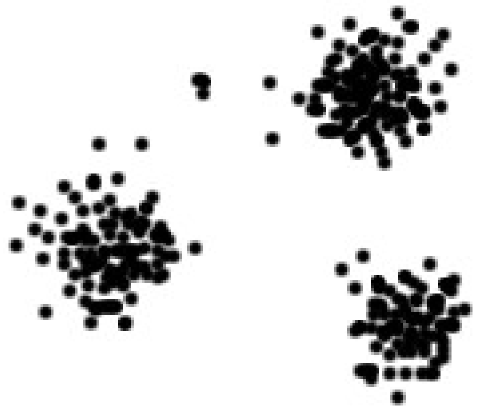
Clustering & EM Algorithm

University of Guelph
ENG*6500

Daniel Jiwoong Im

Clustering

- Suppose that the data was generated from a number of different classes.
- Objective: cluster data that belongs to the same class
- Grouping C objects into K clusters is one of canonical problem in unsupervised learning



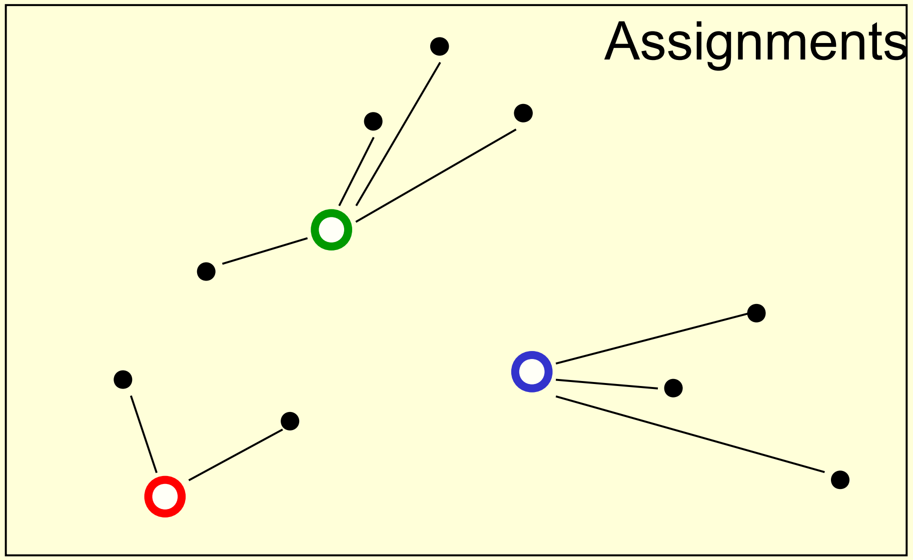
The k-means algorithm

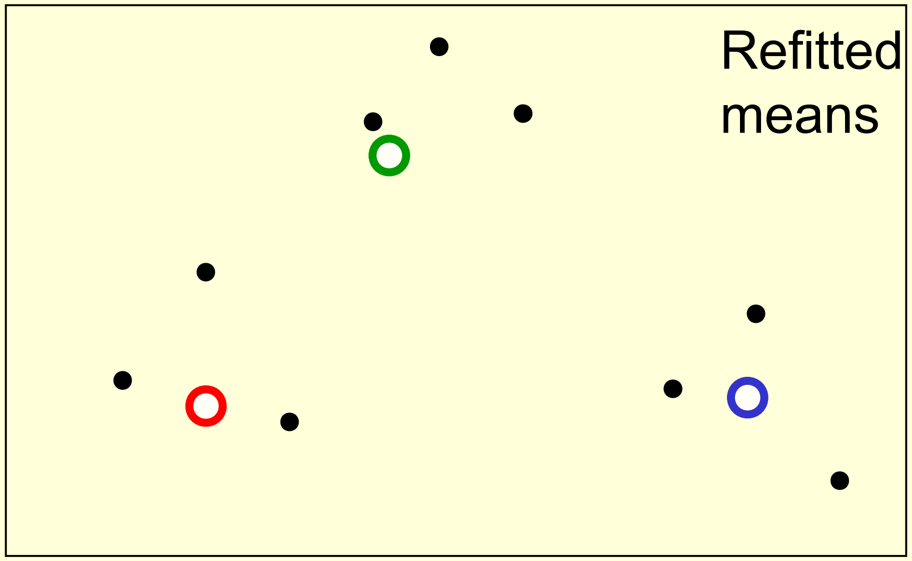
- Assume the data lives in a Euclidean space.
- Assume we want k classes.
- Assume we start with randomly located cluster centers

The algorithm alternates between two steps:

Assignment step: Assign each datapoint to the closest cluster.

Refitting step: Move each cluster center to the center of gravity of the data assigned to it.





Why K-means converges

- Whenever an assignment is changed, the sum of squared distances of data points from their assigned cluster is reduced
- Whenever a cluster centre is moved, the sum of squared distances of the datapoint from their currently assigned cluster centre is reduced
- If the assignment do not change in the assignment step, we have converged (to at least a local minimum)

K-means algorithm

- Initialization : set k means {s_k} to random values
- Assignment : each datapoint c assigned to nearest mean

$$\hat{k}^{(c)} = \mathrm{argmin}_k \{d(\mathbf{m}_k, \mathbf{x}^{(c)})\}$$

- Responsibilities : responsibility r for the cluster k is set to 1 iff that particular data point is assigned is closest to centre of cluster k
- Update: Move the cluster towards the centre of the gravity of its data

$$\mathbf{m}_k = \frac{\sum_c r_k^{(c)} \mathbf{x}^{(c)}}{\sum_c r_k^{(c)}}$$

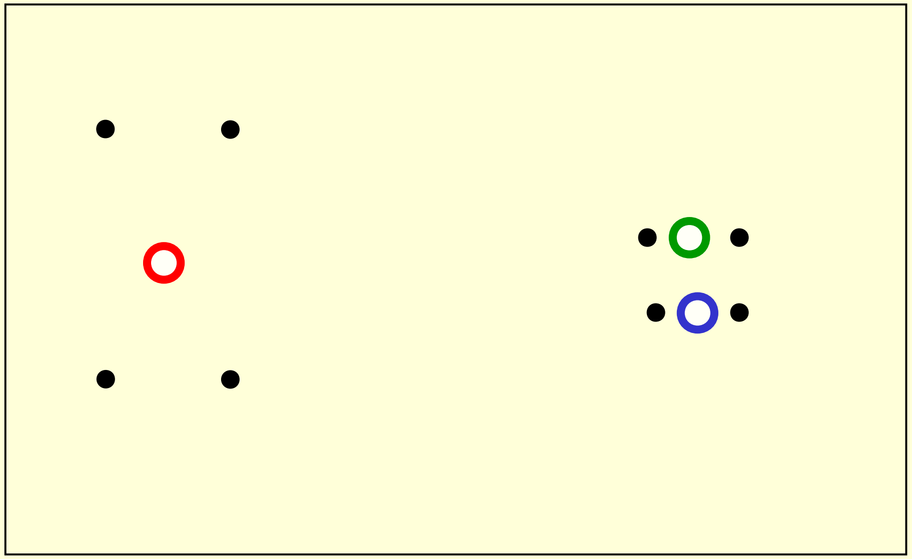
Local minima

- There is nothing to prevent k-means getting stuck at local minima.

- We could try many random starting points

- We could try non-local split-and-merge moves: Simultaneously merge two nearby clusters and split a big cluster into two.

A bad local optimum



Soft k-means

- Responsibilities : responsibility r for the cluster k is set to 1 iff that particular data point is assigned is closest to centre of cluster k

$$r_k^{(c)} = \frac{\exp[-\beta d(\mathbf{m}_k, \mathbf{x}^{(c)})]}{\sum_{k'} \exp[-\beta d(\mathbf{m}_{k'}, \mathbf{x}^{(c)})]}$$

- Update: Move the cluster towards the centre of the gravity of its data

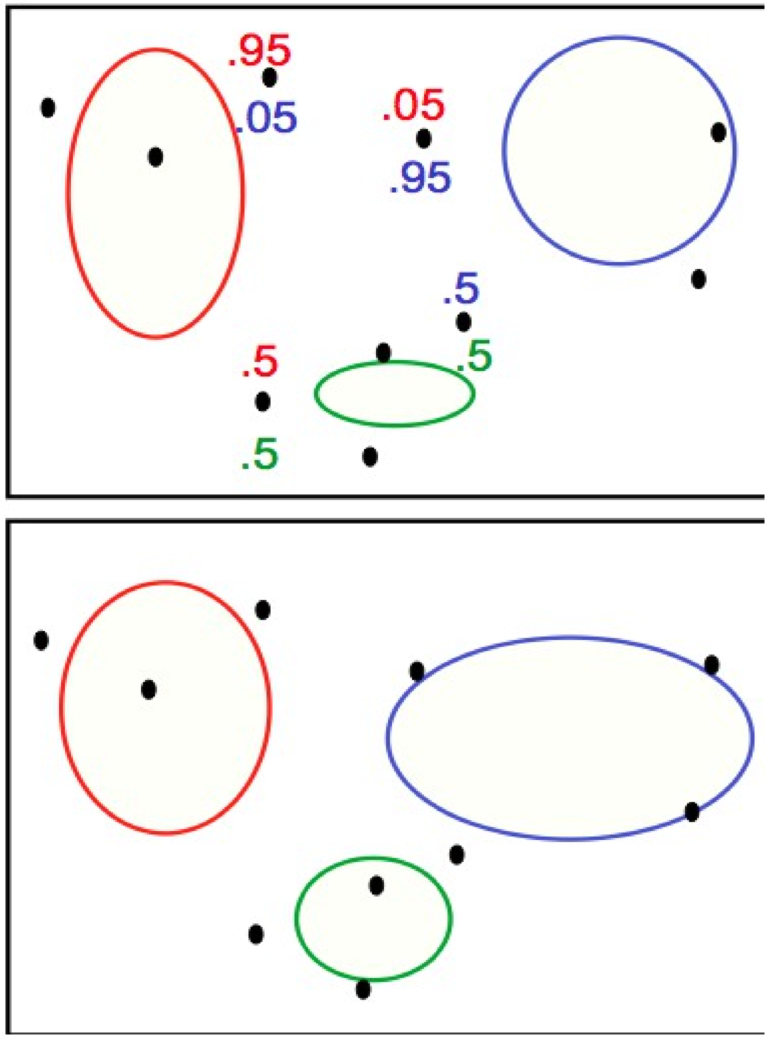
$$\mathbf{m}_k = \frac{\sum_c r_k^{(c)} \mathbf{x}^{(c)}}{\sum_c r_k^{(c)}}$$

A generative view of clustering

- We need a sensible measure of what it means to cluster the data well.
  - This makes it possible to judge different methods.
  - It may make it possible to decide on the number of clusters.
- An obvious approach is to imagine that the data was produced by a generative model.
  - Then we can adjust the parameters of the model to maximize the probability density that it would produce exactly the data we observed.

Fitting a mixture of Gaussians

- Optimization uses the Expectation Maximization Algorithm
- E-step: compute the posterior probability that each Gaussian generates data points
- M-step: Assuming that the data really was generated this way, change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for



The E-step: Computing responsibilities

- In order to adjust the parameters, we must first solve the inference problem: Which Gaussian generated each datapoint?
  - We cannot be sure, so it's a distribution over all possibilities.
- Use Bayes theorem to get posterior probabilities

Prior for Gaussian i

Posterior for Gaussian i

$$p(i \mid \mathbf{x}^{(c)}) = \frac{p(i)\,p(\mathbf{x}^{(c)} \mid i)}{p(\mathbf{x}^{(c)})}$$

Bayes theorem

$$p(\mathbf{x}^{(c)}) = \sum_j p(j)\,p(\mathbf{x}^{(c)} \mid j)$$

$$p(i) = \pi_i$$

Mixing proportion

$$p(\mathbf{x}^{(c)} \mid i) = \prod_{d=1}^{d=D} \frac{1}{\sqrt{2\pi}\sigma_{i,d}} e^{-\frac{\| x_d^{(c)} - \mu_{i,d} \|^2}{2\sigma_{i,d}^2}}$$

Product over all data dimensions

The M-step: Computing new mixing proportion

- Each Gaussian gets a certain amount of posterior probability for each datapoint
- The optimal mixing proportion to use (given these posterior probabilities) is just the fraction of the data that the Gaussian gets responsibility for.

Posterior for Gaussian i

Data for training case c

$$\pi_i^{new} = \frac{\sum_{c=1}^{c=N} p(i \mid \mathbf{x}^{(c)})}{N}$$

Number of training cases

More M-step: Computing the new means

- We just take the center-of gravity of the data that the Gaussian is responsible for.
  - Just like in K-means, except the data is weighted by the posterior probability of the Gaussian.
  - Guaranteed to lie in the convex hull of the data
    - Could be big initial jump

$$\mu_i^{new} = \frac{\sum_c p(i \mid \mathbf{x}^{(c)})\, \mathbf{x}^{(c)}}{\sum_c p(i \mid \mathbf{x}^{(c)})}$$

More M-step: Computing the new variance

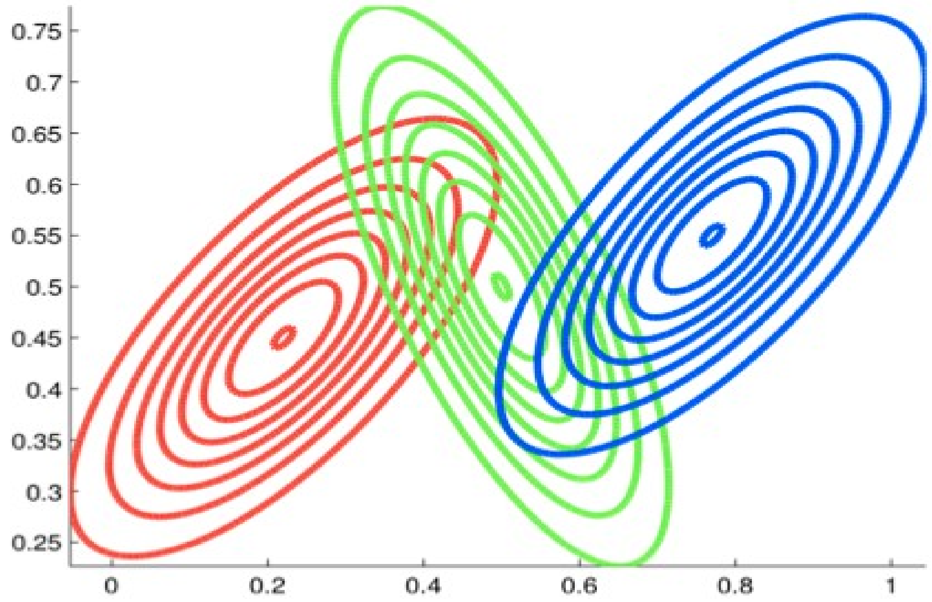
- We fit the variance of each Gaussian, i, on each dimension, d, to the posterior-weighted data.
- The formula only considers diagonal elements of the full-covariance matrix.

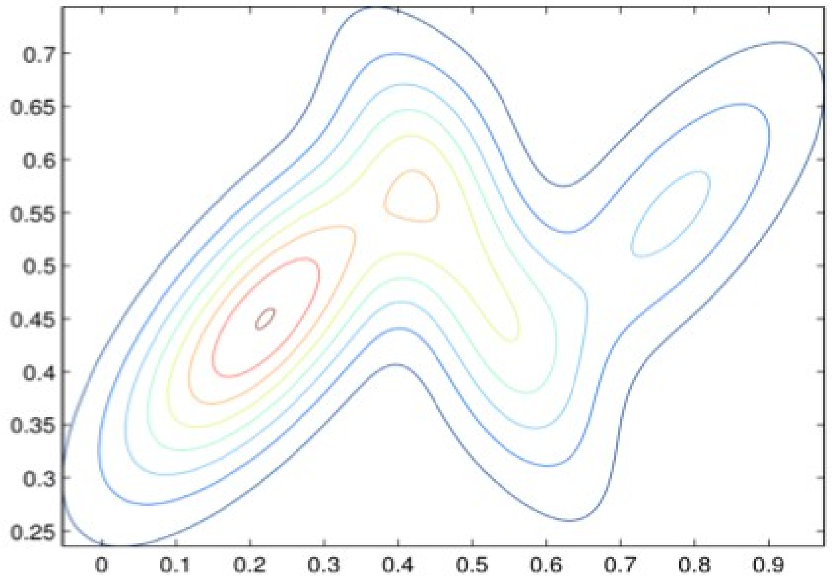
$$\sigma_{i,d}^2 = \frac{\sum_c p(i \mid \mathbf{x}^{(c)}) \; \| x_d^{(c)} - \mu_{i,d}^{new} \|^2}{\sum_c p(i \mid \mathbf{x}^{(c)})}$$

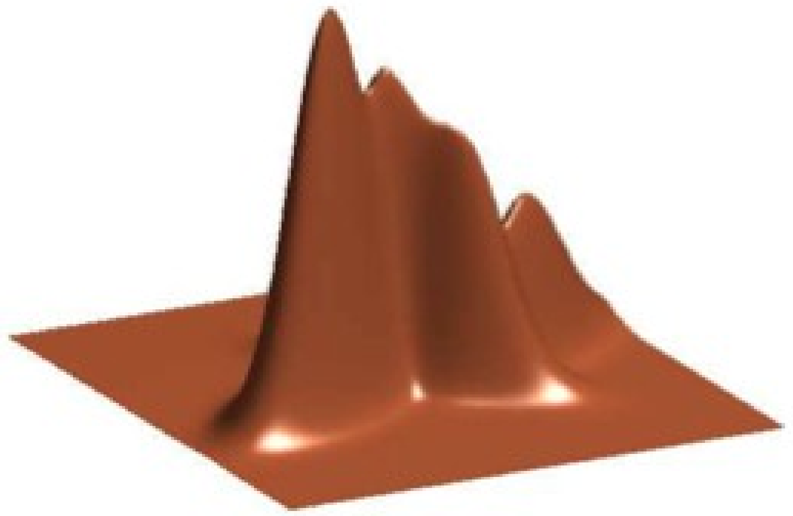
How many Gaussians do we use?

- Hold back a validation set.
  - Try various numbers of Gaussians
  - Pick the number that gives the highest density to the validation set.
- Refinements:
  - We could make the validation set smaller by using several different validation sets and averaging the performance.
  - We should use all of the data for a final training of the parameters once we have decided on the best number of Gaussians.

Visualizing a Mixture of Gaussian







- EM algorithm can easily get stuck in the local minima
- Better to have large Gaussian and gradually decrease

Speeding up the fitting

- If we have huge amounts of data, speed is very important. Some tricks are:
  - Initialize the Gaussians using k-means
    - Makes it easy to get trapped.
    - Initialize K-means using a subset of the datapoints so that the means lie on the low-dimensional manifold.
  - Find the Gaussians near a datapoint more efficiently.
    - Use a KD-tree to quickly eliminate distant Gaussians from consideration.
  - Fit Gaussians greedily
    - Steal some mixing proportion from the already fitted Gaussians and use it to fit poorly modeled datapoints better.

How do we know that the updates improve things?

- Updating each Gaussian definitely improves the probability of generating the data IF we generate it from the same Gaussian after parameter updates.
  - But the posterior will changes when we apply E-step
- A good way to show that updating each Gaussian helps is to show that there is a single function that is improved by both EM step.
  - Free energy

Why EM converges

- There is a cost function that is reduced by both the E-step and the M-step.

  Cost = expected energy – entropy

- The expected energy term measures how difficult it is to generate each datapoint from the Gaussians it is assigned to. It would be happiest giving all the responsibility for each datapoint to the most likely Gaussian (as in K-means).

- The entropy term encourages “soft” assignments. It would be happiest spreading the responsibility for each datapoint equally between all the Gaussians.

The expected energy of the datapoint

- The expected energy of datapoint c is the average of negative log probability of generating the data
  - The average is taken using the probabilities of assigning the datapoint to each Gaussian
  - What probability distribution should we use for the distribution q

probability of assigning c to Gaussian i

parameters of Gaussian i

$$\sum_{c} \sum_{i} q(i \mid \mathbf{x}^{(c)}) \left(-\log \pi_i - \log p(\mathbf{x}^{(c)} \mid \mu_i, \sigma_i^2)\right)$$

data-point

Gaussian

Location of datapoint c

The entropy term

- This term wants the responsibilities to be as uniform as possible.
- It fights the expected energy term.

$$\text{entropy} = -\sum_{c} \sum_{i} q(i \mid \mathbf{x}^{(c)}) \underset{\substack{\uparrow \\ \text{log probabilities are} \\ \text{always negative}}}{\log} q(i \mid \mathbf{x}^{(c)})$$

The E-step chooses assignment probabilities that minimize free energy

- The assignment probability of datapoint that minimizes the cost and sums to 1
- The optimal solution to the trade-off between expected energy and entropy is to make the probabilities be proportional to the exponential of negative energies:

$$\text{energy of assigning } c \text{ to } i \;=\; -\log \pi_i - \log p(\mathbf{x}^{(c)} \mid \mu_i, \sigma_i^2)$$

$$\text{optimal value of } q(i \mid \mathbf{x}^{(c)}) \;\propto\; \exp(-\text{energy}) \propto \;\pi_i \; p(\mathbf{x}^{(c)} \mid i)$$

- So using the posterior probabilities as assignment probabilities minimizes the cost function

The M-step chooses the parameters that minimize the cost function (with the responsibilities held fixed)

- This is easy. We just fit each Gaussian to the data weighted by the responsibilities that the Gaussian has for the data.
  - When you fit a Gaussian to data you are maximizing the log probability of the data given the Gaussian. This is the same as minimizing the energies of the datapoints that the Gaussian is responsible for.
  - If a Gaussian has a responsibility of 0.7 for a datapoint the fitting treats it as 0.7 of an observation.

- Since both the E-step and the M-step decrease the same cost function, EM converges.

- We can think of E-step as finding the best distribution over hidden configurations for each data points
- The M-step holds the distribution fixed and minimizes the F by changing the parameters that determines the energy of a configuration

- Credit for slides: Richard Zemel, Geoffery Hinton
- I have adapted their slides for our class