Segmentation as clustering

- Cluster together (pixels, tokens, etc.) that belong together
- We assume that we can compute how close tokens are, or how close a token is to cluster.

Clustering and dimensionality

- A simple clustering method that does not make sense in high dimensions
  - Partition space into hypercubes of edge size 1/K
  - Put points into the appropriate cube
  - Most cubes do not get used (there are too many of them!)
- Real data lives in low dimensional manifolds (plus noise perturbations)
  - Most of a high dimensional space is not used
- A distance function takes two high dimensional points and maps them into a single number, which looses a lot of information about the points.
- Non-intuitive fact --- points in a cluster tend to be about the same distance away from the center

K-Means (continued)

K-means flow chart

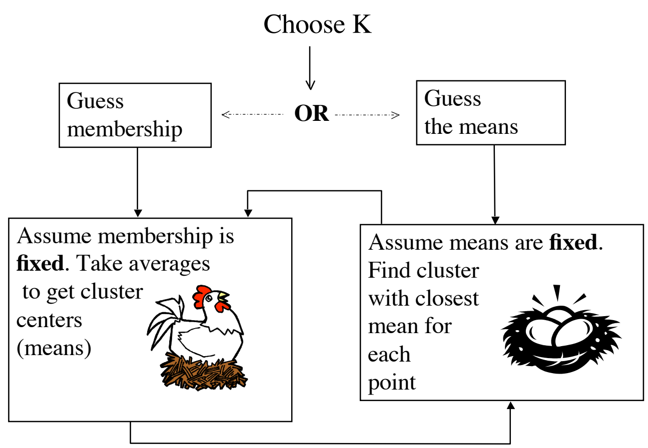
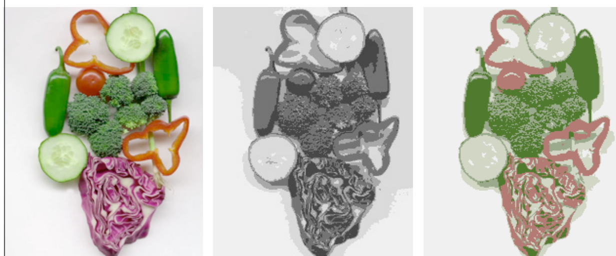


Image | Clusters on intensity | Clusters on color

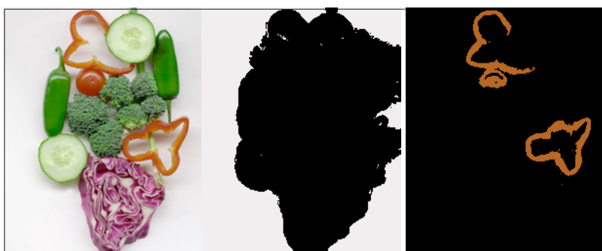


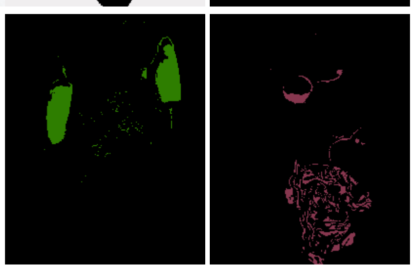
K-means clustering using intensity alone and color alone
(Assuming 5 segments, i.e. k=5)

Image

Clusters on color

K-means using color alone, 11 segments





K-means using
color alone,
11 segments.

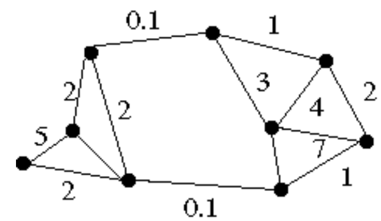
Notes on K-Means

- K-means is “hard” clustering-each point is completely in exactly one cluster
- What you get is a function of starting “guess”
- The error goes down with every iteration
  - This means you get a local minimum
- Unfortunately, the dimension of the space is usually large, and high-dimensional space have lots of local maximum (standard problem!)
  - Dimensionality here is K*dim(**x**)
- Finding the global minimum for a real problem is very optimistic!

you should be able to argue why this is true

Graph theoretic clustering

- Represent distance between tokens using a weighted graph.
  - affinity matrix
- Cut up this graph to get subgraphs with strong interior links (and weak links between the subgraphs).



Graph for 9 tokens

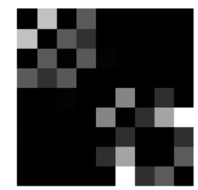
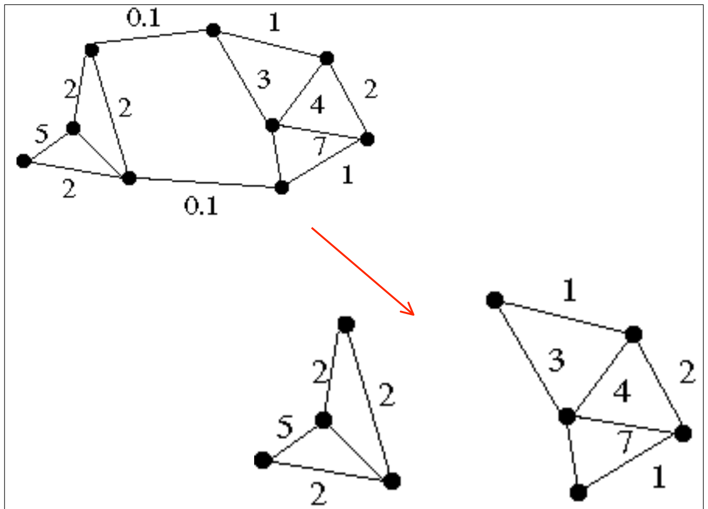


Image representation of weight matrix

(Note that the point ordering is conveniently chosen)



Measuring Affinity

Intensity

$$aff(x,y) = \exp\left\{ -\left(\frac{1}{2\sigma_i^2} \right) \left(\|I(x) - I(y)\|^2 \right) \right\}$$

Distance

$$aff(x,y) = \exp\left\{ -\left(\frac{1}{2\sigma_d^2} \right) \left(\|x - y\|^2 \right) \right\}$$

Texture

$$aff(x,y) = \exp\left\{ -\left(\frac{1}{2\sigma_t^2} \right) \left(\|c(x) - c(y)\|^2 \right) \right\}$$

Texture Descriptor

Eigenvectors and cuts

- For some cluster, consider a vector **a** giving the association between each element and that cluster
- We want elements within this cluster to, on the whole, have strong affinity with one another
- If two elements, i and j, are part of the same cluster, then
  – $\mathbf{a}_i$ and $\mathbf{a}_j$ are both large
  – and the affinity A_{ij} is large
  – thus, $\mathbf{a}_i A_{ij} \mathbf{a}_j$ should be large
- Thus a good cluster is one where $\sum_i \sum_j a_i A_{ij} a_j$ is large.

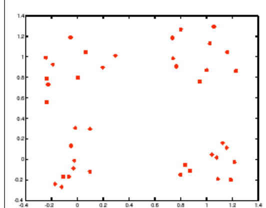
Eigenvectors and cuts

- $\sum_i \sum_j a_i A_{ij} a_j$ should be large for a coherent cluster represented by **a**.
- This suggests maximizing $\mathbf{a}^T A \mathbf{a}$
- But we need the constraint $\mathbf{a}^T \mathbf{a} = 1$ (why?)
  – Arguably it might be more logical to make the sum of the elements of **a** to be one, but the standard (L_2) norm is easier to deal with.

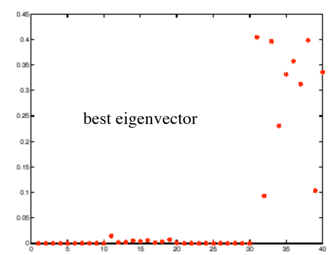
Eigenvectors and cuts

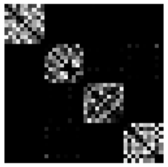
- We want to maximize $\mathbf{a}^T A \mathbf{a}$ subject to $\mathbf{a}^T \mathbf{a} = 1$
- This is an eigenvalue problem - choose the eigenvector of A with largest eigenvalue
- This gives the cluster with greatest internal affinity
  – Ideally, most elements of the eigenvalue are near zero, and the others tell us which tokens are in the cluster

Example eigenvector



points





affinity matrix

(you should know how to interpret these)

(Note that the point ordering is conveniently chosen)