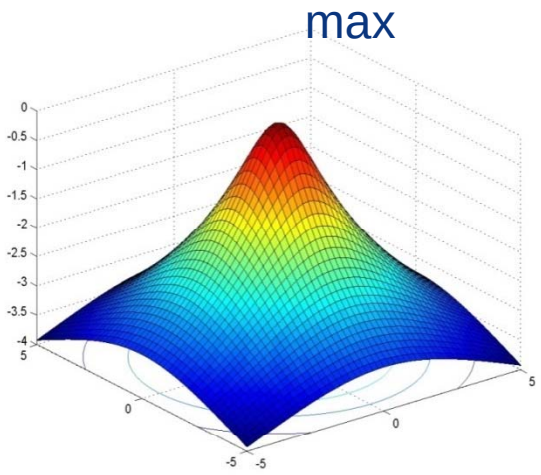
Maximum Likelihood Estimation

- We have seen that MLE can be a straightforward procedure. In particular, *if* the pdf is twice differentiable then:

- Solutions are parameters values such that

$$\frac{\partial}{\partial \theta_i} \boldsymbol{P}_{X|Y}(\mathrm{D}^{(i)} \mid i; \hat{\theta}_i) = \mathbf{0}$$

$$\theta_i^T \frac{\partial^2}{\partial \theta_i^2} \boldsymbol{P}_{X|Y}(\mathrm{D}^{(i)} \mid i; \hat{\theta}_i)\, \theta_i \leq \mathbf{0}, \quad \forall \theta_i \in \Theta_i \subset \mathbb{R}^{p_i}$$



- You always have to check the second-order condition

- We must also find an MLE for the class probabilities $P_Y(i)$
  - But here there is not much choice of probability model
    - E.g. Bernoulli: ML estimate is the percent of training points in the class

Maximum Likelihood Estimation

- We have worked out the Gaussian case in detail:
  - $\mathcal{D}^{(i)} = \{x_1^{(i)}, \ldots, x_{n_i}^{(i)}\}$ = set of examples from class *i*
  - The ML estimates for class *i* are

$$\hat{\mu}_i = \frac{1}{n_i} \sum_j x_j^{(i)} \qquad \hat{P}_Y(i) = \frac{n_i}{n}$$

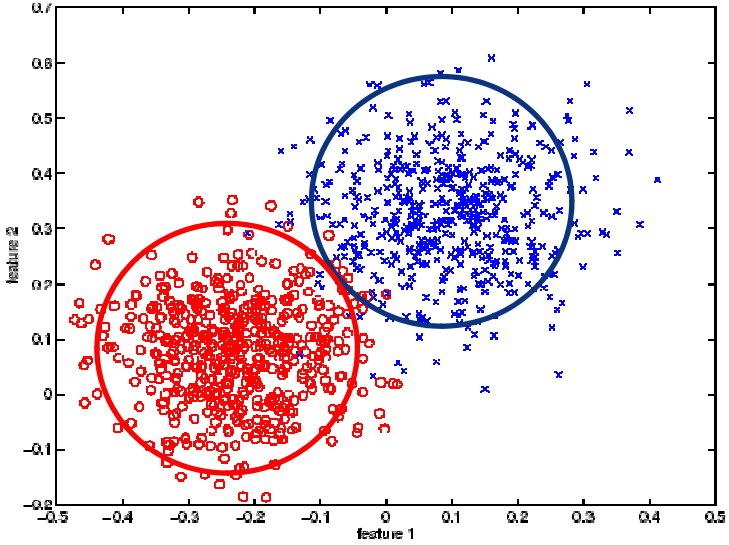
$$\hat{\Sigma}_i = \frac{1}{n_i} \sum_j (x_j^{(i)} - \hat{\mu}_i)(x_j^{(i)} - \hat{\mu}_i)^T$$

- There are many other distributions for which we can derive a similar set of equations
- But the Gaussian case is particularly relevant for clustering (more on this later)

Supervised Learning via MLE

- This gives probability models for each of the classes
- Now we utilize the fact that:
  - assuming the zero/one loss, the optimal decision rule (BDR) is the MAP rule:

$$i^*(x) = \arg\max_i P_{Y|X}(i \mid x)$$



Which can also be written as

$$i^*(x) = \arg\max_i \left[\log P_{X|Y}(x \mid i) + \log P_Y(i) \right]$$

- This completes the process of supervised learning of a BDR. We now have a rule for classifying any (unlabeled) future measurement *x*.

Gaussian Classifier

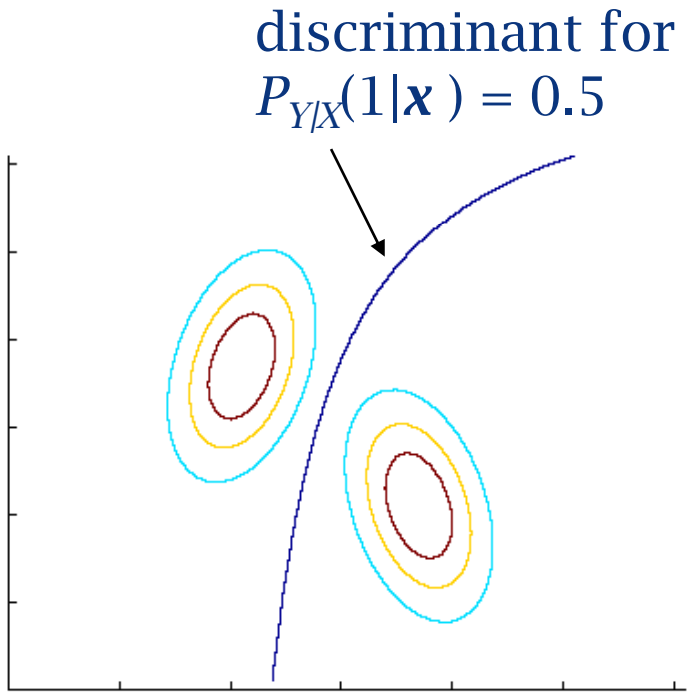
- In the Gaussian case the BDR is

$$i^*(x) = \arg\min_i \left[d_i^2(x, \mu_i) + \alpha_i \right]$$

with

$$d_i^2(x, y) = (x - y)^T \Sigma_i^{-1} (x - y)$$

$$\alpha_i = \log(2\pi)^d \left| \Sigma_i \right| - 2 \log P_Y(i)$$



- This can be seen as finding the nearest class neighbor, using a funny metric
  - Each class has its own squared-distance which is the sum of Mahalanobis-squared for that class plus the α constant
    - We effectively have different metrics in different regions of the space

Gaussian Classifier

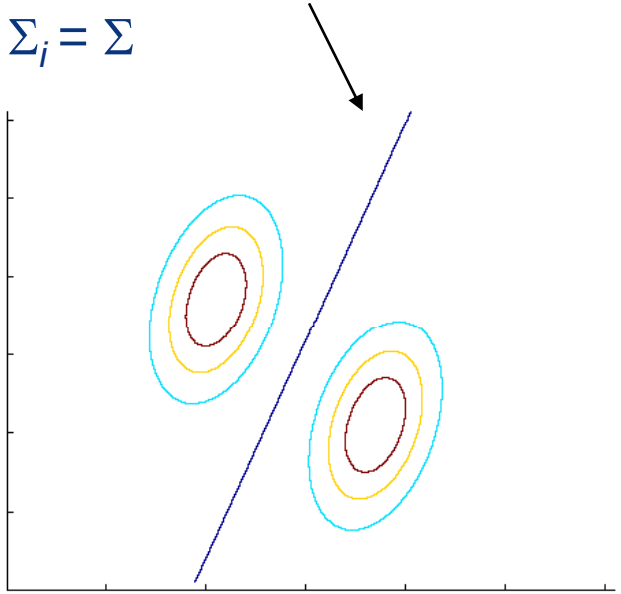
- A special case of interest is when
  - all classes have the same covariance $\Sigma_i = \Sigma$

$$i^*(x) = \arg\min_i \left[d^2(x, \mu_i) + \alpha_i \right]$$

with

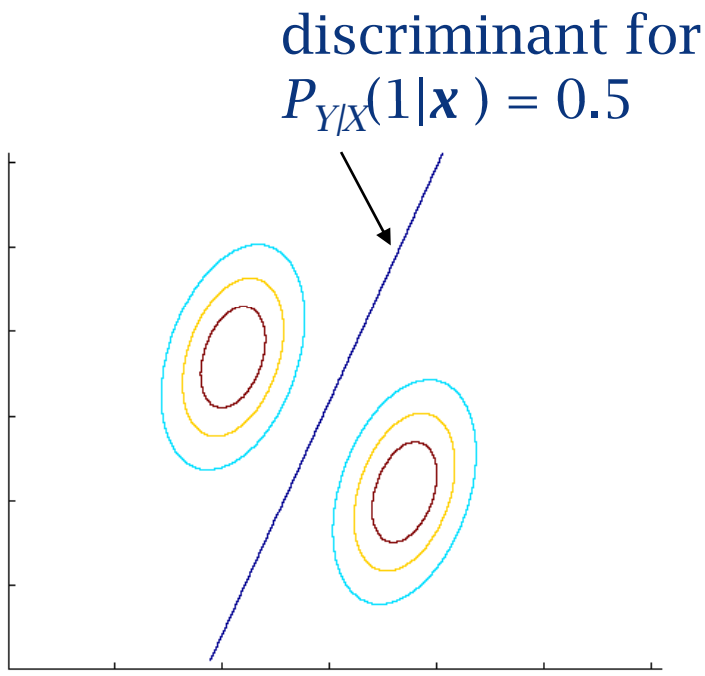
$$d^2(x, y) = (x - y)^T \Sigma^{-1} (x - y)$$

$$\alpha_i = -2 \log P_Y(i)$$



  - Note: α_i can be dropped when all classes have equal probability
    - Then this is close to the NN classifier with Mahalanobis distance
    - However, instead of finding the nearest neighbor, it looks for the nearest class "prototype" or "template" μ_i

Gaussian Classifier



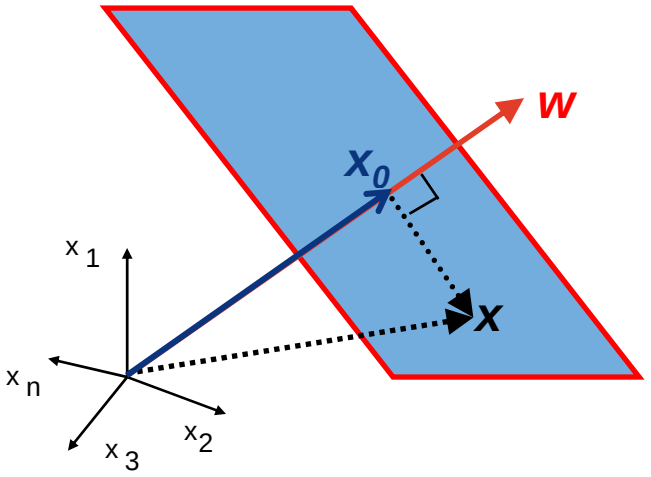
- $\Sigma_i = \Sigma$ for two classes (detection)
  - One important property of this case is that the decision boundary is a hyperplane.
  - This can be shown by computing the set of points x such that

$$d^2(x,\mu_0)+\alpha_0 = d^2(x,\mu_1)+\alpha_1$$

and showing that they satisfy

$$w^T(x-x_0)=0$$

  - This is the equation of a *hyperplane* with normal w. x_0 can be *any* fixed point on the hyperplane, but it is *standard* to choose it to have minimum norm, in which case w and x_0 are then parallel



Gaussian Classifier

- if all the covariances are the identity $\Sigma_i = I$

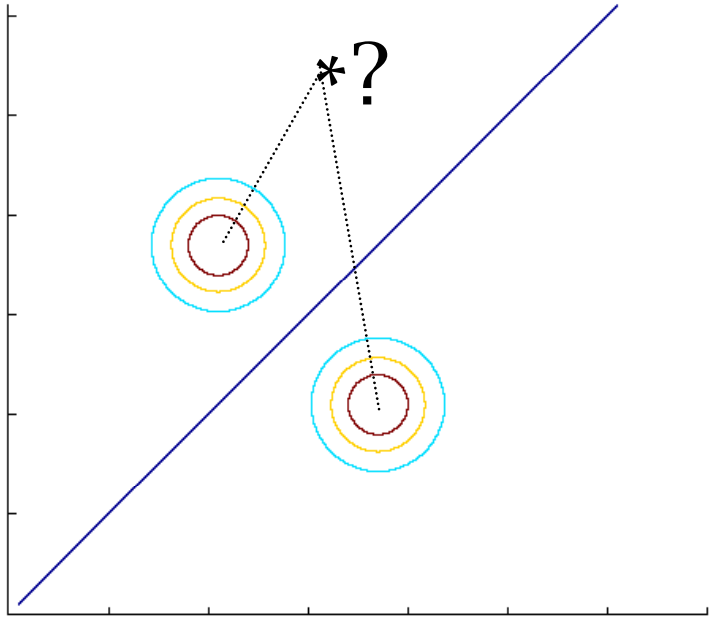
$$i^*(x) = \arg\min_i \left[d^2(x, \mu_i) + \alpha_i \right]$$

with

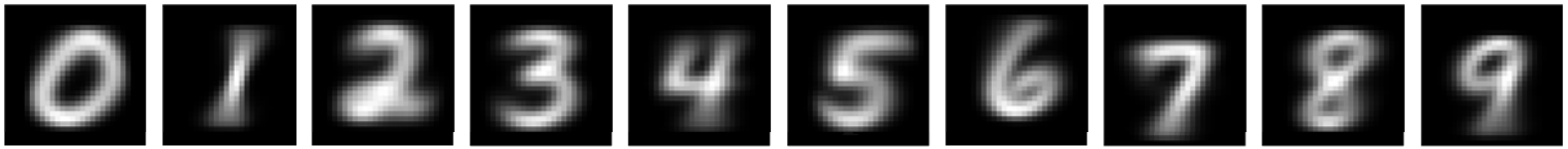
$$d^2(x, y) = \| x - y \|^2$$

$$\alpha_i = -2 \log P_Y(i)$$

This is just (Euclidean distance) template matching with class means as templates



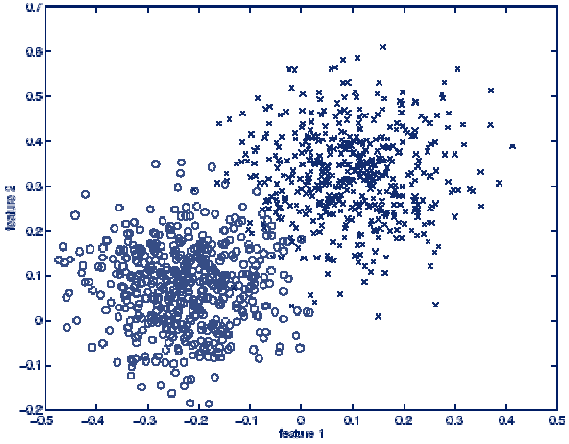
- e.g. for digit classification, the class means (templates) are:



- Compare complexity of template matching to nearest neighbors!

Unsupervised Classification - Clustering

- In a clustering problem we do not have labels in the training set
- We can try to estimate both the class labels and the class pdf parameters
- Here is a strategy:
  - Assume *k* classes with pdf's initialized to randomly chosen parameter values
  - Then iterate between two steps:
    1) Apply the optimal decision rule for the (estimated) class pdf's
       - this assigns each point to one of the clusters, creating pseudo-labeled data
    2) Update the pdf estimates by doing parameter estimation within each estimated (pseudo-labeled) class cluster found in step 1



Unsupervised Classification - Clustering

- Natural question: what probability model do we assume?
  - Let's start as simple as possible
  - Assume: *k* Gaussian classes with identity covariances & equal $P_Y(i)$
  - Each class has an *unknown mean* (prototype) μ_i which must be learned
- Resulting clustering algorithm is the *k*-means algorithm:
  - Start with some initial estimate of the μ_i (e.g. random, but distinct)
  - Then, iterate between
    - 1) BDR Classification using the current estimates of the *k* class means:

$$i^*(x) = \arg\min_{1 \le i \le k} \left\| x - \mu_i \right\|^2$$

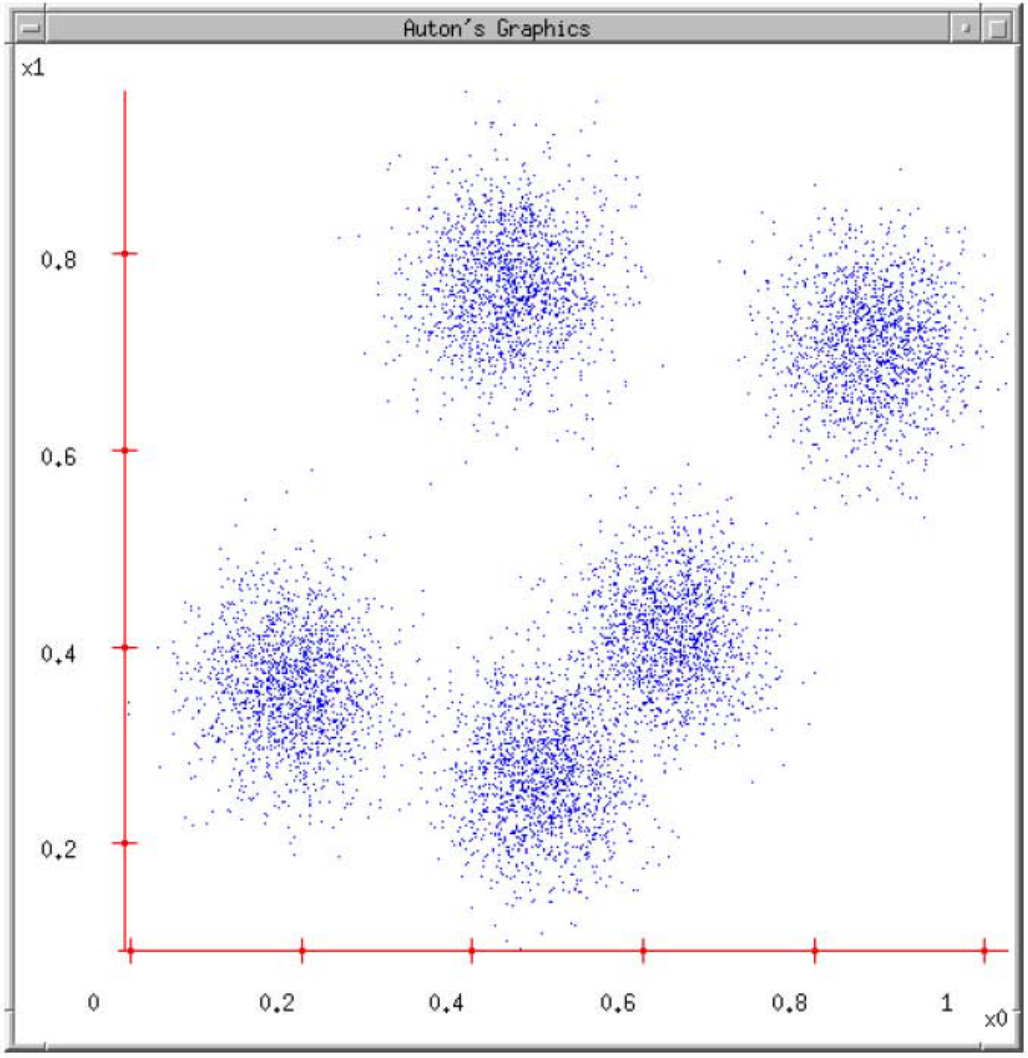
    - 2) Re-estimation of the *k* class means:

$$\mu_i \leftarrow \mu_i^{new} = \frac{1}{n_i} \sum_{j=1}^{n_i} x_j^{(i)} \quad \text{for} \quad i = 1, \cdots, k$$

K-means (thanks to Andrew Moore, CMU)

K-means

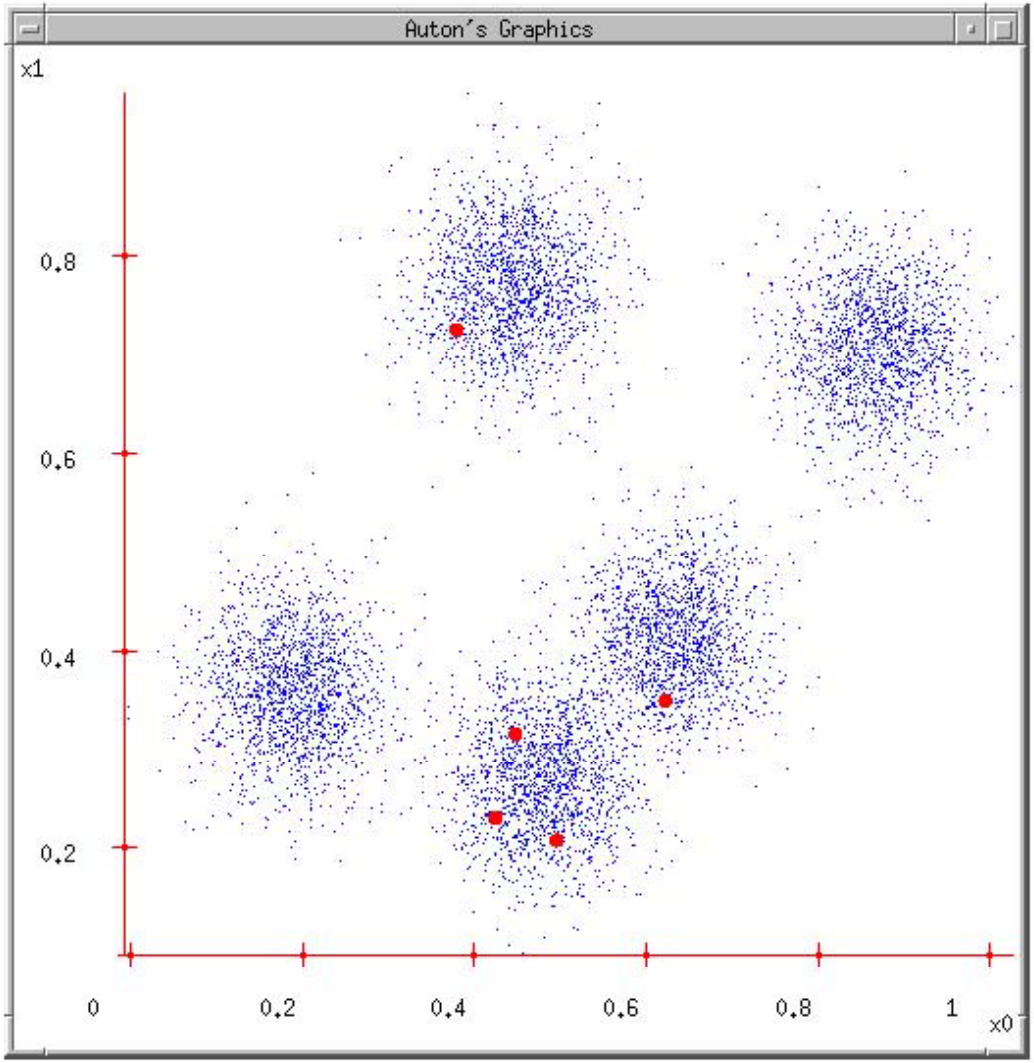
1. Ask user how many clusters they'd like. *(e.g. k=5)*



K-means (thanks to Andrew Moore, CMU)

K-means

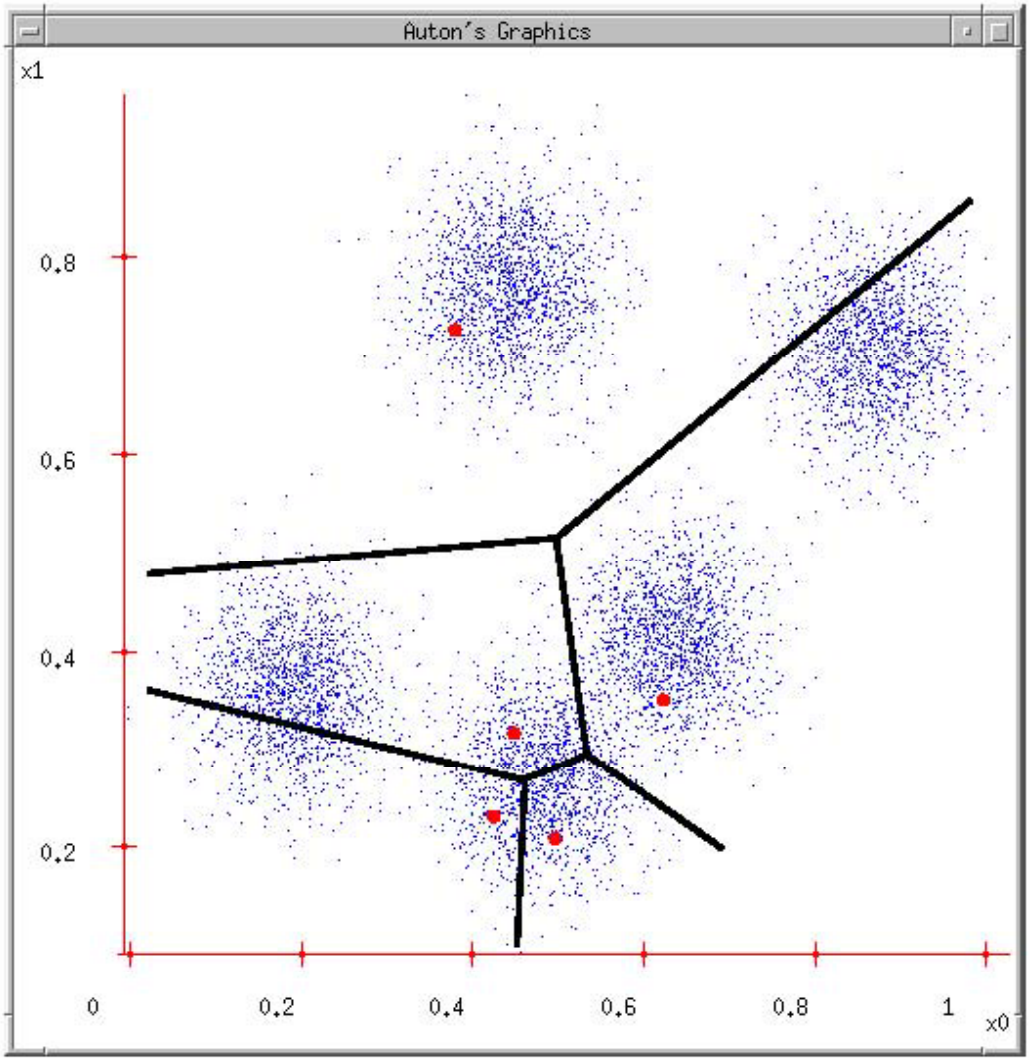
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations



K-means (thanks to Andrew Moore, CMU)

K-means

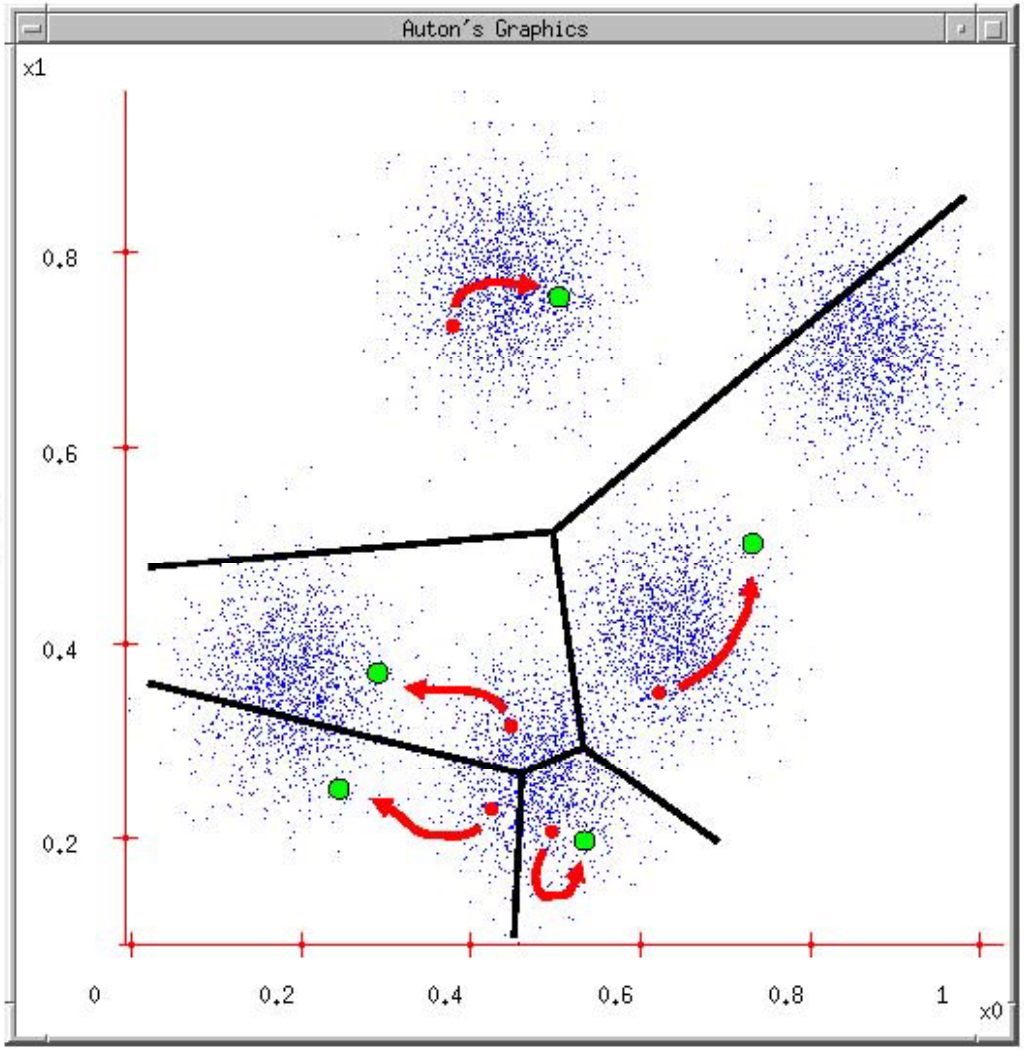
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



K-means (thanks to Andrew Moore, CMU)

K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns

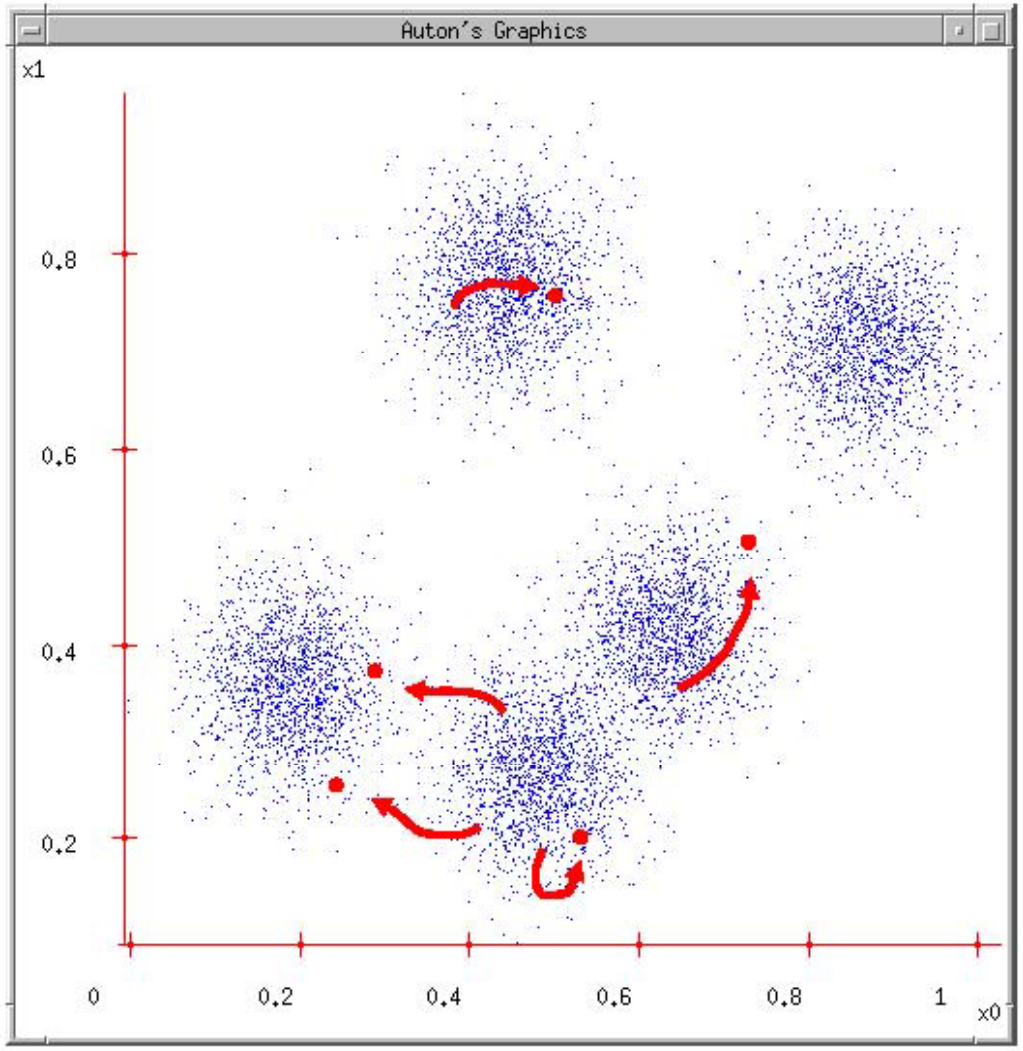


K-means and Hierarchical Clustering: Slide 9

K-means (thanks to Andrew Moore, CMU)

K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!



K-means and Hierarchical Clustering: Slide 10

K-means Clustering

- The name comes from the fact that we are trying to learn the "*k*" means (mean values) of "*k*" assumed clusters
- It is optimal if you want to minimize the expected value of the squared error between vector x and template to which x is assigned. K-means results in a *Voronoi tessellation* of the feature space.
- Problems:
  - How many clusters? (i.e., what is *k?*)
    - Various methods available, Bayesian information criterion, Akaike information criterion, minimum description length
    - Guessing can work pretty well
  - Algorithm converges to a local minimum solution only
  - How does one initialize?
    - Random can be pretty bad
    - Mean Splitting can be significantly better

Growing *k* via Mean Splitting

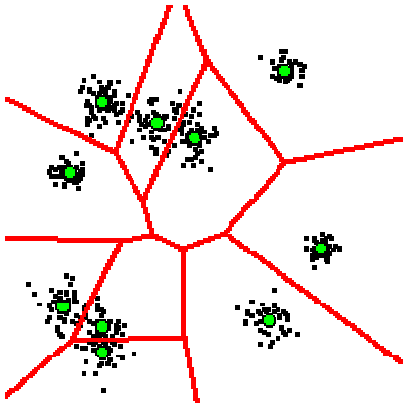
- Let $k = 1$. Compute the sample mean of all points, $\mu^{(1)}$. (The superscript denotes the current value of k)
- To initialize means for $k = 2$ perturb the mean $\mu^{(1)}$ randomly
  - $\mu_1^{(2)} = \mu^{(1)}$
  - $\mu_2^{(2)} = (1+\varepsilon)\ \mu^{(1)}$ $\quad\quad$ $\varepsilon << 1$
- Then run k-means until convergence for $k = 2$
- Initialize means for $k = 4$
  - $\mu_1^{(4)} = \mu_1^{(2)}$
  - $\mu_2^{(4)} = (1+\varepsilon)\ \mu_1^{(2)}$
  - $\mu_3^{(4)} = \mu_2^{(2)}$
  - $\mu_4^{(4)} = (1+\varepsilon)\ \mu_2^{(2)}$
- Then run k-means until convergence for $k = 4$
- Etc ….

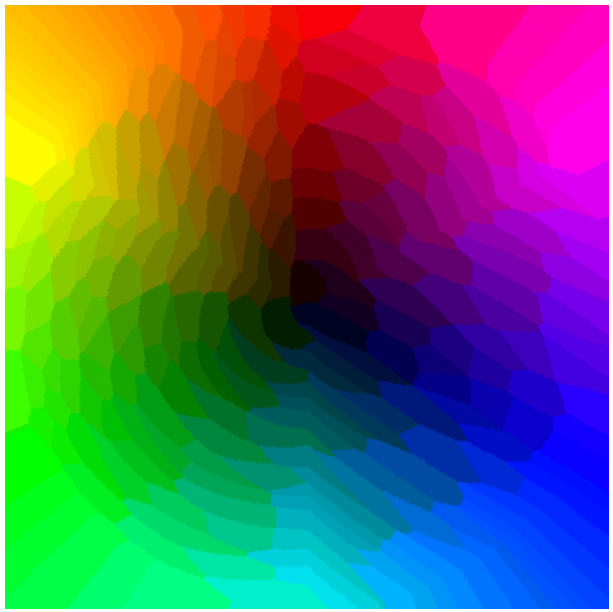
Deleting “Empty” Clusters

- “Empty” Clusters can be a source of algorithmic difficulties
- Therefore, at the end of each iteration of *k*-means
  - Check the number of elements in each cluster
  - If too low, throw the cluster away
  - Reinitialize the mean of the most populated cluster with a perturbed version of that mean
- Note that there are alternative names:
  - In the compression literature this is known as the Generalized Loyd Algorithm
    - This is actually the right name, since Loyd was the first to invent it
  - It is also known as (data) Vector Quantization and is used in the design of vector quantizers

Vector Quantization

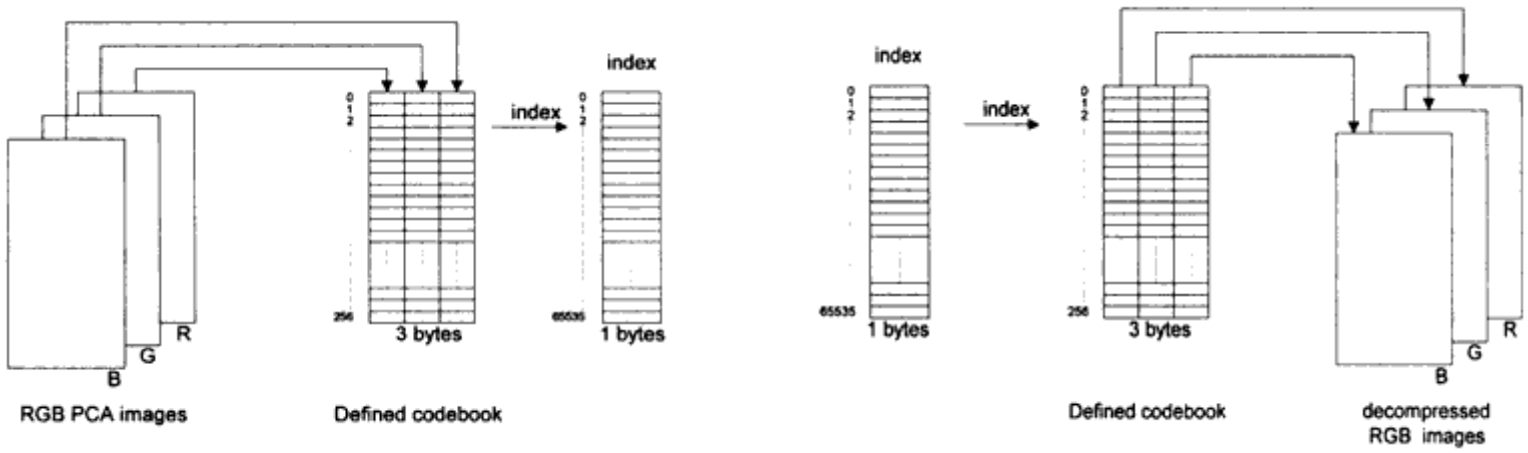
- Is a popular data compression technique
  - Find a “codebook” of prototypes for the vectors to compress
  - Instead of transmitting each vector, transmit the codebook index
  - Image compression example
    - Each pixel has 3 colors (requiring 3 bytes of information)
    - Instead, find the optimal 256 color prototypes! (256 ~ 1 byte of information)





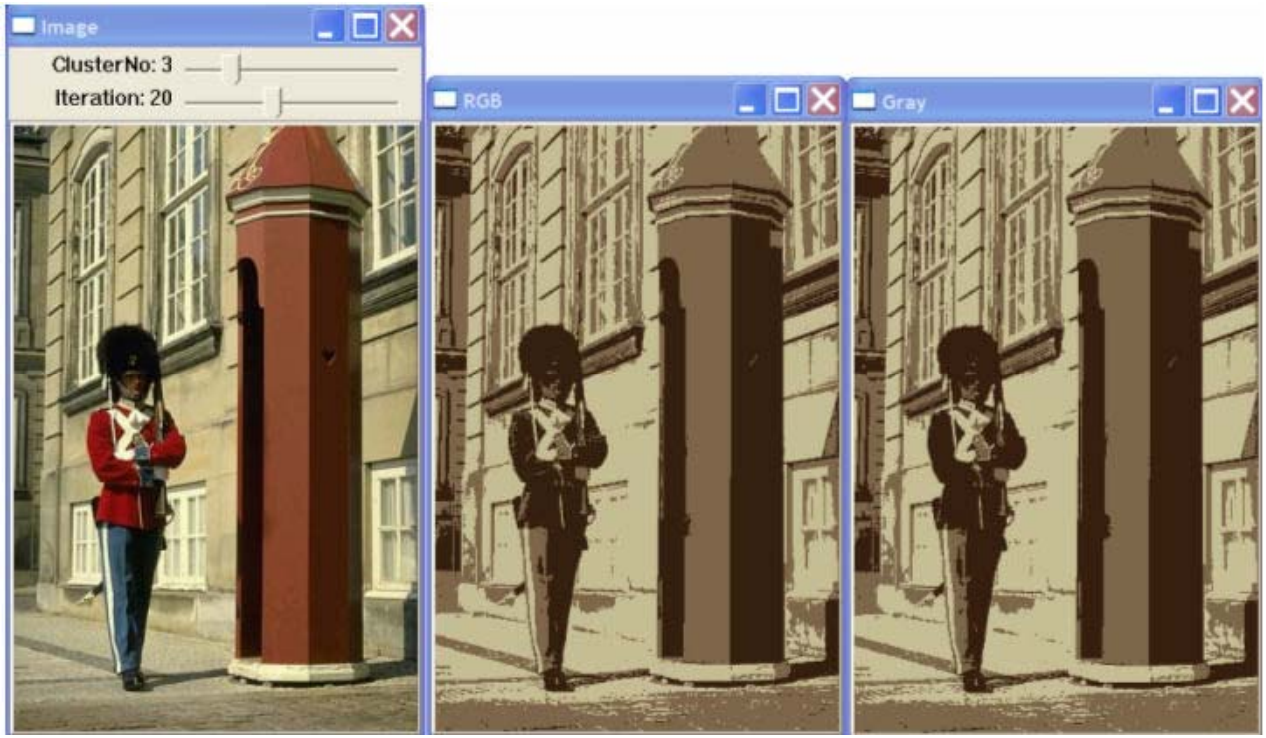
Vector Quantization

- We now have an image compression scheme
  - Each pixel has 3 colors (1 byte per color = 3 bytes total needed))
  - Instead, find the nearest neighbor template for 256 colors
  - We transmit the template index
  - Since there are only 256 templates, only need one byte needed
  - Using the index, the decoder looks up the prototype in its table
  - By sacrificing a little bit of distortion, we saved 2 bytes per pixel!



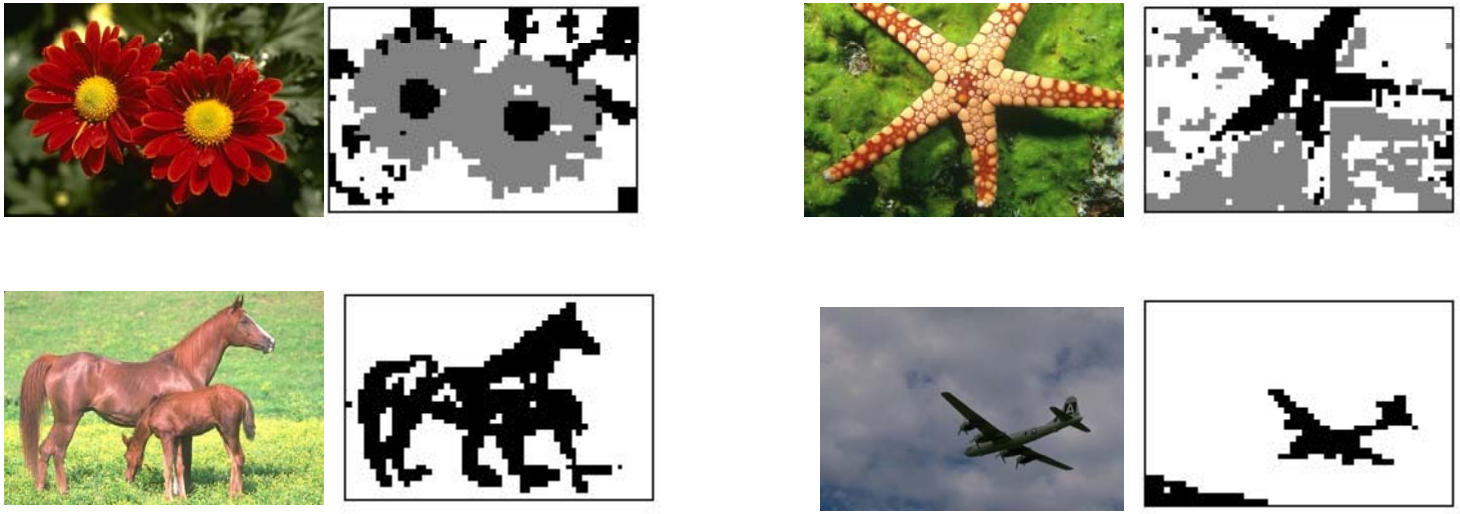
K-means

- There are many other applications of K-means
  - E.g. image segmentation: decompose each image into component objects
    - Then run *k*-means on the colors and look at the assignments
    - E.g., the pixels assigned to the red cluster tend to be from the booth:



K-means

- We can also use texture information in addition to color
  - Many methods for clustering using “texture metrics”
  - Here are some results



  - Note that this is not the state-of-the-art in image segmentation
  - But gives a good idea of what *k*-means can do

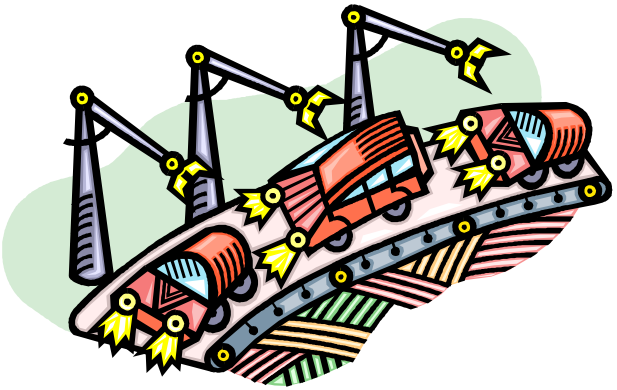
Extensions to basic K-means

- There are many extensions to the basic *k*-means algorithm
  - One of the most important applications is to the problem of learning accurate approximations to general, nontrivial PDF's.
  - Remember that the optimal decision rule

$$i^*(x) = \arg\max_i \left[\log P_{X|Y}(x \mid i) + \log P_Y(i) \right]$$

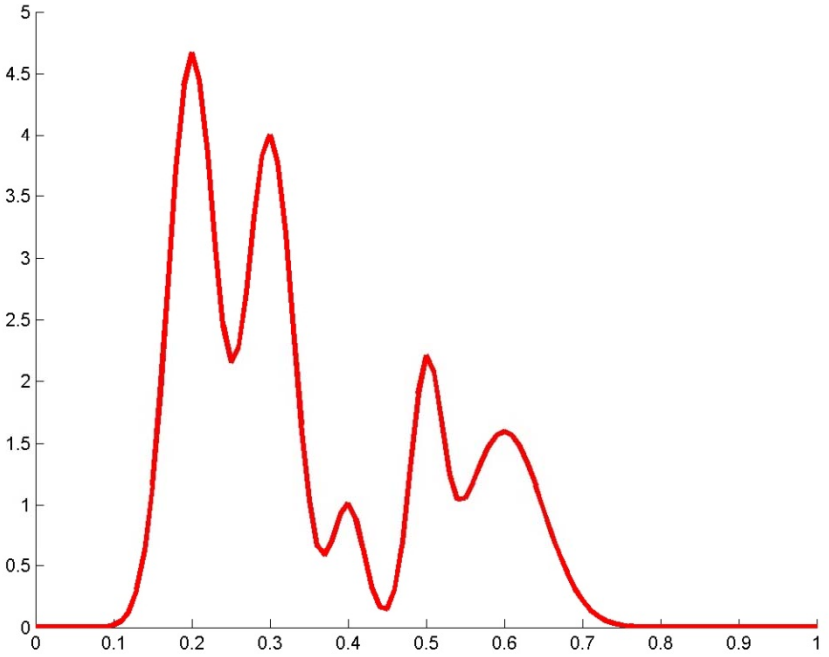
    is optimal iff the true probabilities $P_{X|Y}(x|i)$ are correctly estimated
  - This often turns out to be impossible when we use overly simple parametric models like the Gaussian – Often the true probability is too complicated for any simple model to hold accurately
  - Even if simple models provide good local approximations, there are usually multiple clusters when we take a global view
  - These weakness can be addressed by use of mixture distributions and the use of the Expectation-Maximization (EM) Algorithm

Mixture Distributions



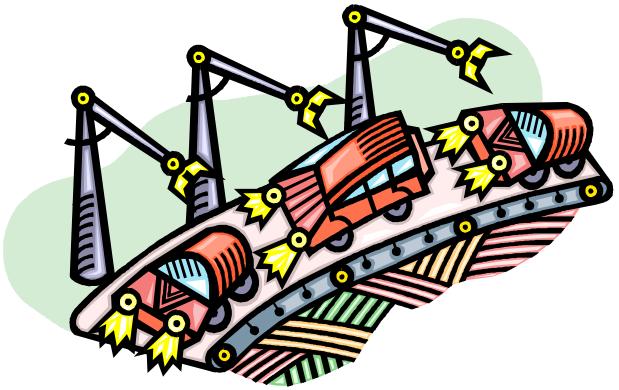
- Consider the following problem
  - Certain types of traffic banned from a bridge
  - We want an automatic detector/classifier to see if the ban is holding
  - A sensor measures vehicle weight
  - Want to classify each car into class = “OK” or class = “Banned”
  - We know that in each class there are multiple sub-classes
  - E.g. OK = {compact, sedan, station wagon, SUV}

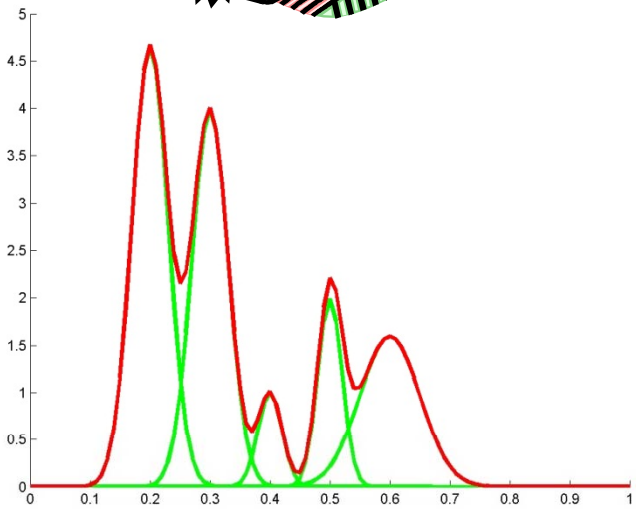
    Banned = {truck, bus, semi}
  - Each of the sub-classes is close to Gaussian, but for the whole class we get this



Mixture distributions

- This distribution is a mixture
  - The overall shape is determined by a number of (sub) class densities
  - We introduce a random variable Z to account for this
  - A value of $Z = c$ points to class c and thus picks out the c^{th} component density from the mixture.
  - E.g. a Gaussian mixture:





$$P_{\mathbf{X}}(\mathbf{x}) = \sum_{c=1}^{C} P_{\mathbf{X}|Z}(\mathbf{x}|c) P_Z(c)$$

$$= \sum_{c=1}^{C} P_{\mathbf{X}|Z}(\mathbf{x}|c) \pi_c$$

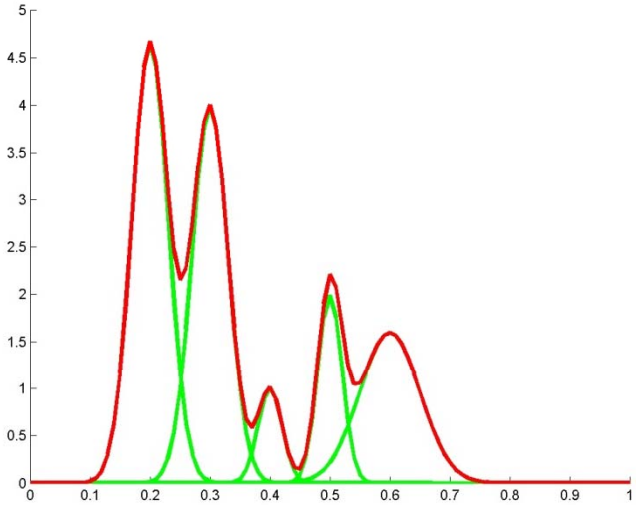
# of mixture components

c^{th} component "weight"

c^{th} "mixture component" = Gaussian pdf

Mixture Distributions

- Learning a mixture density is a type of "soft" clustering problem
  - For each training point x_k we need to figure out from which component class $Z_k = Z(x_k) = j$ it was drawn
  - Once we know how points are assigned to a component j we can estimate the component j pdf parameters



- This *could* be done with *k*-means
- A more general algorithm is Expectation-Maximization (EM)
  - A key difference from *k*-means: we never "hard assign" the points x_k
  - In the expectation step we compute posterior probabilities that a point x_k belongs to class j, for every j, conditioned on all the data $\mathcal{D}$.
    - But we do not make a hard decision! (e.g., we do not assign the point x_k only to a single class via the MAP rule.)
  - Instead, in the maximization step, the point x_k "participates" in *all* classes to a degree weighted by the posterior class probabilities

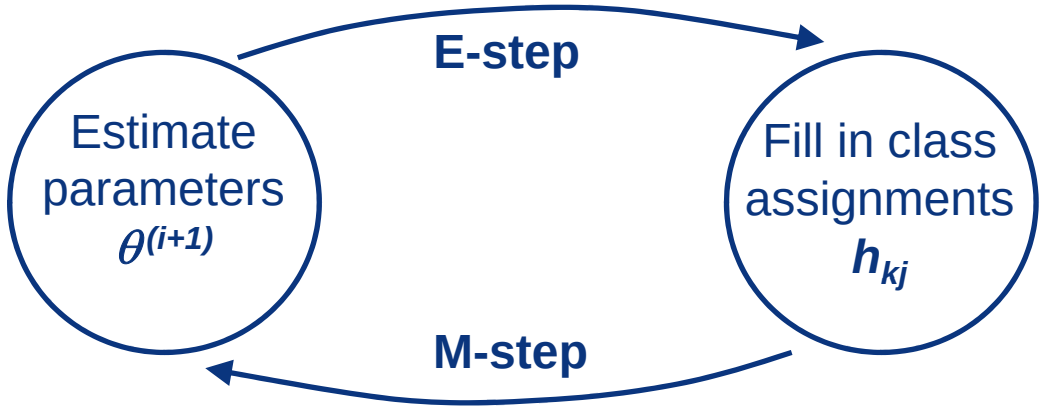
Expectation-Maximization (EM)

- The EM Algorithm:
  1. Start with an initial parameter vector estimate $\theta^{(0)}$
  2. E-step: Given current parameters $\theta^{(i)}$ and observations in $\mathcal{D}$, estimate the indicator functions $\chi(Z_k = j)$ via the conditional Expectation

$$h_{kj} = \mathrm{E}\left\{ \chi(Z_k = j) \middle| \mathcal{D}; \theta^{(i)} \right\} = \mathrm{E}\left\{ \chi(Z_k = j) \middle| x_k; \theta^{(i)} \right\}$$

  1. M-step: Weighting the data x_k by h_{kj}, we have a complete data MLE problem for *each* class j. I.e. Maximize the class j likelihoods for the parameters, i.e. re-compute $\theta^{(i+1)}$
  2. Go to 2.

- In a graphical form:



Expectation Maximization (EM)

- Note that for *any* mixture density we have:

$$h_{kj} = \mathrm{E}\left\{\chi(\mathrm{Z}_k = j) \mid x_k;\theta^{(i)}\right\} = P_{Z|X}\left(\mathrm{Z}_k = j \mid x_k;\theta^{(i)}\right)$$

$$= \frac{P_{X|Z}(x_k \mid Z_k = j;\theta^{(i)})\, P_Z(Z_k = j;\theta^{(i)})}{P_X(x_k;\theta^{(i)})} \quad \textbf{(from Bayes rule)}$$

$$= \frac{P_{X|Z}(x_k \mid Z_k = j;\theta^{(i)})\, \pi_j^{(i)}}{\sum_{c=1}^{C} P_{X|Z}(x_k \mid Z_k = c;\theta^{(i)})\, \pi_c^{(i)}}$$

and

$$n_j = \sum_{k=1}^{n} \chi(\mathrm{Z}_k = j) \quad \Rightarrow \quad \hat{n}_j \triangleq \mathrm{E}\left\{n_j \mid x_k;\theta^{(i)}\right\} = \sum_{k=1}^{n} h_{kj}$$

$$n = \sum_{j=1}^{C} n_j \quad \Rightarrow \quad n = \sum_{j=1}^{C} \hat{n}_j$$

Expectation-Maximization (EM)

- In particular, for a Gaussian mixture we have:
- Expectation Step

$$h_{kj} = P_{Z|X}(Z_k = j \mid x_k; \theta^{(i)}) = \frac{\mathrm{G}(x_k; \mu_j^{(i)}, \sigma_j^{2^{(i)}})\pi_j^{(i)}}{\sum_{c=1}^{C} \mathrm{G}(x_k; \mu_c^{(i)}, \sigma_c^{2^{(i)}})\pi_c^{(i)}}$$

- Maximization Step

$$\hat{n}_j = \sum_{k=1}^{n} h_{kj}, \qquad \pi_j^{(i+1)} = \frac{\hat{n}_j}{n}$$

$$\mu_j^{(i+1)} = \frac{1}{\hat{n}_j} \sum_{k=1}^{n} h_{kj} x_k, \qquad \sigma_j^{2^{(j+1)}} = \frac{1}{\hat{n}_j} \sum_{k=1}^{n} h_{kj} \left(x_k - \mu_j^{(i+1)} \right)^2$$

  - Compare to the single (non-mixture) Gaussian MLE solution shown on slide 7! *They are equivalent solutions* when h_{kj} is the hard indicator function which selects class-labeled data.

Expectation-Maximization (EM)

- Note that the difference between EM and *k*-means is that
  - In the E-step h_{ij} is not hard-limited to 0 or 1
    - Doing so would make the M-step exactly the same as *k*-means
  - Plus we get estimates of the class covariances and class probabilities automatically
- *k*-means can be seen as a "greedy" version of EM
  - At each iteration, for each point we make a hard decision (the optimal MAP BDR for identity covariances & equal class priors)
  - But this does not take into account the information in the points we "throw away". I.e., potentially all points carry information about all (sub) classes
  - Note: If the hard assignment *is* best, EM will learn it
- To get a feeling for EM you can use
  - http://www-cse.ucsd.edu/users/ibayrakt/java/em/

END