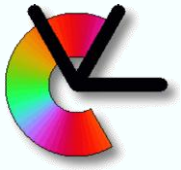
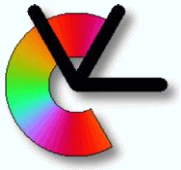


Spectral clustering
Lecture 3

Vasileios Zografos
zografos@isy.liu.se

Klas Nordberg
klas@isy.liu.se



Contents

- **Applications of spectral clustering**
  - Mainly computer vision (image, shape and motion segmentation)
- **Practical issues**
  - Parameter tuning
  - Number of clusters
  - Large affinity matrices
  - K-means modifications (Mahalanobis vs spherical)
- **Spectral clustering extensions**
  - Multi-way affinities
  - Multiple affinities

Some basic applications

- Image segmentation
- Image clustering
- Motion segmentation
- Shape extraction
- Point correspondence
- Anywhere else there is need for segmentation or clustering

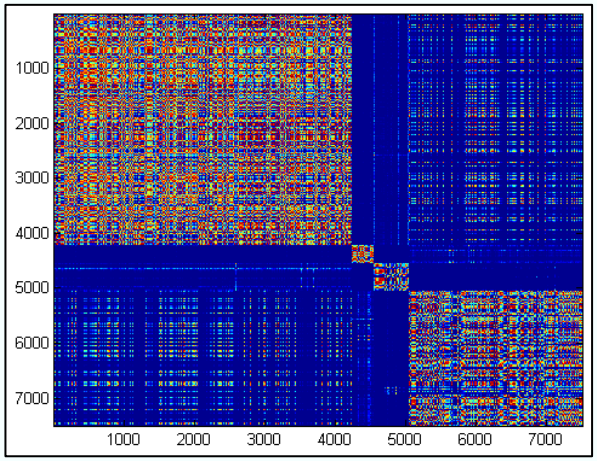
- **General principle:**
  - Determine what needs to be clustered/segmented – feature type
    - e.g. pixels, points of interest, regions, or whole images
  - Extract an appropriate descriptor with a distance measure appropriate for the problem
  - Create affinity matrix
  - Carry out spectral clustering

Simple image segmentation

- 100x75 colour image
- Features: RGB pixels in R^3

- Distance: $D(i,j) = ||x_i - x_j||_2$, Affinity: $A(i,j) = exp(-D(i,j)^2/\sigma^2)$
- 7500x7500 affinity matrix (too large! Ask for the k largest eigenvalues)
- Spectral clustering with 4 clusters

- SC at the pixel level will not scale well for larger images.
- And of course simple RGB distance criterion will fail for more complicated images



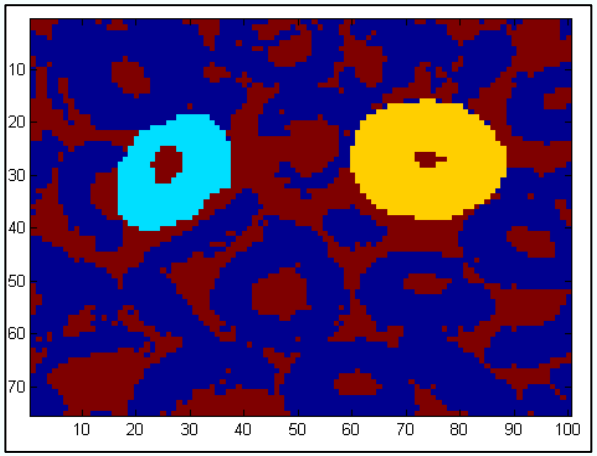


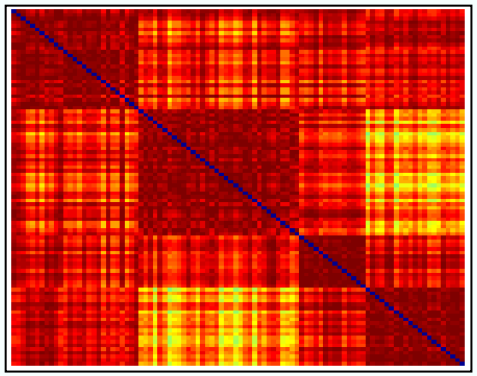
Image clustering

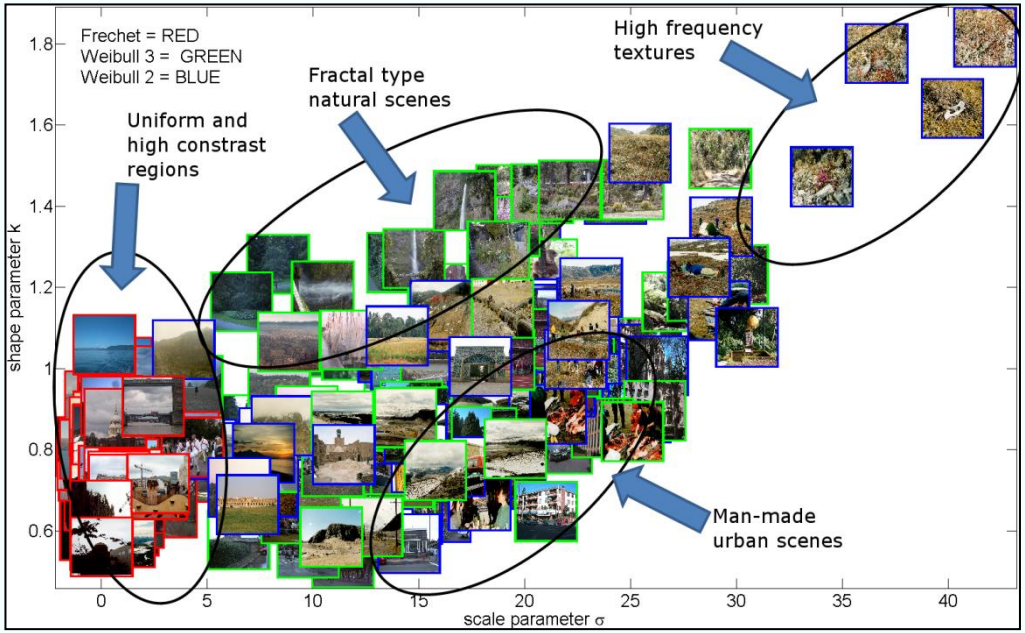
- Cluster based on texture property (e.g. frequency). One can also cluster on object-type

- Database of **N** images
- For each image extract a holistic texture-type feature (e.g. Gist, Weibull parameters, a distribution of texture variance etc)

- Define a pairwise distance criterion in the embedding space (for Weibull it is R^2)
  - Natural choice ***Rao distance*** between two Weibulls
  - Or a divergence measure between distributions

- Build **NxN** affinity matrix
- Spectral clustering with k clusters
- The choice of k is usually arbitrary





Simple motion segmentation

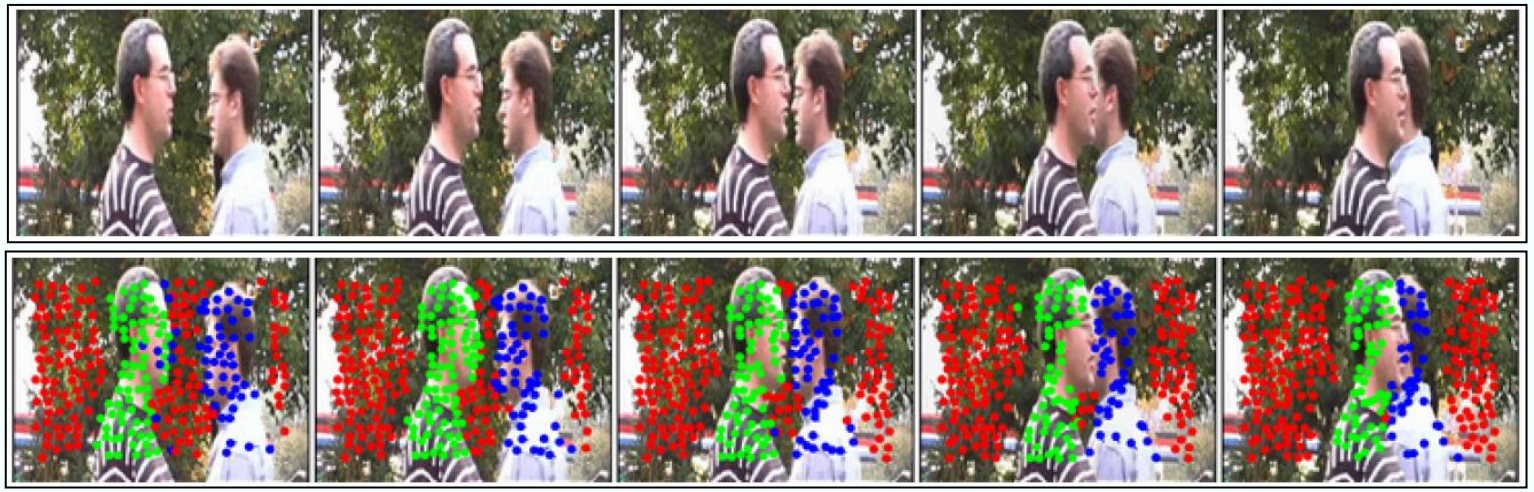
- Extract **N** sparse features (e.g. Harris points ***p***) and determine correspondence across image ***F*** frames (e.g. KLT tracker)
- Define a pairwise affinity. For example from the local pairwise distances….

$$D(i,j) = \sum_t ||p_i - p_j||_t^2 \qquad D(i,j) = \max_t D_t(i,j) \qquad D_t(i,j) = \overline{D}\,||p_i - p_j||_t^2 / c_t$$

….the usual affinity $$A(i,j) = \exp(-D(i,j)^2/\sigma^2)$$

- Build **NxN** affinity matrix and do spectral clustering

- Simple pairwise affinities will only work well for 2D translational models.
- For robust segmentation we need to look for 3D motion = multi-way affinities in 2D

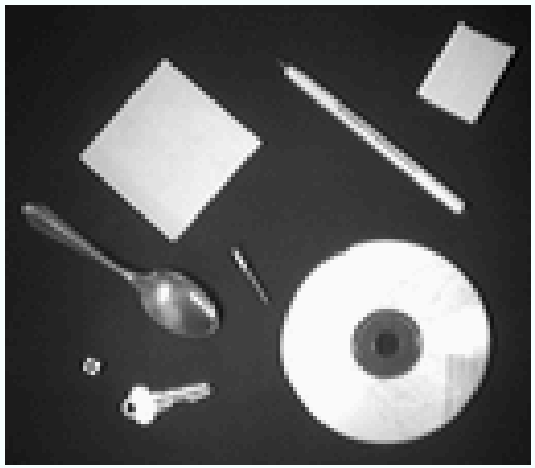
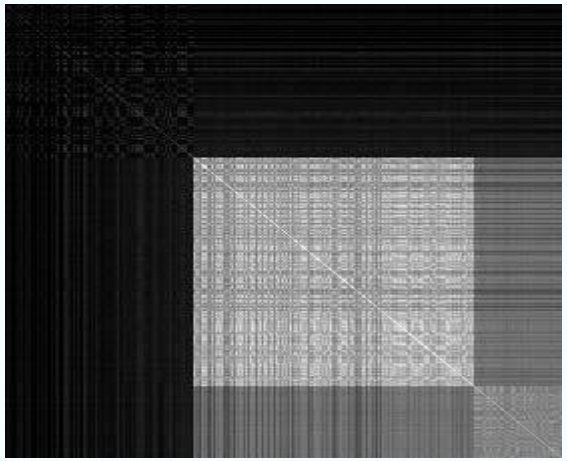
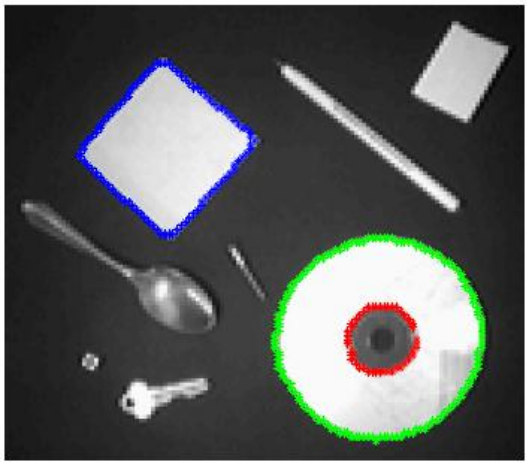


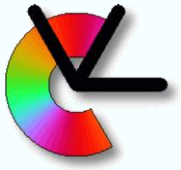
Simple shape extraction

- Really one has to use **multi-way affinities** here (we will discuss that later)

- Extract point features in the image (e.g. points of interest)
- Decide on a geometric shape model (e.g. conics, lines, contours or something more advanced)
- If the model needs $\boldsymbol{d}$ points, take at least $\boldsymbol{d+1}$ and fit the model
- **The residual is the affinity. But a $\boldsymbol{d+1}$-wise affinity**
- Convert to **pairwise affinity** matrix $\boldsymbol{A(i,j)}$ (we will discuss that later)
- Do spectral clustering

- How does it compare with RANSAC type methods?



Parameter tuning

- The kernel parameter(s) are amongst the most important parameters in SC
  - They supress or enhance affinities between points
  - Essentially “compacting” or “stretching” clusters
  - Can make a big difference in clustering quality (depending on data complexity and kernel choice)
- Choice of kernel parameter is data dependent – choosing a fixed value is not recommended for different datasets
- Generally we have to search for them, although there are certain fixed choices
- Global vs local
- Here we will deal with single parameter kernels (i.e. Gaussian and σ, kNN and k, etc)

Parameter tuning : An example

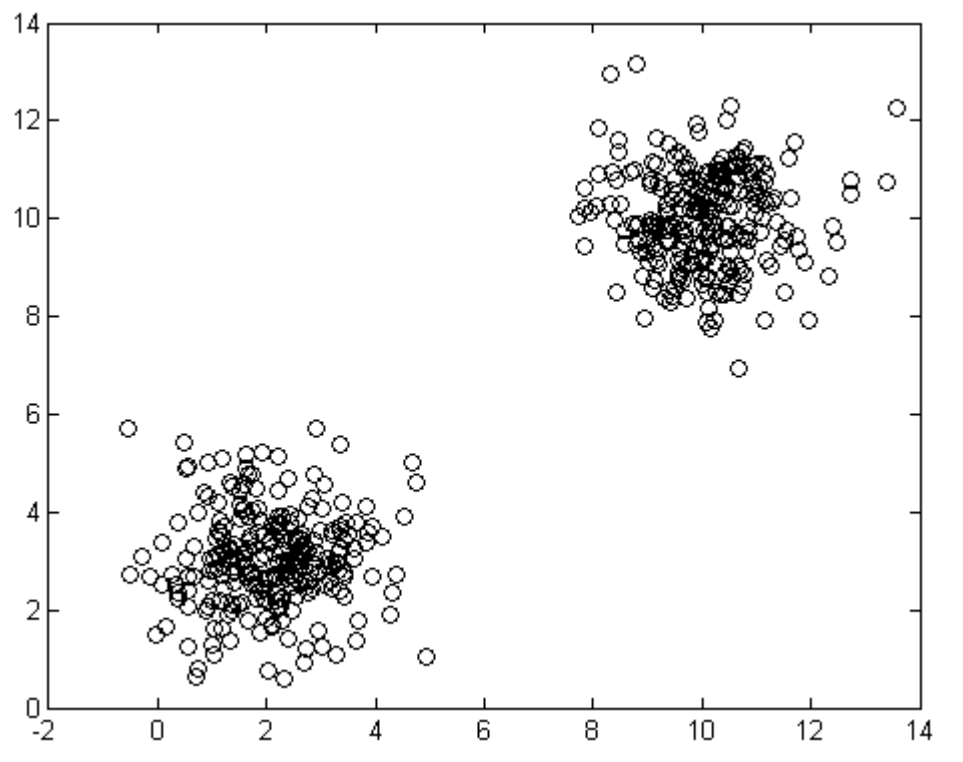
An easy problem: Two well separated clusters

$$\mu_1 = (2,3), \Sigma_1 = [1,0;0,1]$$

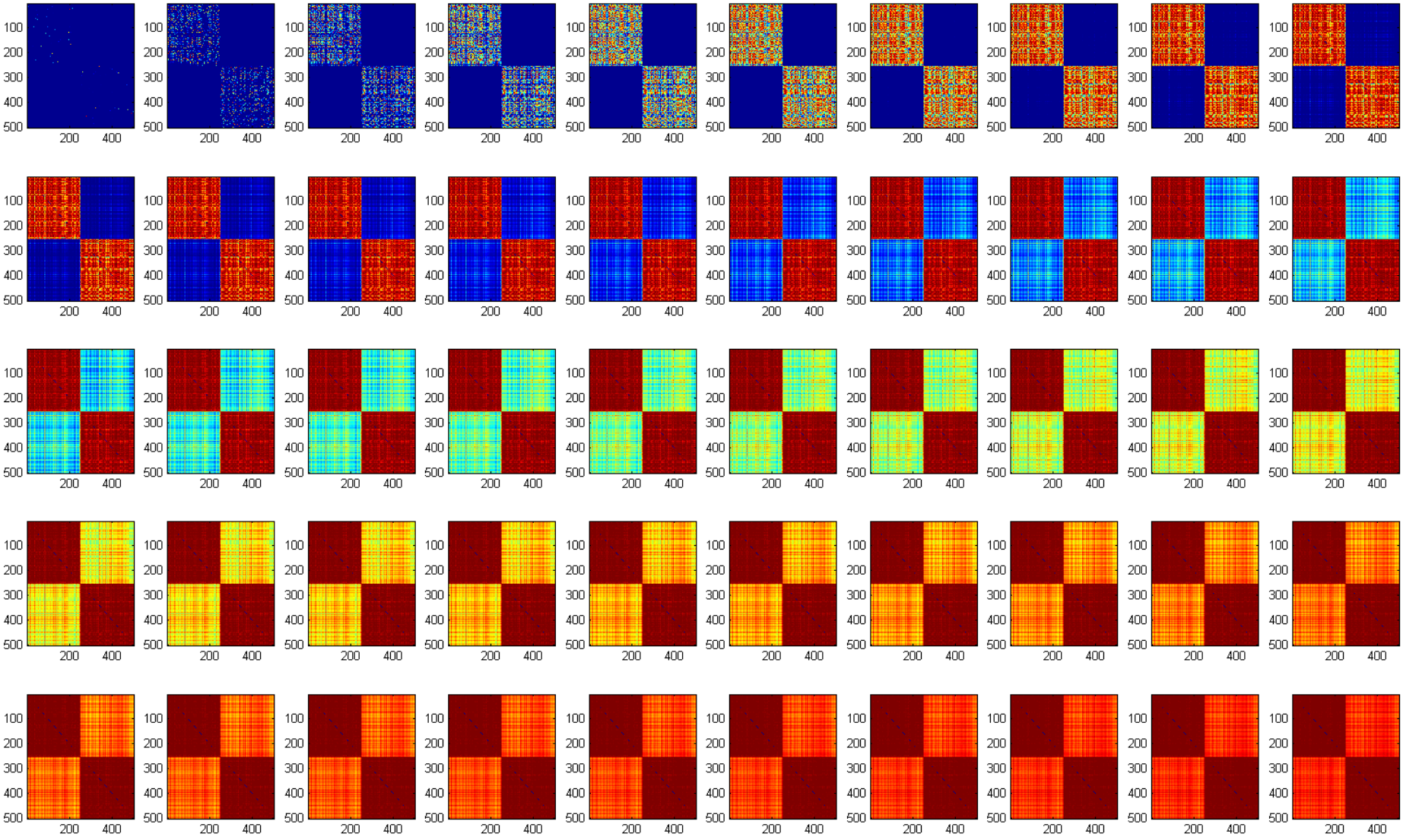
$$\mu_2 = (10,10), \Sigma_2 = [1,0;0,1]$$

$$A(i,j) = exp(-\frac{||x_i - x_j||^2}{\sigma^2})$$

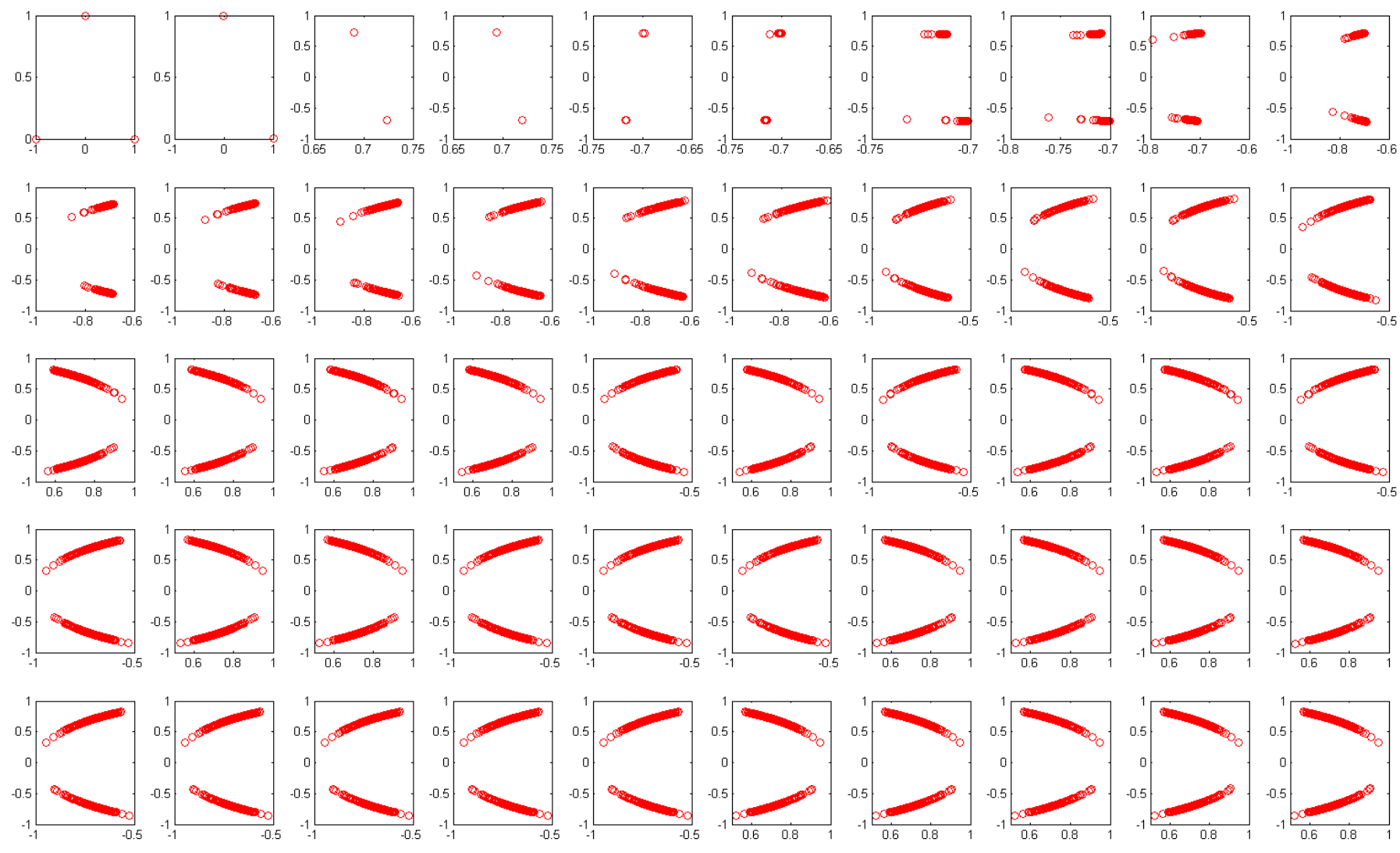
- We take 50 equidistant σ samples from [0.1,…,10]
- At each sample, we calculate the affinity matrix and perform spectral clustering
- We then evaluate the ground truth clustering error



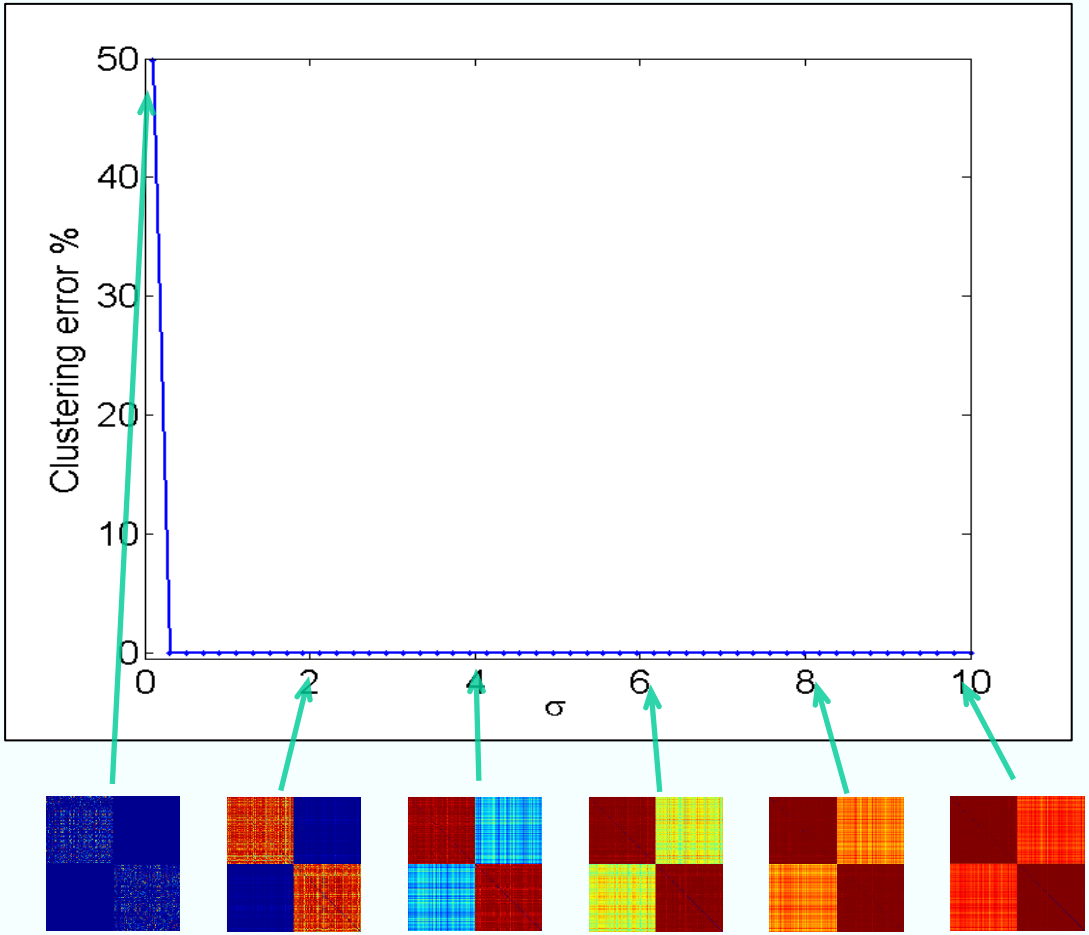
Parameter tuning : An example



Parameter tuning : An example



Parameter tuning : An example



No problem choosing σ in this case

Parameter tuning : An example

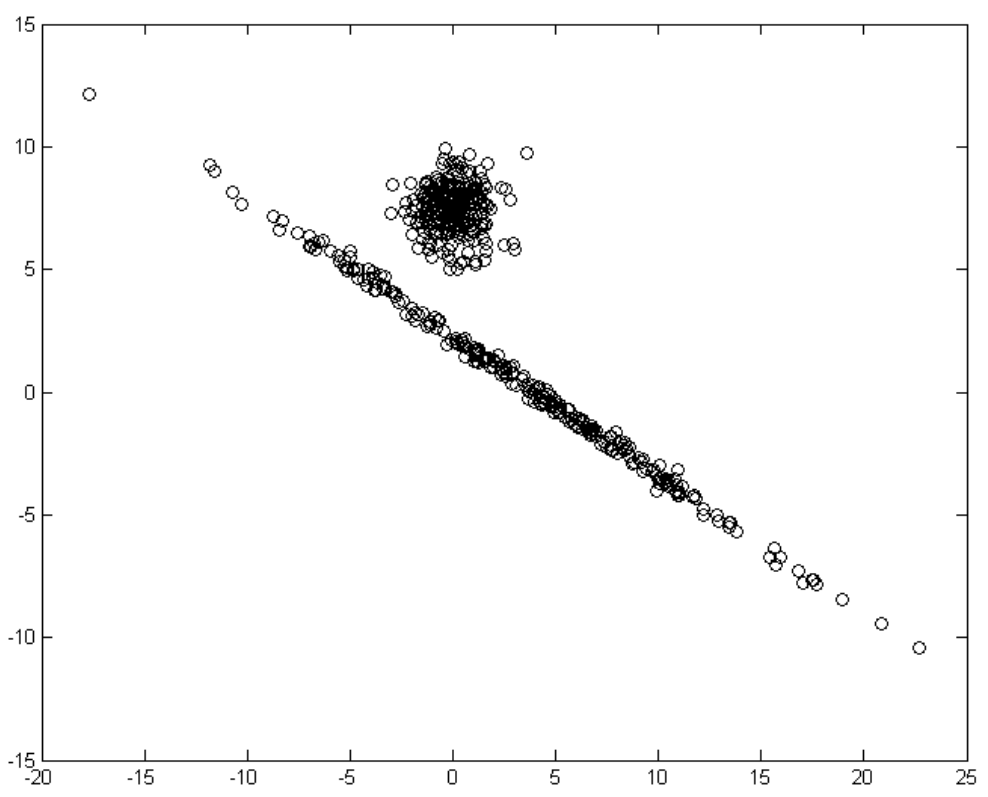
A harder problem: Two proximal clusters of different shapes

$$\mu_1 = (0, 7.5), \Sigma_1 = [1, 0; 0, 1]$$

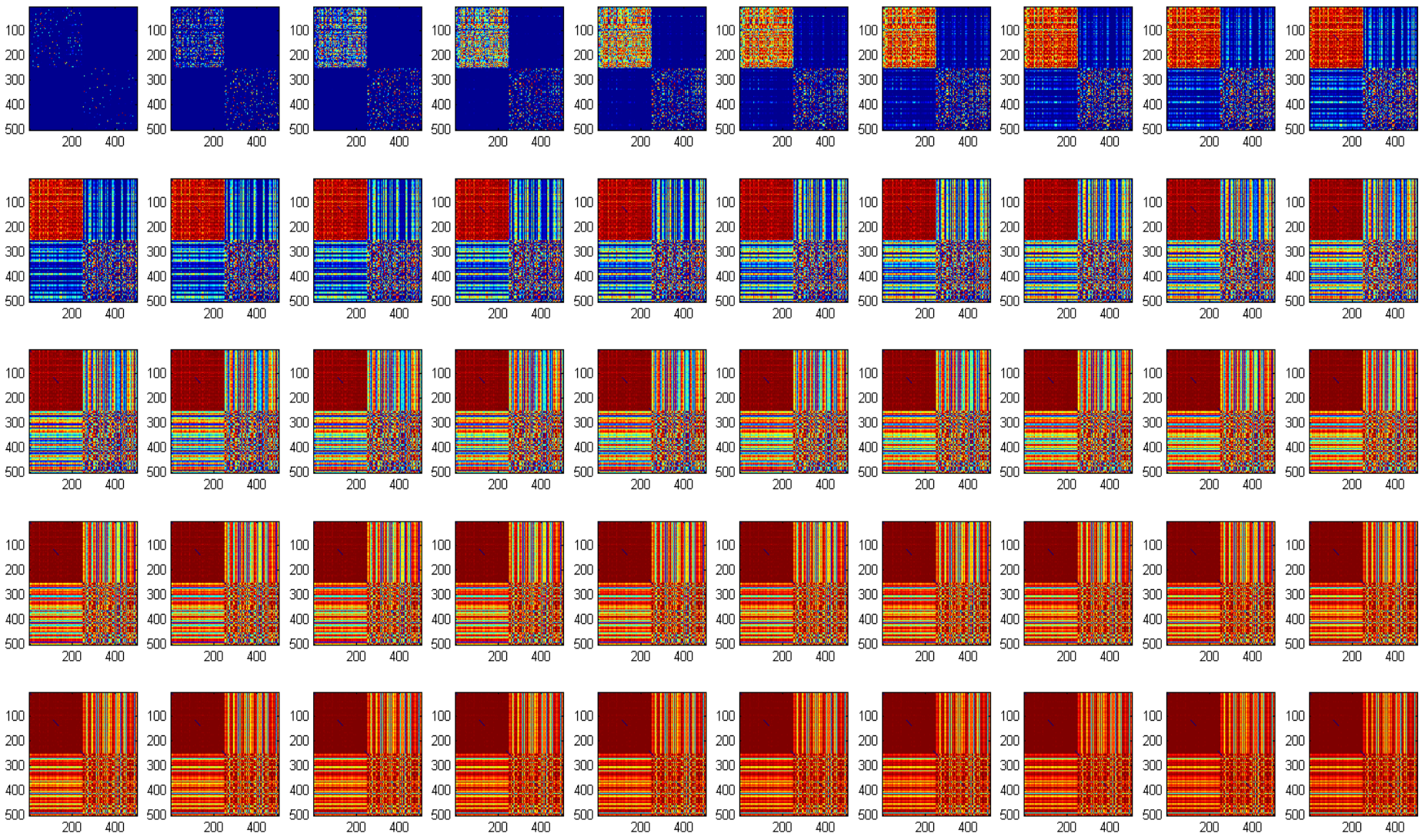
$$\mu_2 = (4, 0), \Sigma_2 = [45, -25; -25, 14]$$

$$A(i,j) = exp(-\frac{||x_i - x_j||^2}{\sigma^2})$$

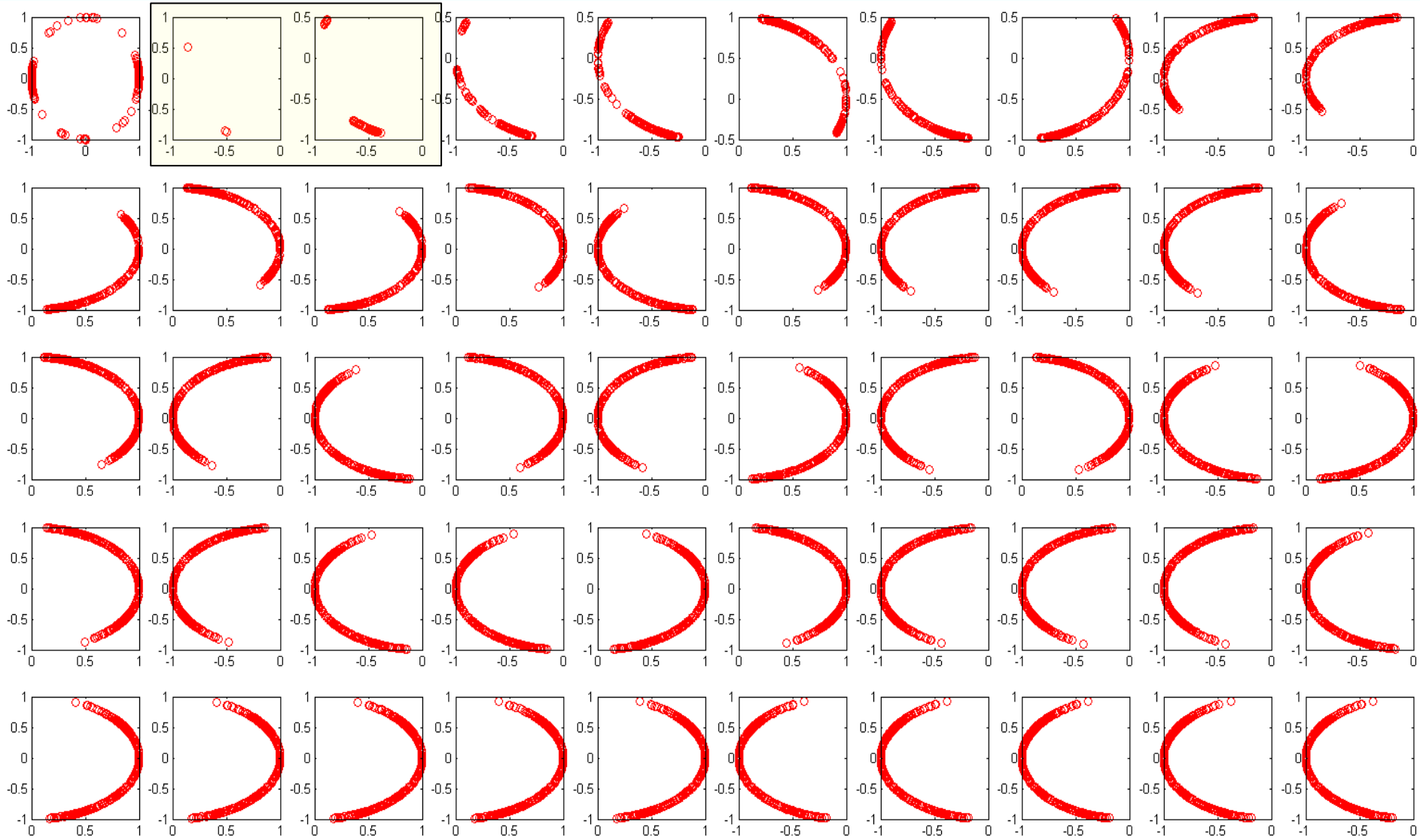
- We take 50 equidistant σ samples from [0.1,…,25]
- At each sample, we calculate the affinity matrix and perform spectral clustering
- We then evaluate the ground truth clustering error



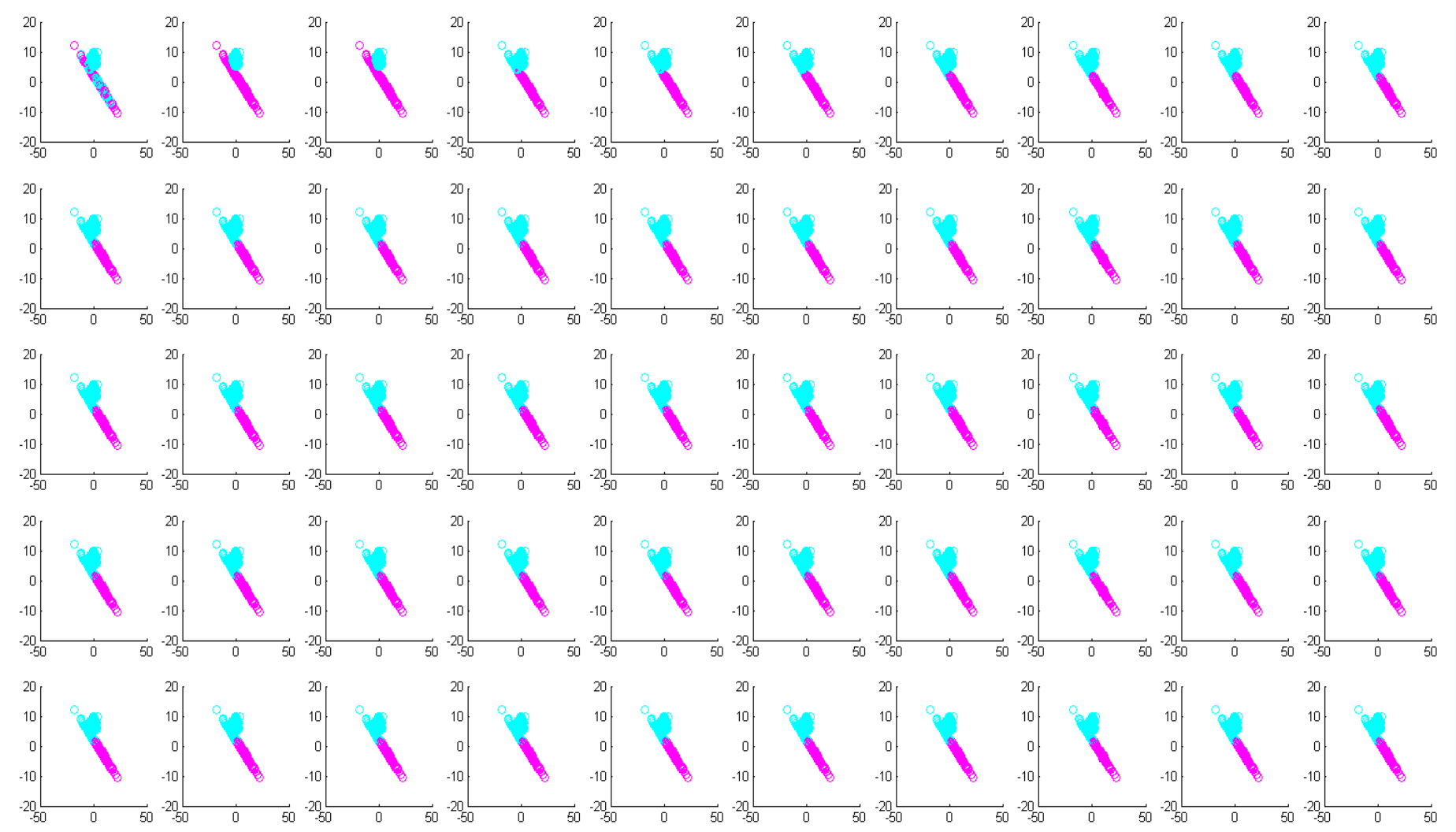
Parameter tuning : An example



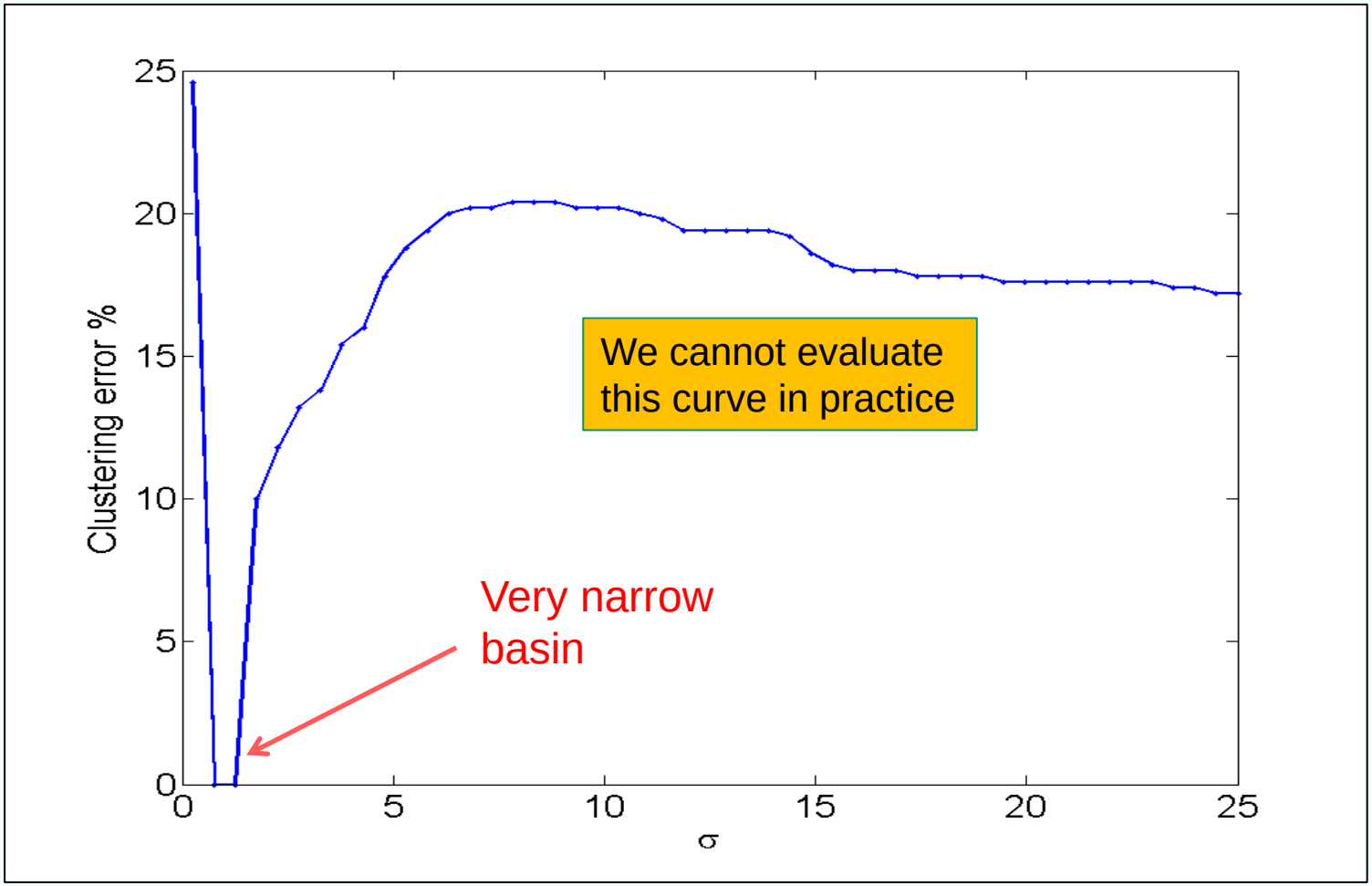
Parameter tuning : An example



Parameter tuning : An example



Parameter tuning : An example



How do we choose σ in this case?

Parameter tuning

- Search for a global σ which gives the “best” clusters [Ng et al]
- What is a good cluster?
  - Recall cluster quality measures from Lecture 1
- Either use **generic** cluster measures
- Or use a **problem-specific** cluster quality measure
- What about a fixed data dependent σ?
  - For Gaussian affinities and isotropic clusters use

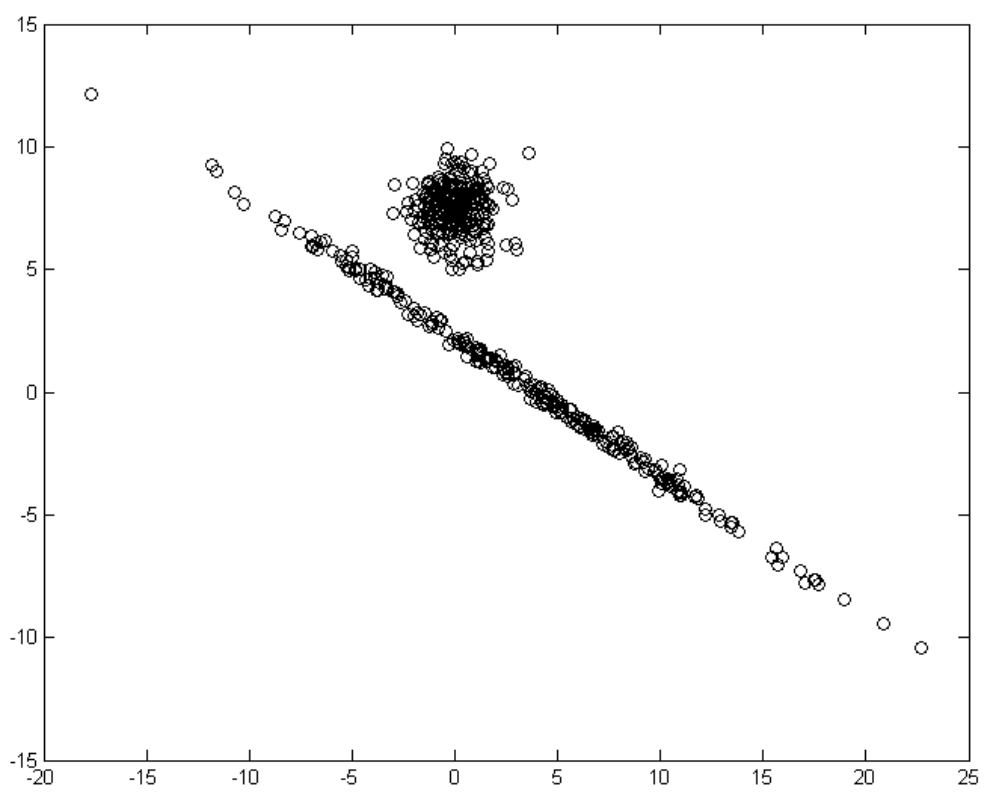
$$\sigma = \frac{D}{N^{\frac{1}{k}}} \quad \text{where} \quad D = \max ||x_i - x_j||$$

  - But not as good as [Ng et al] approach

Parameter tuning : Example revisited

A harder problem: Two proximal clusters of different shapes

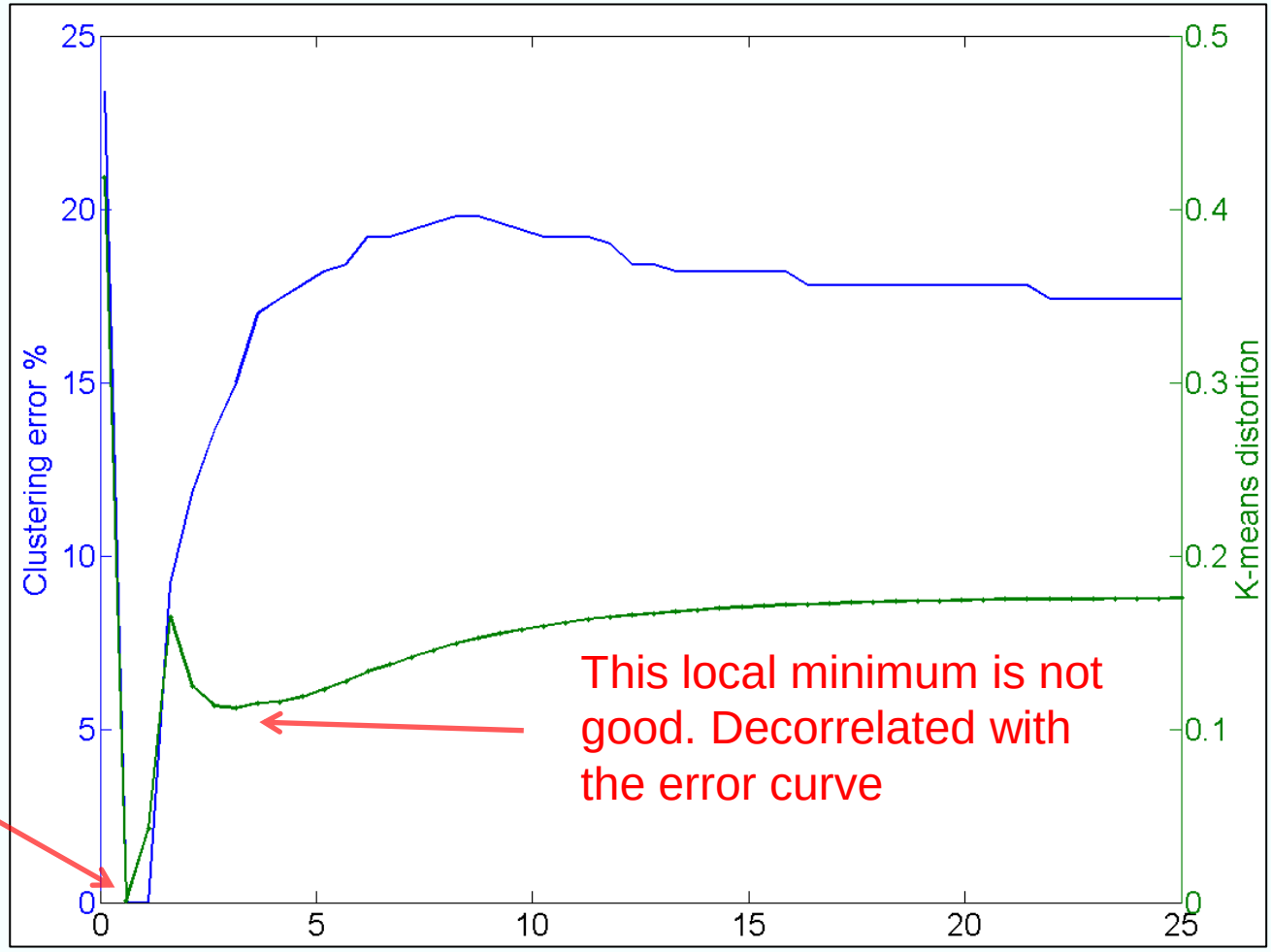
- We take 50 equidistant σ samples from [0.1,…,25]
- At each sample, we calculate the affinity matrix and perform spectral clustering
- But now after each clustering we evaluate a quality measure
- We then evaluate the ground truth clustering error



Parameter tuning : Example revisited

Generic quality criterion ➡ K-means distortion error

$$\sum_{i=1}^{k} \sum_{x_j \in C_i} ||x_j - \mu_i||^2$$



Global minimum found.
But very narrow.
We might miss it

This local minimum is not good. Decorrelated with the error curve

Parameter tuning : Example revisited

- Can we do something better?
  - Yes. Use a problem specific measure
- **Example of existing problem assumption**
  - **Say we know that there should be an isotropic cluster around (0,5)**
  - Set $\hat{\mu_1} = (0,5), \hat{\Sigma}_1 = [1,0;0,1]$

- Then calculate **Bhattacharyya distance** between two multivariate Gaussians

$$B(p_1,p_2) = \frac{1}{8}(\mu_1-\mu_2)^T P^{-1}(\mu_1-\mu_2) + \frac{1}{2}\ln\frac{det(P)}{\sqrt{det(\Sigma_1)det(\Sigma_2)}}$$

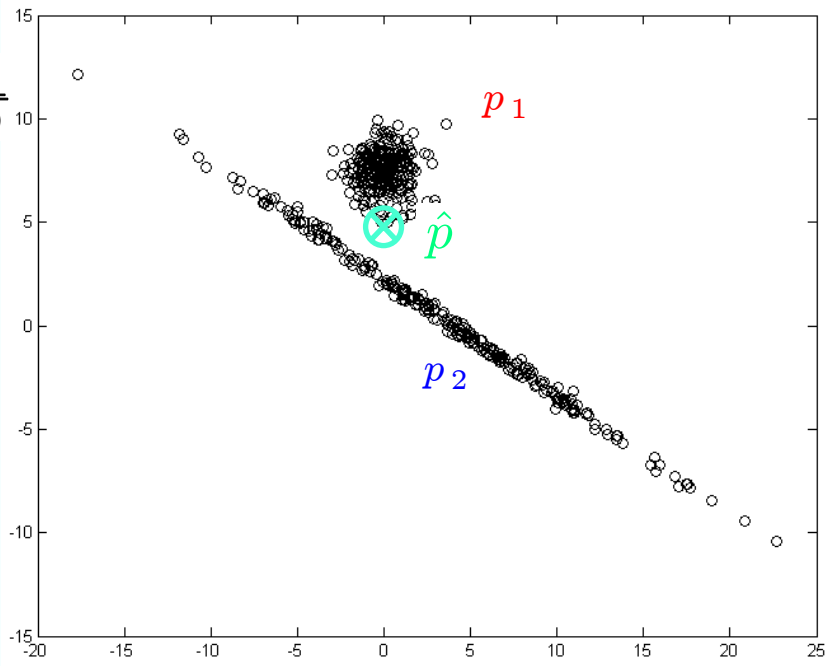
$$P = (\Sigma_1+\Sigma_2)/2$$

- Remember labels might change!! So set quality as $\min(B_1,B_2)$ where

$$B_1 = B(p_1,\hat{p}) \quad B_2 = B(p_2,\hat{p})$$

- The estimates for each cluster at each iteration are given by

$$\mu = mean(X) \quad \Sigma = cov(X)$$

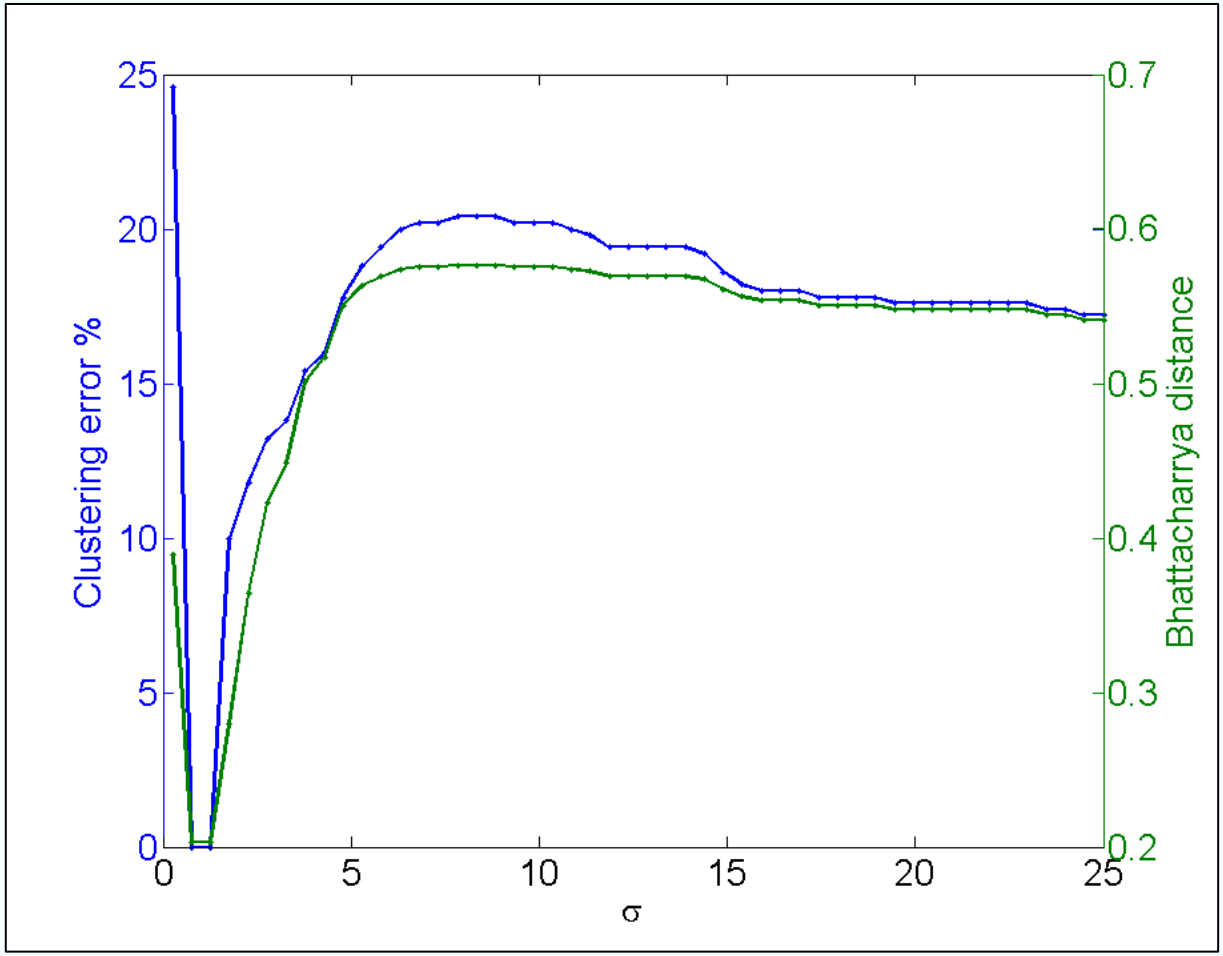


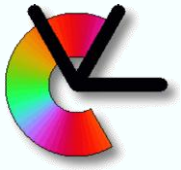
Parameter tuning : Example revisited

Problem specific measure:

- Now the two curves are better correlated.
- The more specific the assumptions the more correlation

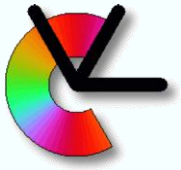
But minimum still narrow





Parameter tuning

- Ideally we would like to have a clustering quality criterion that behaves exactly as the ground truth clustering error (i.e. correlated)
  - Many choices. Some are better than others.
- In most cases the generic solutions will work ok, but really one should use a problem specific
- Clustering quality criteria = at the cluster level
  - Other problem specific information at the data point level it should go to the affinity matrix
- Unfortunately it is not always possible to come up with problem specific quality measures



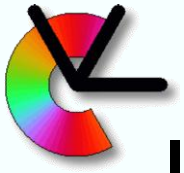
Parameter tuning

- Use your favourite optimisation method or just brute force. Might become very expensive for multiple-parameter kernels

- Error surface non-smooth and local minima
  - Due to the distributions in eigen-space and k-means behaviour

Improving the problem

- Of course the assumption is that there is a good global minimum in the clustering error function in the first place
- We would also like a wide basin around the minimum
- What can we do?
  - **Generic way** (Enhancement: local σ, the Q matrix)
  - **Problem specific way** (Kernel choice + going to multiway affinities)
  - **Incorporate additional info** (Multiple affinities, prior information = constrained spectral clustering)



Improving the problem: Local σ

- Different clusters have different local statistics
- A single, global σ might not work well of all data

- [Zelnik-Manor, Perona] Instead build the affinity matrix

$$A(i,j) = exp(-\frac{||x_i - x_j||^2}{\sigma_i \sigma_j})$$

- Allows self-tuning from the local statistics of the neighbourhood around the points x_i, x_j

- Chose $\sigma_i = d(x_i, x_K)$ where x_K is the K-th neighbour of point x_i

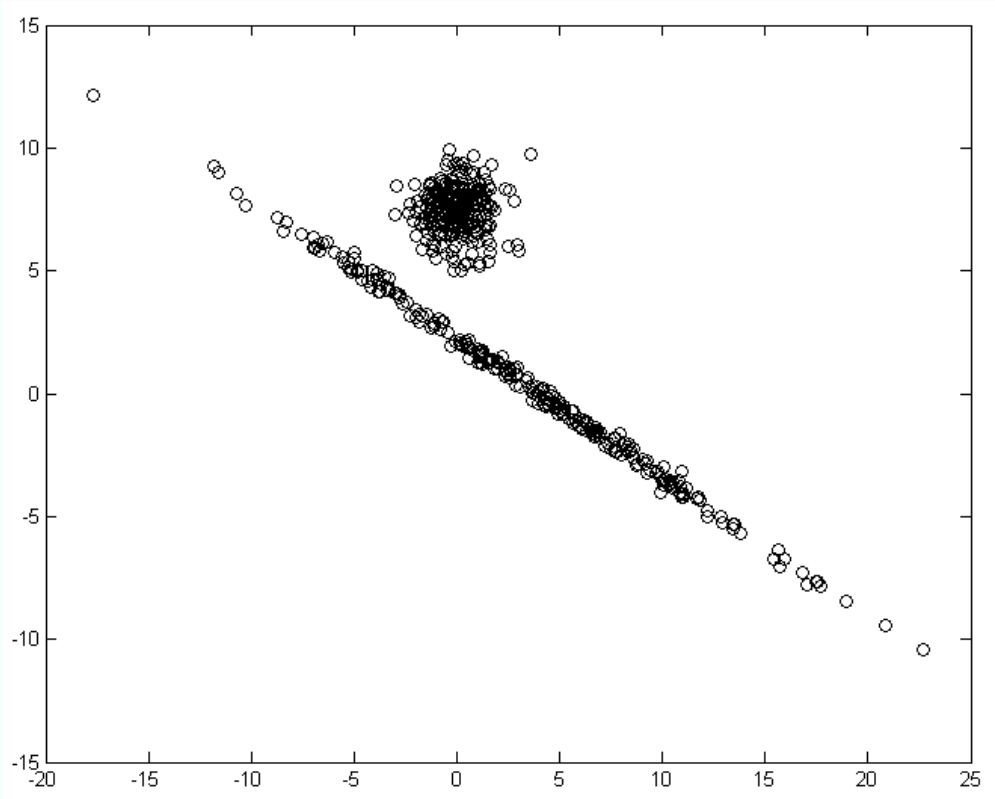
- We can have a fixed K or do a similar search as [Ng et al] over K

- Obviously a data dependent σ must be selected from the original distances BEFORE we build an affinity matrix

Parameter tuning : Example revisited

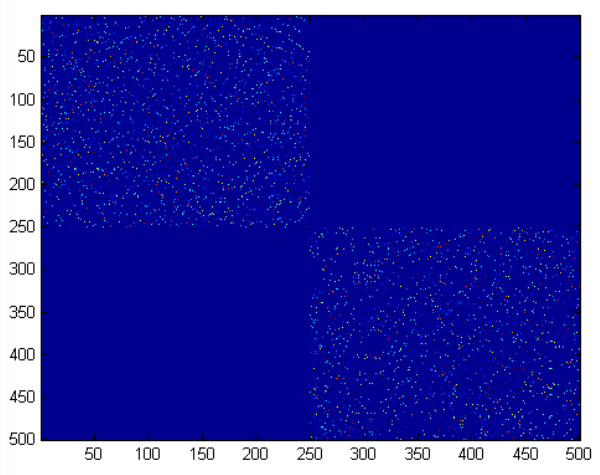
A harder problem: Two proximal clusters of different shapes

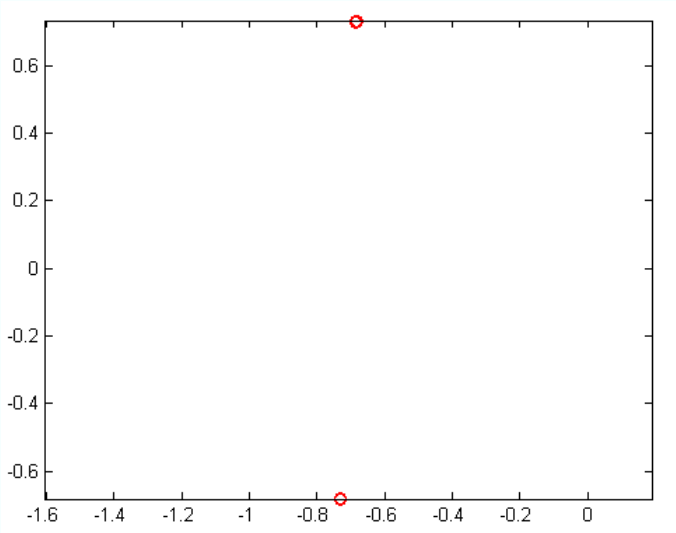
- We calculate one affinity matrix with local σ instead and perform spectral clustering
- [Zelnik-Manor, Perona] suggest K=7

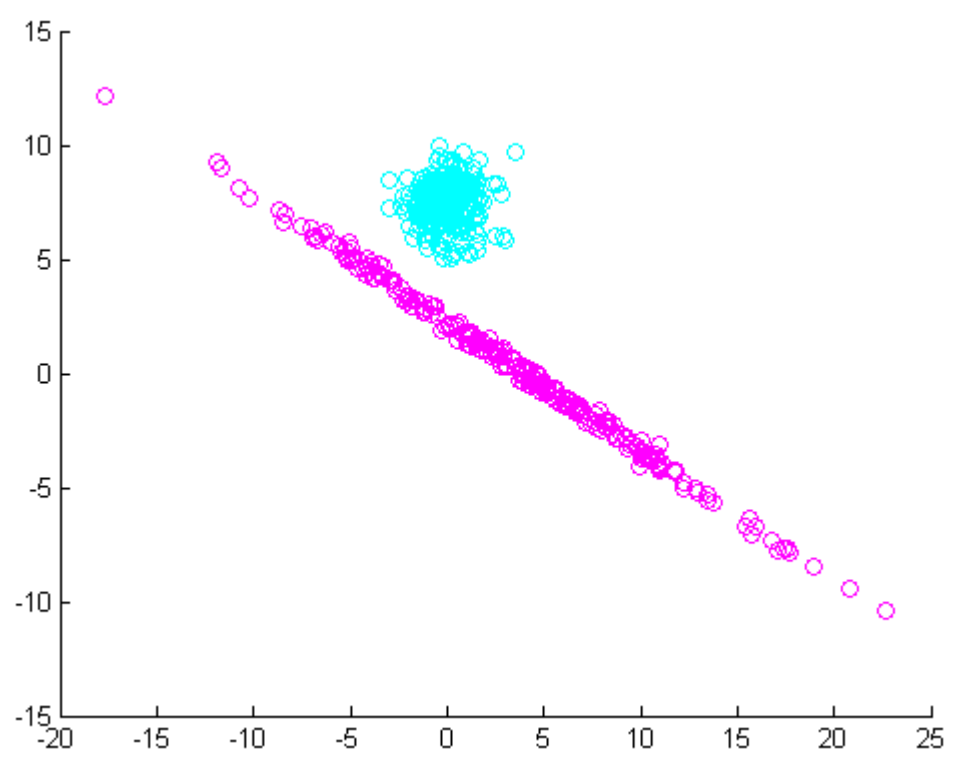


Improving the problem: Local σ

Results



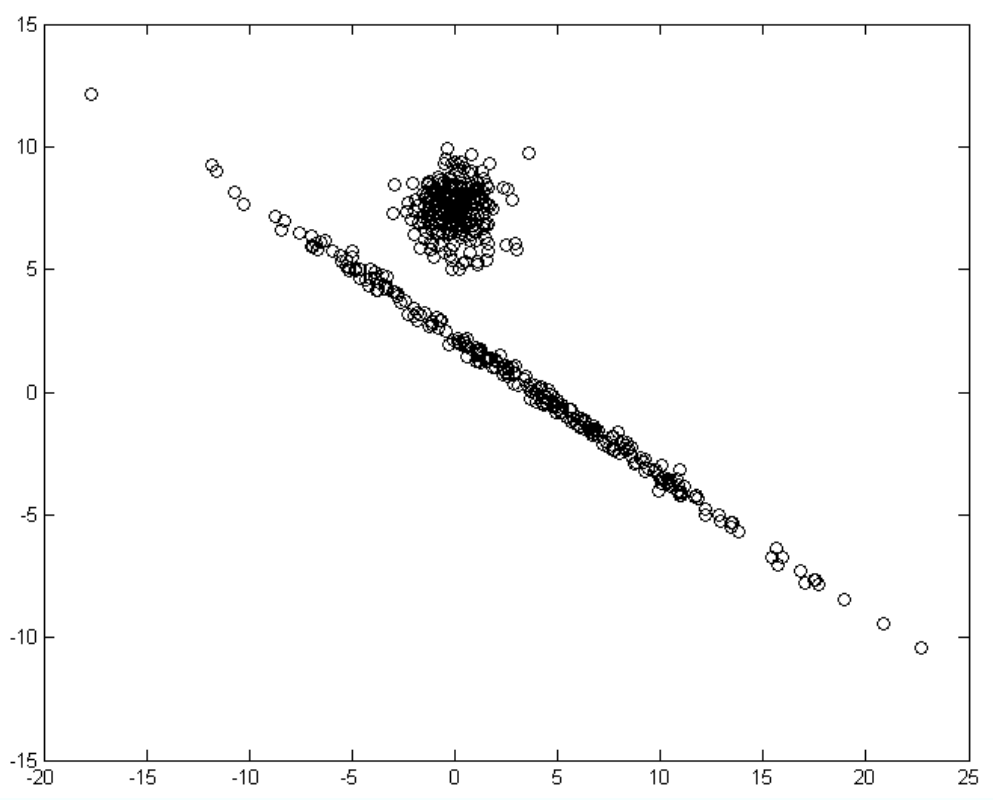




Searching for the local σ

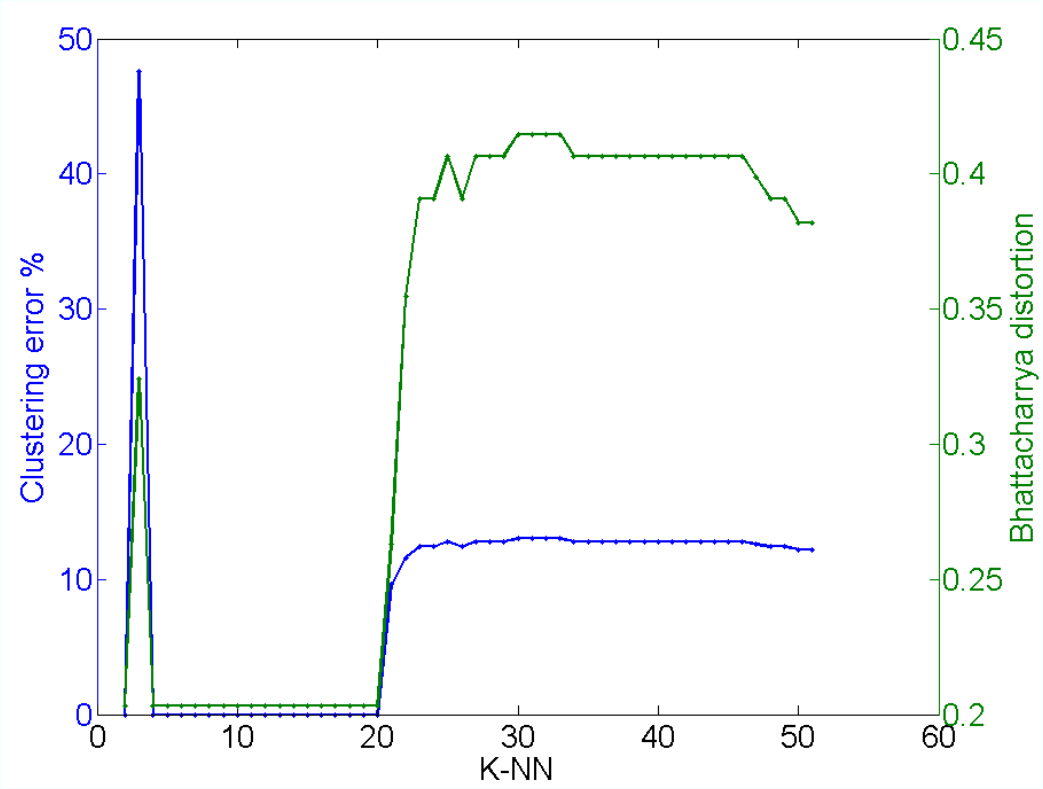
A harder problem: Two proximal clusters of different shapes

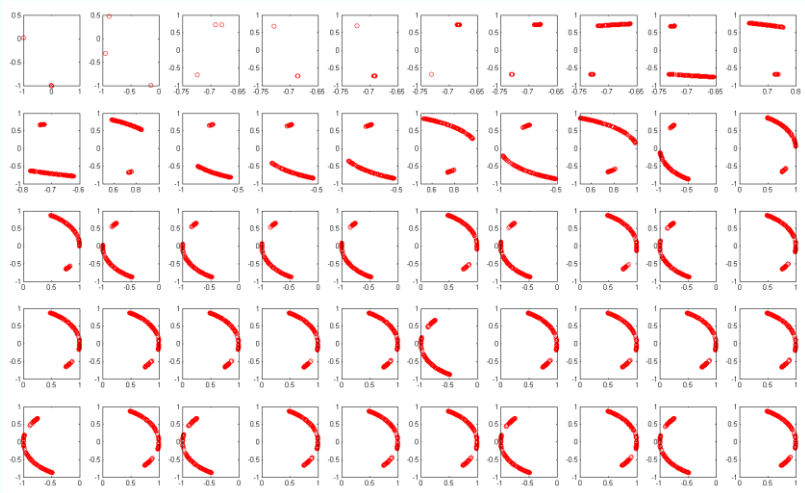
- We take 50 equidistant K-NN samples from [2,…,52]
- At each sample, we calculate the affinity matrix with the local σ and perform spectral clustering
- But now after each clustering we evaluate a quality measure
- We then evaluate the ground truth clustering error



Improving the problem: Local σ

Results





- In general the use of an appropriate local σ can improve the problem by finding clustering solutions not available to a global σ and with a wider basin

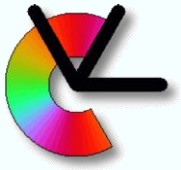
Improving the problem: The Q matrix

The basic algorithms described in the last lecture can be enhanced in various ways
In the following:

- Assume that self-affinities are = 1
- Diagonal entries in **A** are = 1

An observation

- An observation:
  - adding noise to **A** causes noise to appear both in the eigenvalues of **A** and in the corresponding eigenvectors
- This is the case also for **D** and **L**
- For the eigenvalues:
  - Eigenvalues that should be = 0 become $\neq 0$
  - Such eigenvalues can mix with those that should be > 0 but are relatively small
- For the eigenvectors:
  - The space spanned by the *k* largest eigenvectors is not exactly the space spanned by the indicator vectors



Rank *k*-approximation

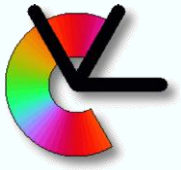
- We know that **A** should have rank *k*
- In Frobenius norm, the best approximation of **A** with a rank *k* matrix is given by

$$\mathbf{A}_{\text{rank } k} = \mathbf{U}\,\Sigma\,\mathbf{U}^{\mathsf{T}}$$

  - **U** = $n \times k$ matrix of normalized eigenvectors of **A**
  - Σ = $k \times k$ diagonal of the *k* largest eigenvalues

Rank *k*-approximation

- $\mathbf{A}_{\text{rank } k}$ is the “closest” affinity matrix relative to **A** that has the correct rank
- $\mathbf{A}_{\text{rank } k}$ is not a “true” affinity since entries may be negative
- Set negative values to zero
  - This operation changes eigenvalues in $\mathbf{A}_{\text{rank } k}$ and the rank k-constraint is no longer valid
  - In practice, only small adjustments to eigenvalues occur if the negative values are small



The matrix **Q**

- As a representation of $\mathbf{A}_{\text{rank } k}$ we can even use the matrix

$$\mathbf{Q} = \mathbf{U}\,\mathbf{U}^{T}$$

- This is a projection operator onto the space spanned by the *k* largest eigenvectors

The matrix **Q**

- **Q** is the same as $\mathbf{A}_{\text{rank } k}$ but without the Σ
- In the ideal case: Σ contains n_1, n_2, ...
- In the ideal case, removing Σ causes elements in **Q** to be either
  - $1/n_i$ in the blocks in the diagonal
  - 0 outside the blocks

The matrix **Q**

- From a graph with affinity **A**, and *k* clusters

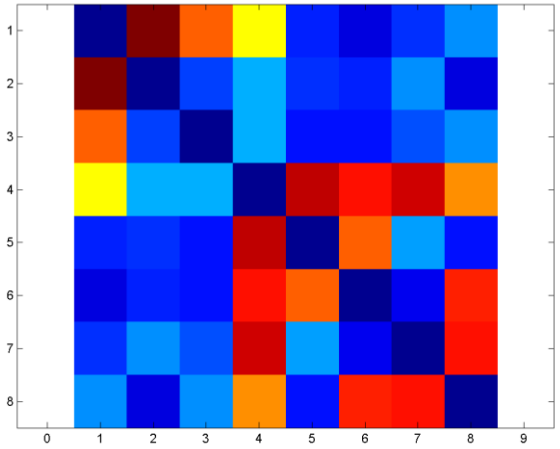
- Form $\mathbf{Q} = \mathbf{U}\,\mathbf{U}^T$
- Suppress negative affinities in **Q** to zero
- **Q** can be used directly for segmentation by thresholding

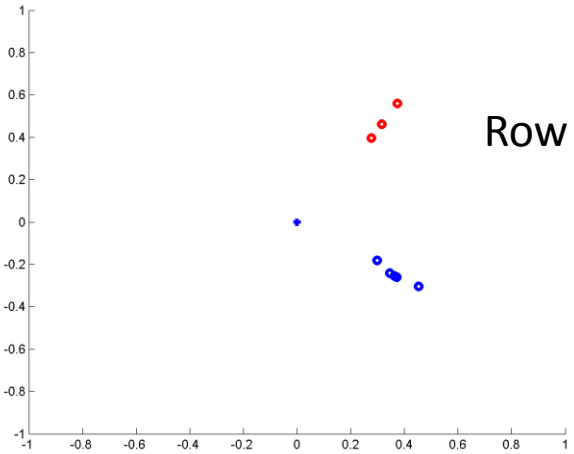
- Threshold should be chosen carefully so it gives the desired number of clusters

- Reduces the “eigen-noise”

A numerical example

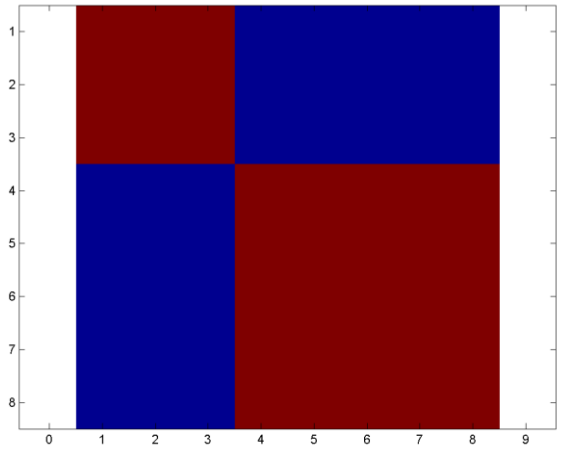
A

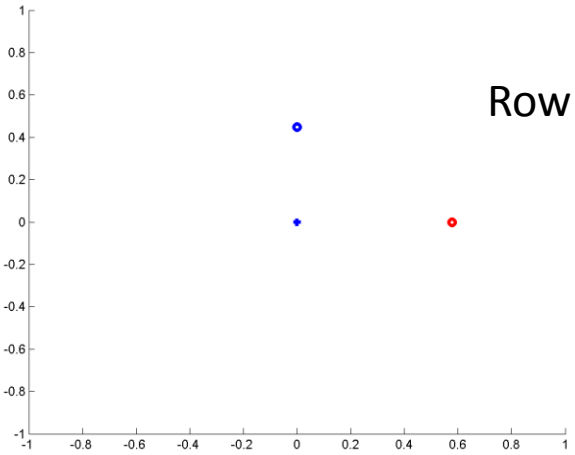




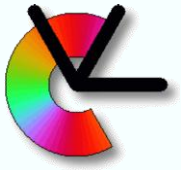
Row space of $\mathbf{U_A}$

Q





Row space of $\mathbf{U_Q}$

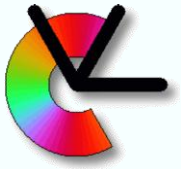


Improving the problem

- Limitations of a single affinity matrix
- We need additional info (if available)
  - Multiple-affinity matrices

- This only makes sense if the solution can be given by either affinity matrix in the perfect, noise-free case
  - Intersection of solution space
  - Correlation of information in the affinity matrices

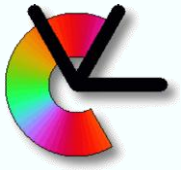
- Use basic fusion approaches
- Use more advanced methods e.g. co-regularisation
- Use information theoretic criteria for fusion



Improving the problem: Multi-view Spectral Clustering

In some cases:

- We can form affinities from one and the same data set in multiple ways
  - Different features
  - Different distance functions
  - Different kernel functions
  - ...
- We get affinity matrices $\mathbf{A}_1$, $\mathbf{A}_2$, ...
- All of them approximately describing the same graph segmentation problem



Improving the problem: Multi-view Spectral Clustering

- We want to do clustering based on all of $\mathbf{A}_1$, $\mathbf{A}_2$, ...
- Referred to as ***multi-view spectral clustering***

- In the following presentation:
  - Assume two affinities $\mathbf{A}_1$, $\mathbf{A}_2$
  - Straight-forward to generalize to more affinities
- In general, we assume that affinities in $\mathbf{A}_1$ and $\mathbf{A}_2$ are in the [0, 1] range (i.e. normalised)

Improving the problem: Multi-view Spectral Clustering

- One category of approaches implies *fusing* $\mathbf{A}_1$ and $\mathbf{A}_2$ into a *joint affinity*

$$\mathbf{A}_{\text{joint}} = f(\mathbf{A}_1, \mathbf{A}_2)$$

and do spectral clustering on $\mathbf{A}_{\text{joint}}$

- How do we choose f ?
  - Several options exist in the literature

Improving the problem: Multi-view Spectral Clustering

Hadamard product

- $\mathbf{A}_{\text{joint}}$ = Hadamard product of $\mathbf{A}_1$ and $\mathbf{A}_2$
  - $a_{\text{joint } ij} = a_{1ij} \cdot a_{2ij}$
- The Hadamard product implies a sort of AND operation on the joint graph
  - Only edges that have large weights in both $\mathbf{A}_1$ and $\mathbf{A}_2$ will be large also in the joint graph
  - Edges that have weights close to zero in one or both of the $\mathbf{A}_1$ and $\mathbf{A}_2$ graphs will have close to zero weight also in the joint graph

Improving the problem: Multi-view Spectral Clustering

Mean

- $\mathbf{A}_{joint}$ = mean of $\mathbf{A}_1$ and $\mathbf{A}_2$
  - $a_{joint\ ij} = (a_{1ij} + a_{2ij}) / 2$
- The mean operation implies a sort of OR operation on the joint graph
  - Edges that have small weights in both $\mathbf{A}_1$ and $\mathbf{A}_2$ will be small also in the joint graph
  - Edges that have large weights one or both of the $\mathbf{A}_1$ and $\mathbf{A}_2$ graphs will have large weight also in the joint graph

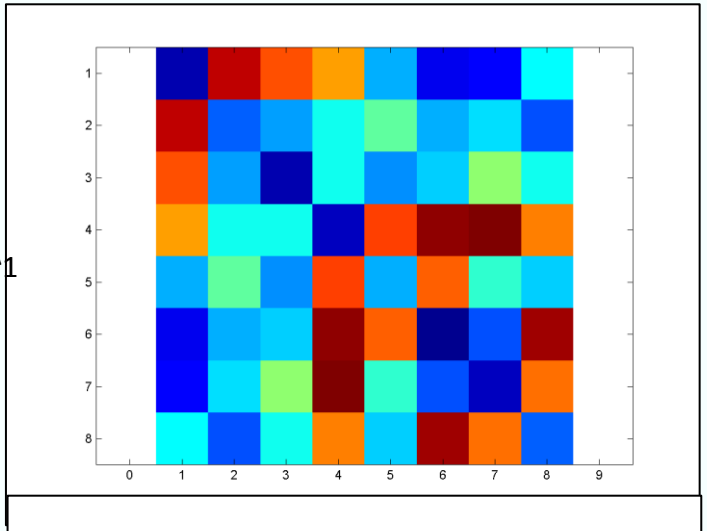
Improving the problem: Multi-view Spectral Clustering

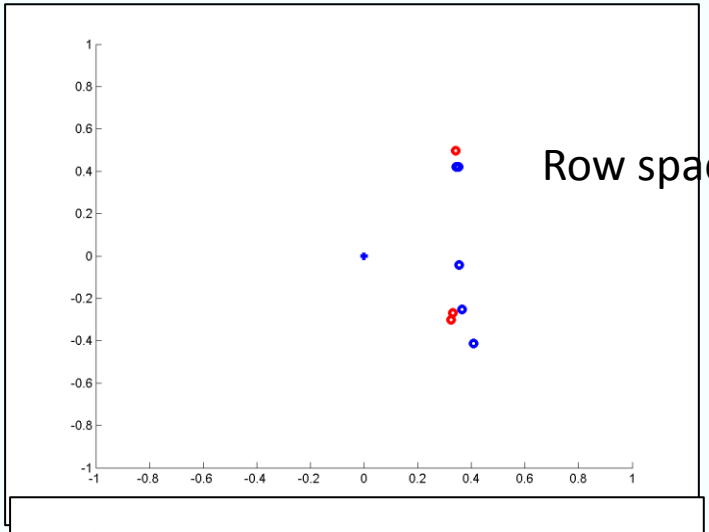
Min & Max

- The AND and OR type of operations can also be implemented as
  - $a_{\text{joint } ij} = \max(a_{1ij}, a_{2ij})$ (OR)
  - $a_{\text{joint } ij} = \min(a_{1ij}, a_{2ij})$ (AND)
- All four approaches can be implemented in a very simple way and can often solve the problem, but
  - it depends heavily on the data which approach works best
  - Not obvious which of the AND or OR approach is the better

A synthetic example

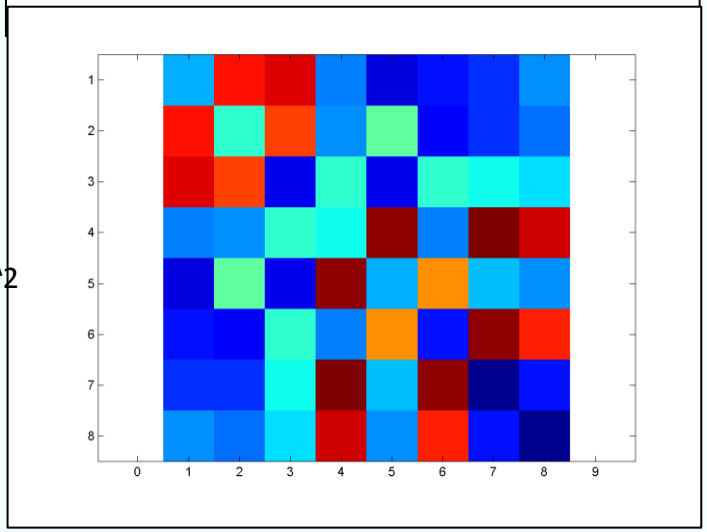
$\mathbf{A}_1$

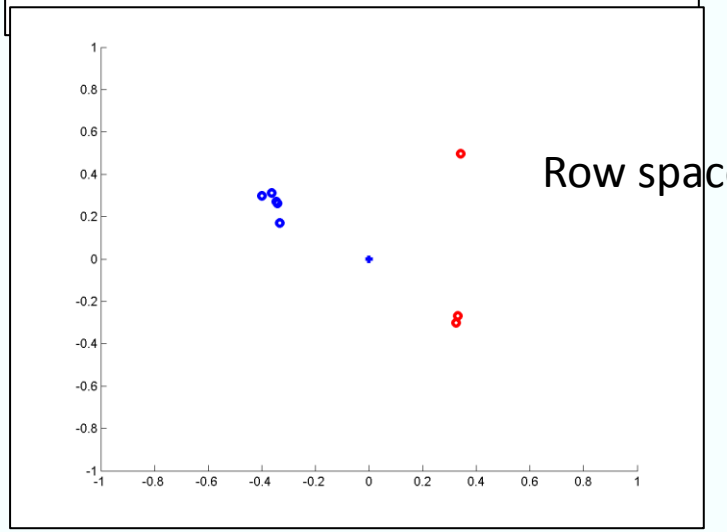




Row space of $\mathbf{U}_1$

$\mathbf{A}_2$

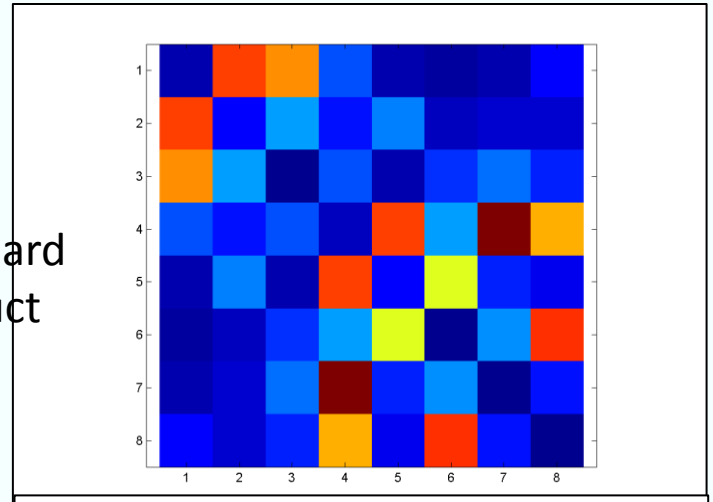


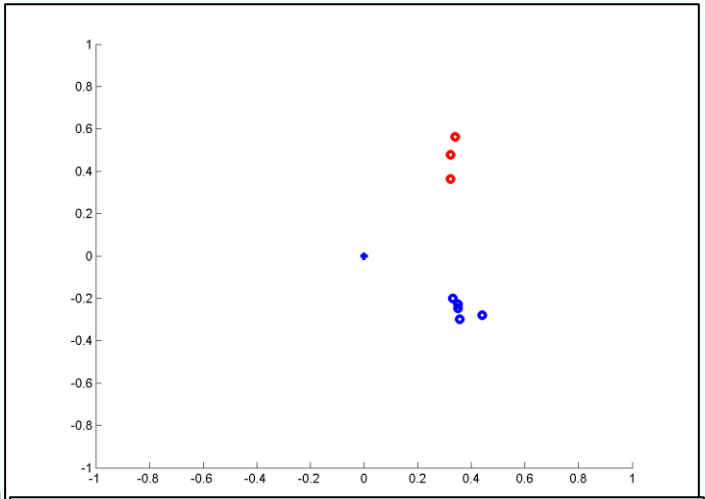


Row space of $\mathbf{U}_2$

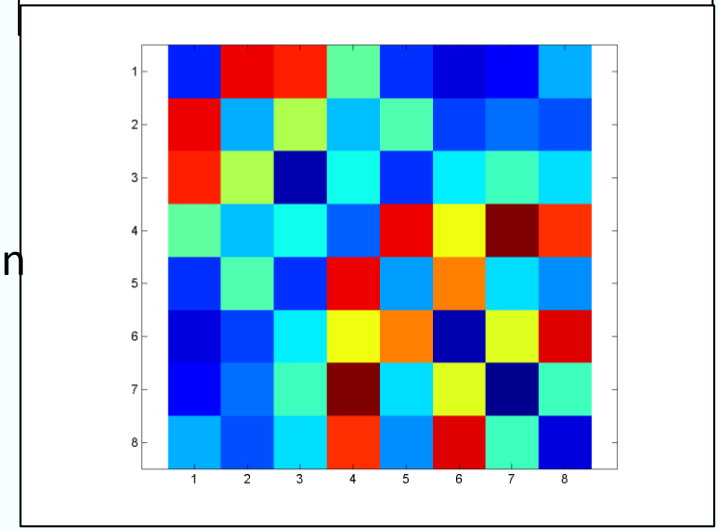
A synthetic example

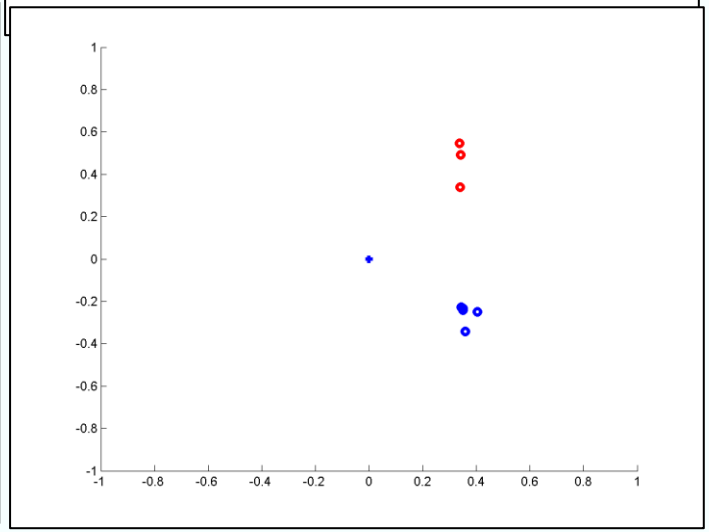
Hadamard product





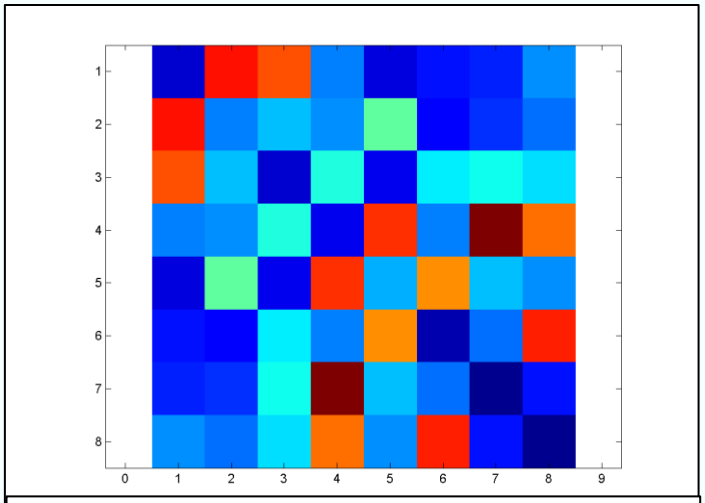
Mean

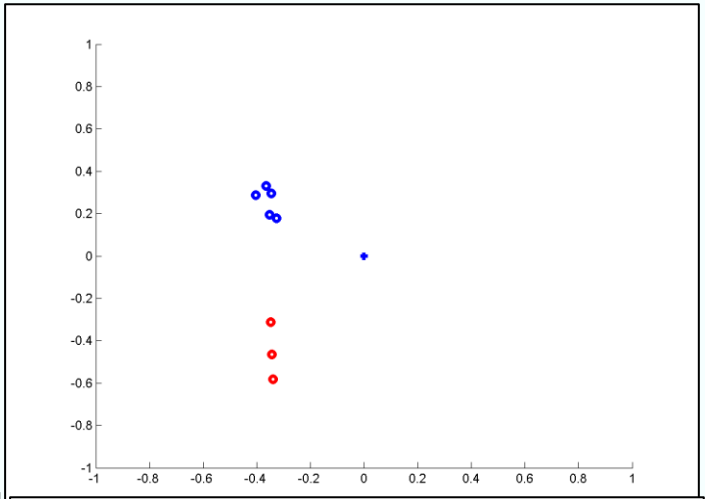




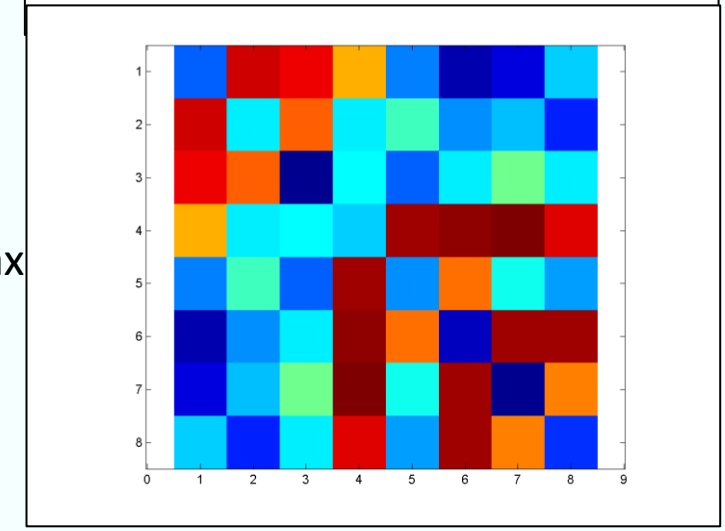
A synthetic example

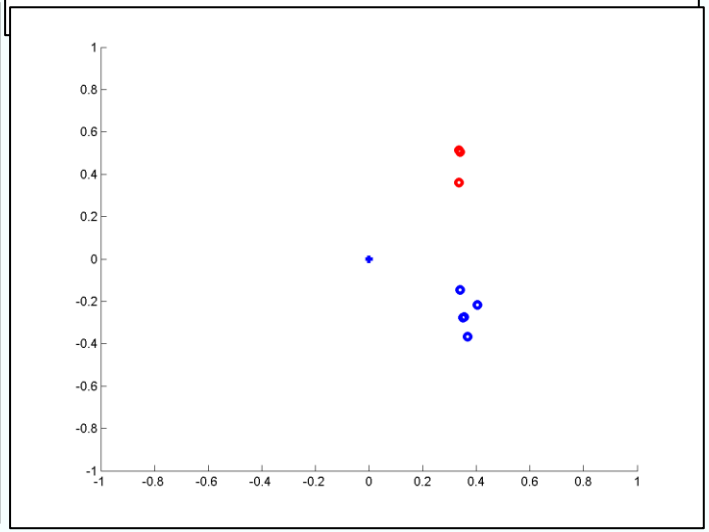
Min



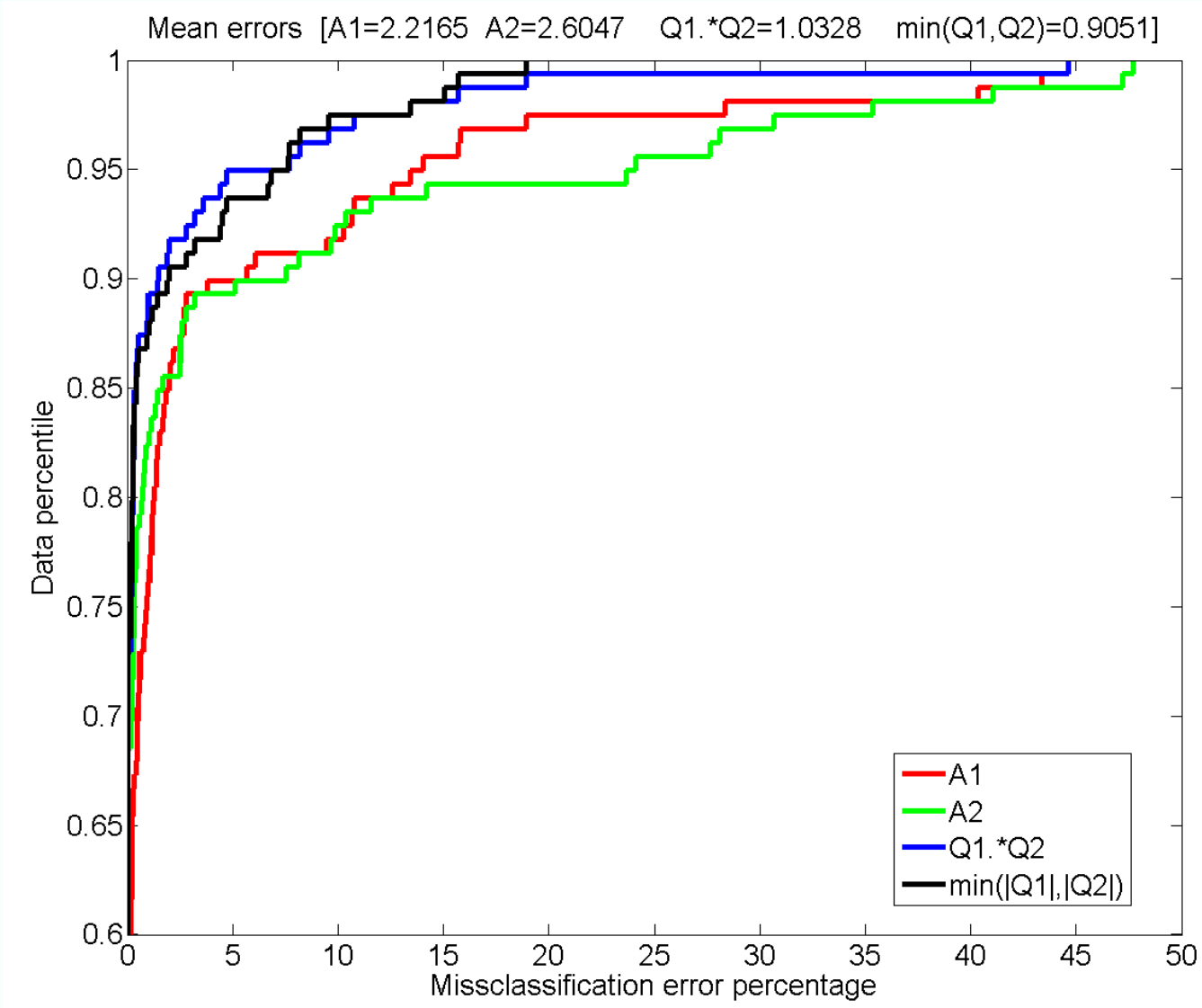


Max





A real example: Motion segmentation



Improving the problem: Co-regularisation

- A slightly more complicated approach is described by Kumar & Daumé, ICML 2011
- Basic idea
  1. Form the normalized Laplacian of each affinity
  2. Compute the eigensystem of both
  3. Use the eigensystem of one to modify the affinities of the other
  4. Iterate from 1.
- Referred to as co-regularization of $\mathbf{A}_1$ and $\mathbf{A}_2$

Improving the problem: Co-regularisation

More in detail:

- Column *i* of $\mathbf{A}_1$ or $\mathbf{A}_2$ ideally should be an indicator vector of the cluster that point i belongs to
- Due to imperfect data, not every column has this character
- For each of $\mathbf{A}_1$ and $\mathbf{A}_2$ we get a separate estimate the space spanned by the indicator vectors
- Project the columns of $\mathbf{A}_1$ onto the space estimated from $\mathbf{A}_2$, and vice versa
- Symmetrizise since the projection destroys the symmetry
  - $\text{sym}(\mathbf{X}) = (\mathbf{X} + \mathbf{X}^T) / 2$

Improving the problem: Co-regularisation

- Kumar and Daumé use a slightly different notation for their matrices (than Luxburg):
  - $\mathbf{A}_1$ and $\mathbf{A}_2$ are the affinities
  - $\mathbf{D}_1$ and $\mathbf{D}_2$ are the corresponding degree matrices
  - $\mathbf{L}_1 = \mathbf{D}_1^{-1/2} \mathbf{A}_1 \mathbf{D}_1^{-1/2}$ and $\mathbf{L}_2 = \mathbf{D}_2^{-1/2} \mathbf{A}_2 \mathbf{D}_2^{-1/2}$ are the normalized Laplacians (not the same as before!)

  - In the ideal case: the eigenvectors of the ***largest*** eigenvalues in $\mathbf{L}_1$ and $\mathbf{L}_2$ span the space of the cluster indicator vectors

Kumar-Daumé algorithm:

Initialize for k clusters

- $\mathbf{L}_p = \mathbf{D}_p^{-1/2}\mathbf{A}_p\mathbf{D}_p^{-1/2}$, $p = 1, 2$
- $\mathbf{U}_{p,0}$ = normalized eigenvectors corresponding to the k largest eigenvalues of $\mathbf{L}_p$

For $i = 1$ to I

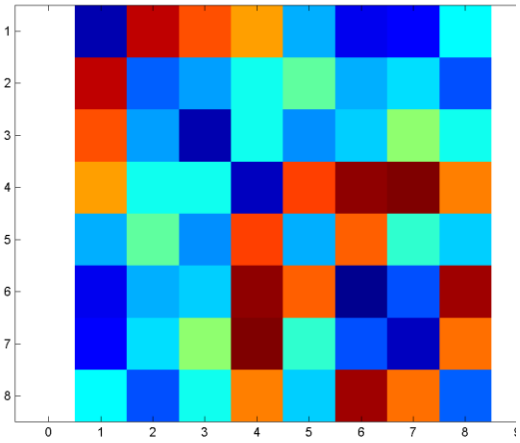
1. $\mathbf{Q}_{p,i\text{-}1} = \mathbf{U}_{p,i\text{-}1}\mathbf{U}_{p,i\text{-}1}^{T}$, $p = 1, 2$ (a projection operator!)
2. $\mathbf{S}_1 = \text{sym}(\mathbf{Q}_{2,i\text{-}1}\,\mathbf{A}_1)$
3. $\mathbf{S}_2 = \text{sym}(\mathbf{Q}_{1,i\text{-}1}\mathbf{A}_2)$
4. Use $\mathbf{S}_1$ and $\mathbf{S}_2$ as the new affinities: compute the Laplacians $\mathbf{L}_p$ and their eigenvectors in $\mathbf{U}_p$
5. End

Kumar-Daumé algorithm:

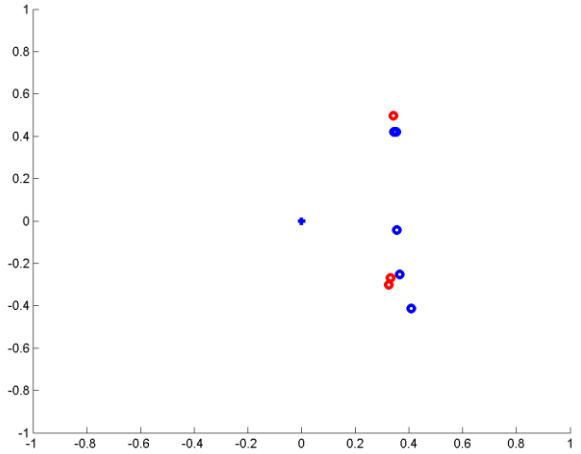
- $\mathbf{U}_{1,l}$ and $\mathbf{U}_{2,l}$ are now *co-regularized*
- Row normalize $\mathbf{U}_{1,l}$ and $\mathbf{U}_{2,l}$
- Form matrix $\mathbf{V}$ from $\mathbf{U}_{1,l}$ and $\mathbf{U}_{2,l}$, either by
  - $\mathbf{V} = \mathbf{U}_{1,l}$ or $\mathbf{U}_{2,l}$ if there is a prior on which graph is most informative
  - $\mathbf{V} = [\mathbf{U}_{1,l} \ \ \mathbf{U}_{2,l}]$ = concatenation of rows in $\mathbf{U}_{1,l}$ and $\mathbf{U}_{2,l}$
- Do *k*-means clustering on the row space of $\mathbf{V}$

A numerical example

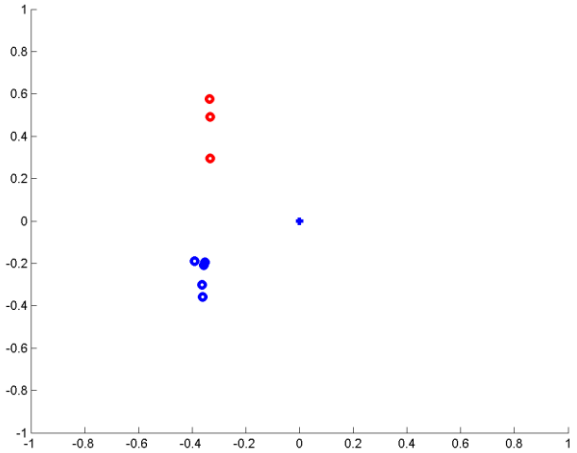
$\mathbf{A}_1$

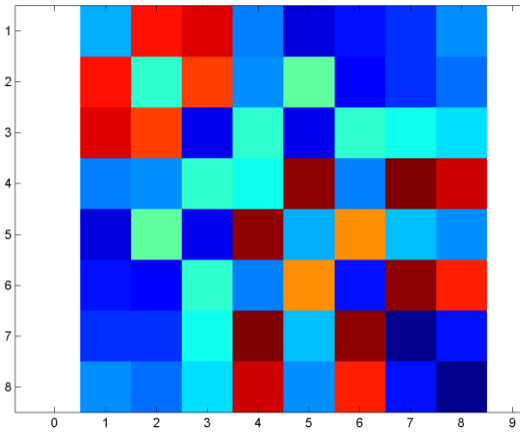


Row space of $\mathbf{U}_1$

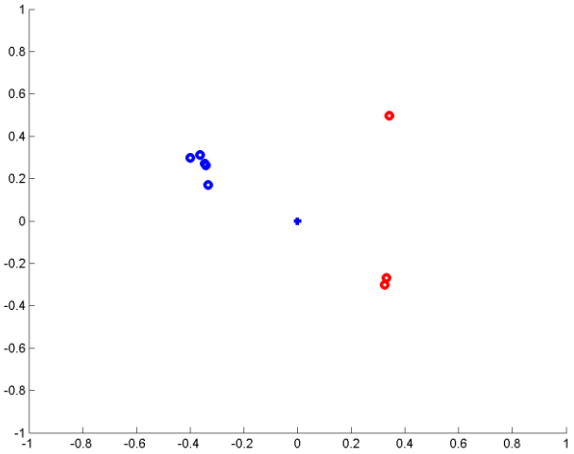


After 5 KD-iterations

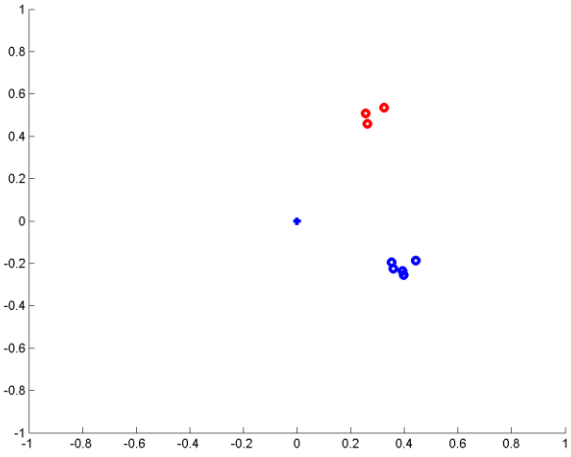




$\mathbf{A}_2$



Row space of $\mathbf{U}_2$



After 5 KD-iterations

The number of clusters (1)

- A general problem of clustering – There as many clusters as you want to find
- No real solution. Some heuristics. Problem dependent

Spectral clustering specific: The eigen-gap

- Find the number of clusters by analyzing the eigenvalues of the Laplacian matrix

- The number of eigenvalues of magnitude 0 is equal to the number of clusters k.
  - This implies one could estimate k simply by counting the number of eigenvalues equal or close to 0.
  - This criterion works when the cluster are well separated

- *Eigen-gap*: the difference between two consecutive eigenvalues

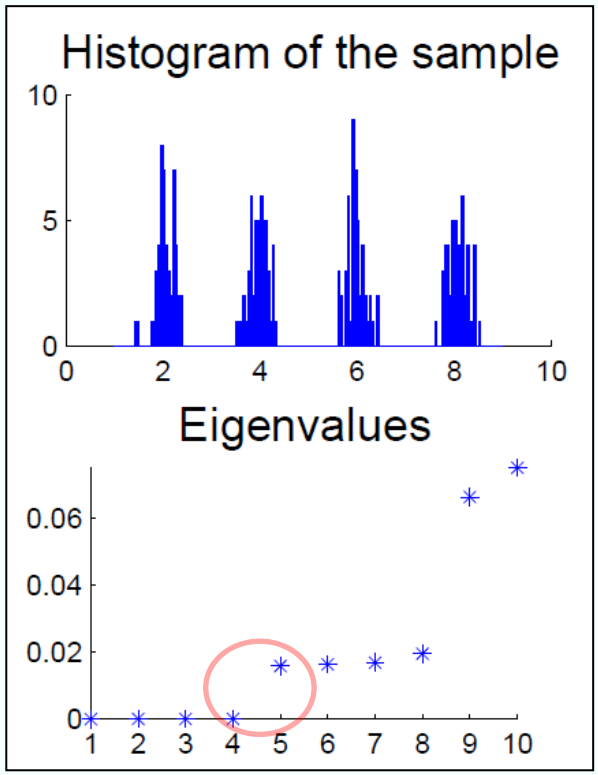
$$\Delta_k = |\lambda_k - \lambda_{k-1}|$$

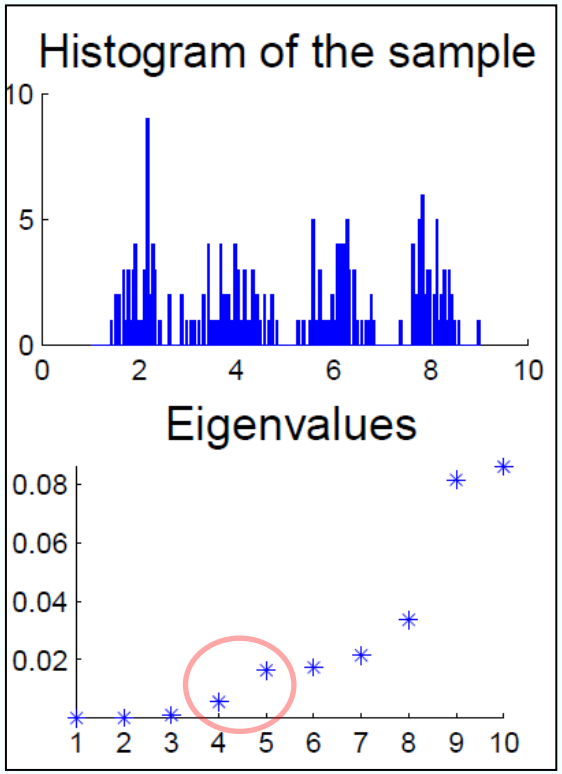
- In general search for a significant increase in the eigen-gap of the eigenvalues arranged in increasing order

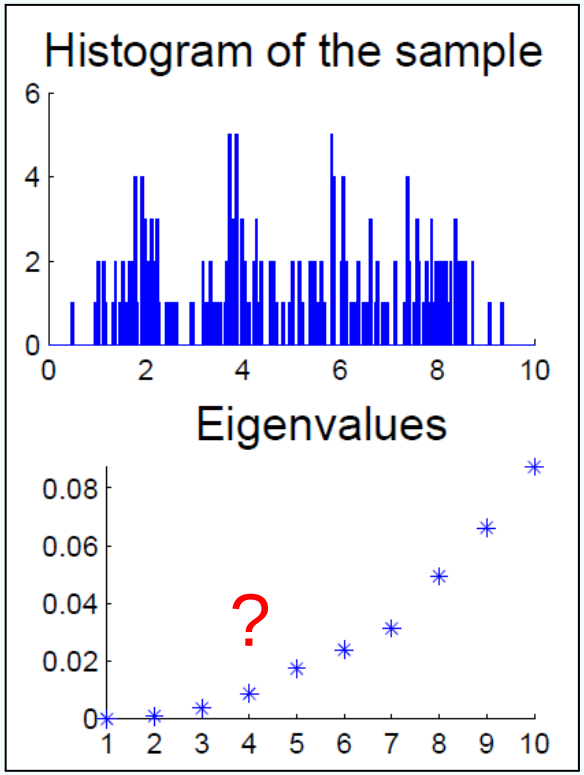
The number of clusters (1)

- The eigen-gap is not very robust to noise
- It also needs a data dependent threshold

Noise →





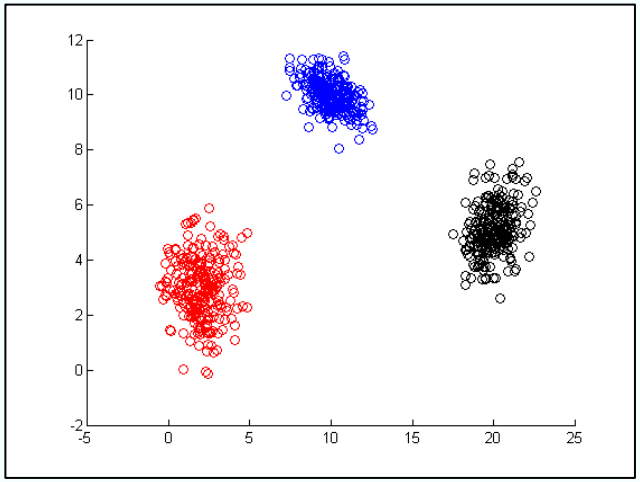


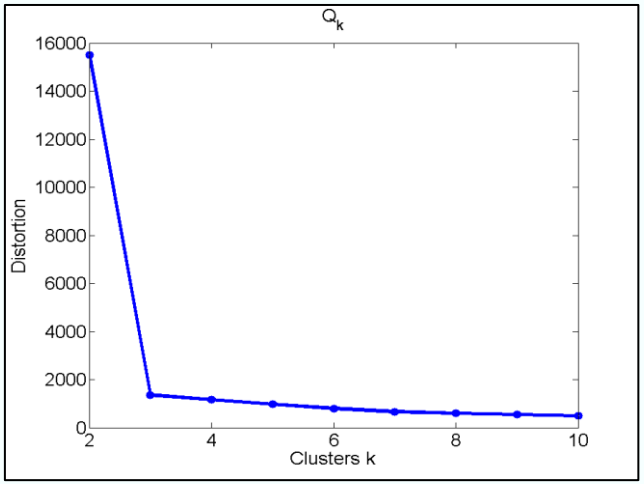
The number of clusters (2)

- Generic criteria – Do spectral clustering and check afterwards

Gap-statistics

- Use some quality function $\boldsymbol{Q}$ (see Lecture 1)
- We cannot use $\boldsymbol{Q_k}$ to determine number of clusters k directly because $\boldsymbol{Q_k}$ scales with the number of clusters - Needs to be normalised





$\boldsymbol{Q_k}$

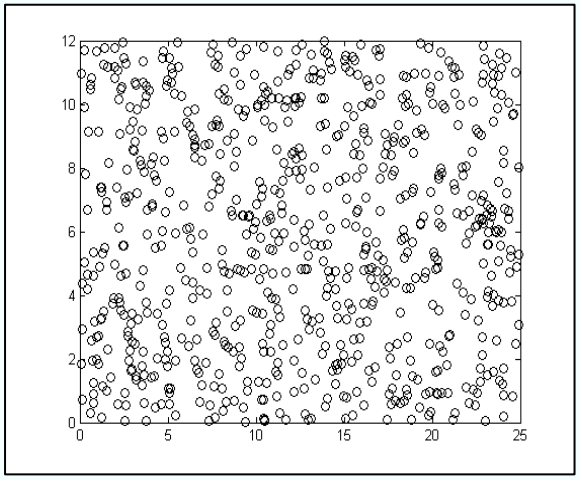
- **Remember**: Clustering can be done by Spectral Clustering but in this case Quality measure MUST be calculated in the data domain.

The number of clusters (2)

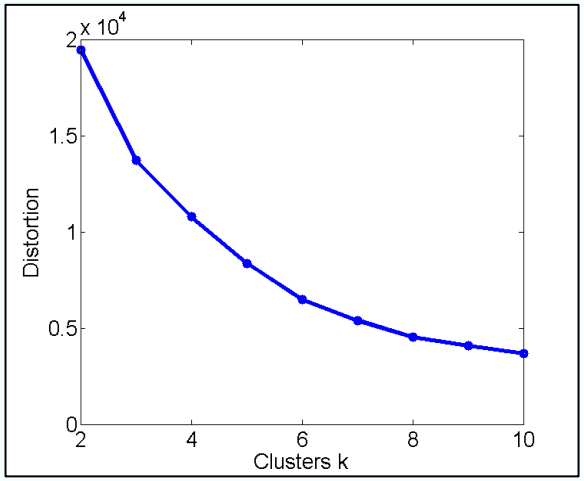
- Generate uniform data from same domain
  - Might be a problem for high dimensional datasets or additional samples could be expensive

- Calculate the quality function $\boldsymbol{Q_{U,k}}$ for the uniform data
- Normalise to obtain the $\boldsymbol{Q_k/Q_{U,k}}$

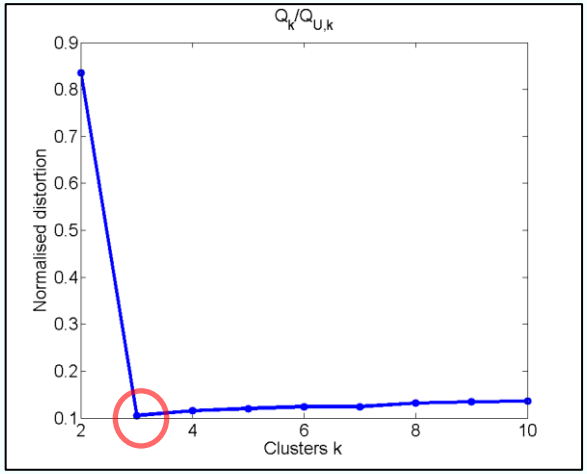
- **The minimum of $\boldsymbol{Q_k/Q_{U,k}}$ is the number of clusters**



Random data over the same dimension and domain



$\boldsymbol{Q_{U,k}}$



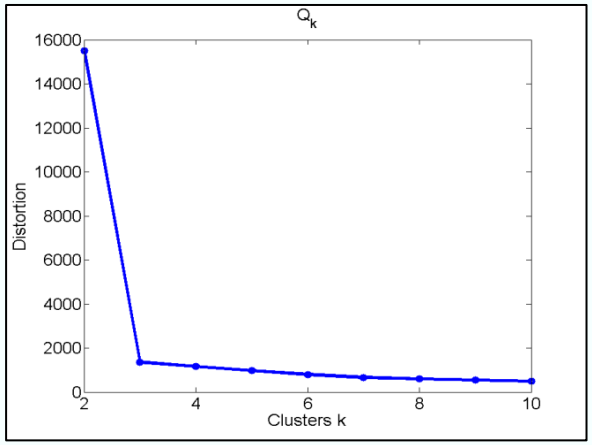
$\boldsymbol{Q_k/Q_{U,k}}$

The number of clusters (3)

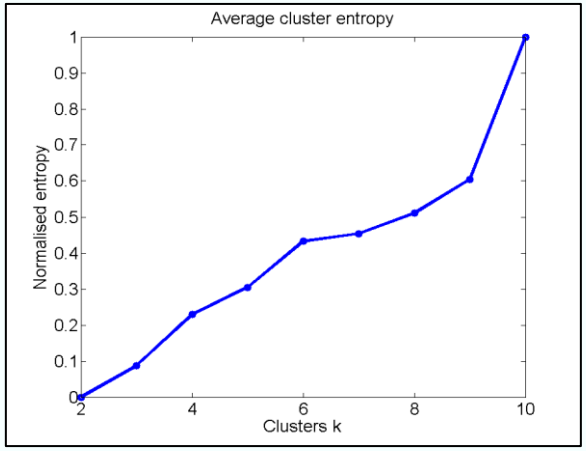
Information theoretic

- Clustering = data compression
- We minimise a quality function (e.g. distortion)
- The quality function depends on cluster distribution and is minimised for $k = \infty$

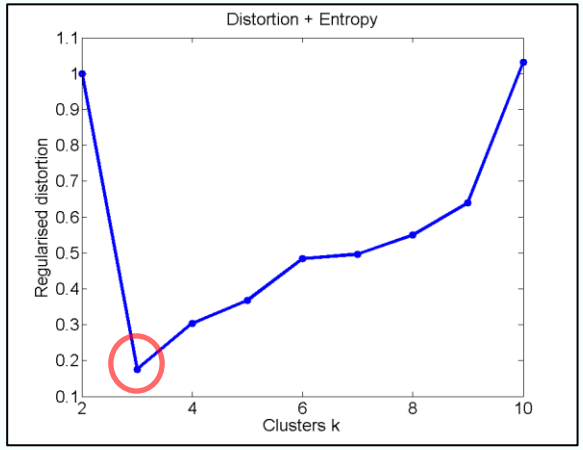
- We need a “regulariser” to avoid overfitting
- Minimise instead quality + regulariser (e.g. MDL)



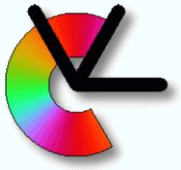
Q_k



$E = -\sum (p \log_2(p))$



$Q_k + E$



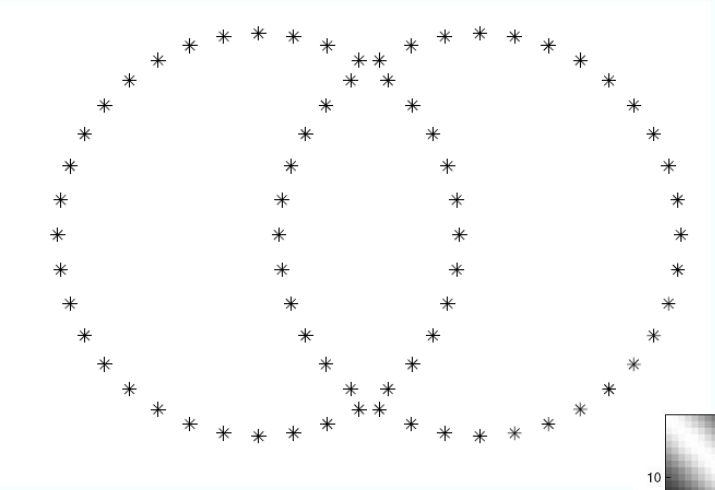
Multi-way affinities

- In some applications (e.g. shape segmentation) it does not make sense to use pairwise-affinities
  - Require a residual

- For example
  - Points on a circle ($\geq$ 4 points)
  - Motion segmentation ($\geq$ 4, 5 or 6 points)
  - ...

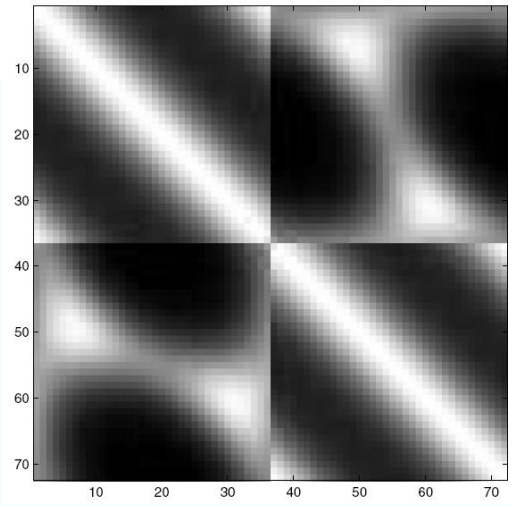
Example, points on circles

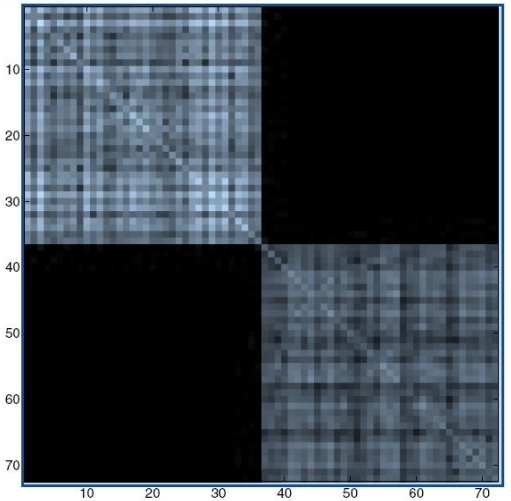
From Govindu, CVPR 2005



Affinities based on circle fitting

Affinities based on Euclidean distances



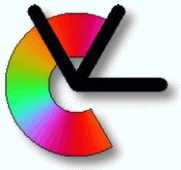


Multi-way affinities

- In the case that we need p points to determine an affinity, we form a p-dimensional array $\mathcal{P}$

$$\mathcal{P}(i_1, i_2, \ldots, i_p) =$$
$$= \text{affinity between points } (i_1, i_2, \ldots, i_p)$$

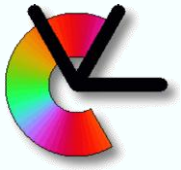
- $\mathcal{P}$ is referred to as a p-way array or tensor
- We assume that $\mathcal{P}$ is *super-symmetric*



Multi-way affinities

Govindu, CVPR 2005

- Interprets the elements of $\mathcal{P}$ as probabilities that points (i_1, i_2, ..., i_p) belong together
- Form an $n \times n^{p-1}$ matrix **P** by "flattening" $\mathcal{P}$ along indices 2, 3, ..., p
- Due to the super-symmetry it does not matter which indices that are use for the flattening
- Form pair-wise affinity as $\mathbf{A} = \mathbf{P}\,\mathbf{P}^{\mathsf{T}}$



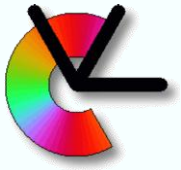
Multi-way affinities

- We see that

$$a_{ij} = \sum_{\mathbf{c} \in C} \mathcal{P}(i, \mathbf{c}) \mathcal{P}(j, \mathbf{c})$$

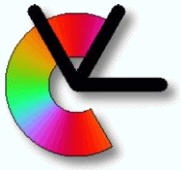
$$C = \text{all combinations of } p-1 \text{ indices in the range } [1, \text{n}]$$

- Basic idea
  - If points (*i,j*) belong to the same cluster: there are many combinations of additional points **c** that make both $\mathcal{P}(i, \mathbf{c})$ and $\mathcal{P}(j, \mathbf{c})$ large $\Rightarrow a_{ij} > 0$
  - If points (*i,j*) belong to different clusters: there are no or few combinations of additional points **c** that make both $\mathcal{P}(i, \mathbf{c})$ and $\mathcal{P}(j, \mathbf{c})$ large $\Rightarrow a_{ij} \approx 0$



Multi-way affinities

- Computing first **P**, and then **A** = **P** $\mathbf{P}^T$ becomes very expensive for large *n* and *p*
- There are n^{p-1} elements in *C*
- Simplification:
  - Choose *C'* as a subset of *C*
  - *C'* must be of reasonable size: not too small/large
  - With *q* as the number of combinations in *C'*
  - Corresponds to a subsampling of the columns in **P**
- For example:
  - Choose *C'* as *q* random element from *C*



Multi-way affinities

$$a_{ij} \approx \sum_{\mathbf{c} \in C'} \mathcal{P}(i, \mathbf{c}) \mathcal{P}(j, \mathbf{c})$$

$$\mathcal{O}(n^2 \cdot q) \quad \text{instead of} \quad \mathcal{O}(n^{(p+1)})$$

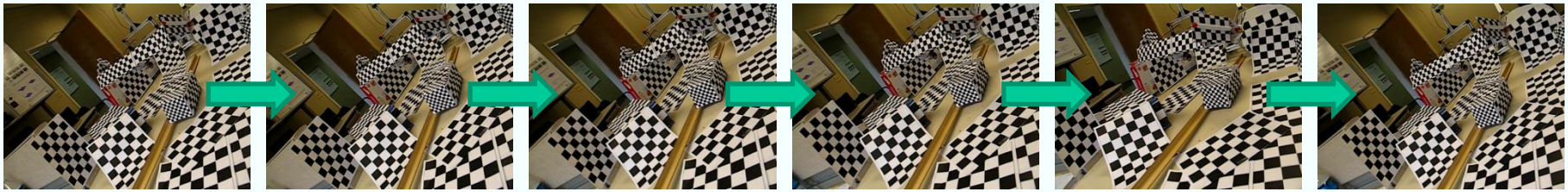
Improvement of multi-way affinities clustering

Chen & Lerman, IJCV 2009

- Choosing C'_{ij} randomly may be an ineffective approach
- Efficiency increases slowly with increasing q
- They propose an iterative approach:
  1. Do an initial clustering based on random C'_{ij}
     - Produces tentative clusters
  2. Redo the clustering: select the elements of C'_{ij} randomly but only within the tentative clusters
     - Increases the chance of getting large and correct affinities
  3. Iterate from 2 until happy 

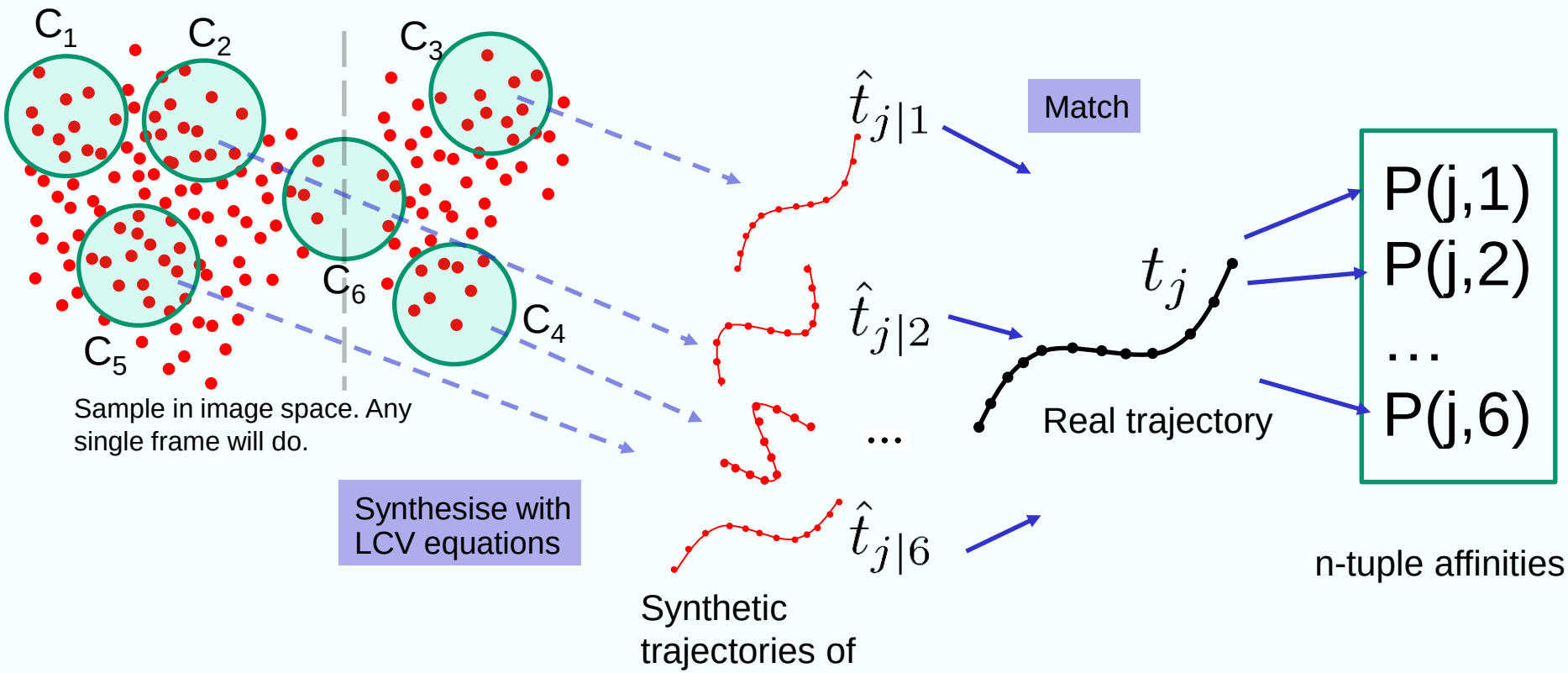
Multi-way affinities: A real example

- Revisit the motion segmentation problem
- How do we deal with 3d motions? Real example Hopkins155 dataset



- Define an affinity between multiple points that implies 3d motion consistency
- Much more robust than simple pairwise 2d point affinities
- Linear combination of views for motion segmentation [Zografos and Nordberg 2011]:
  - Use 7+1 points in an image to define a 3d motion affinity (8-tuple)
- We use spatial K-means to get good quality n-tuple columns

Multi-way affinities: A real example

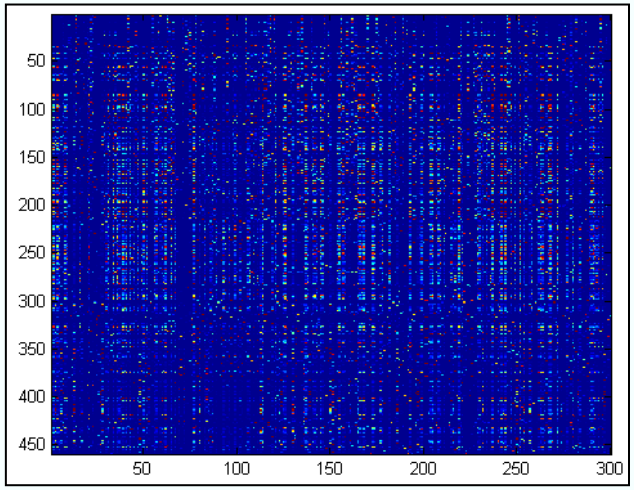


Multi-way affinities: A real example

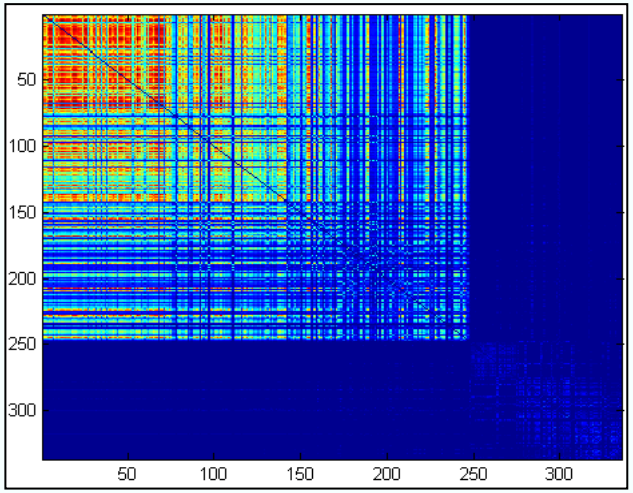
- The n-tuple affinity between the point p_j and the n-points c is defined as:

$$P(j,c) = K(\left\| t_j - \hat{t}_{j|c} \right\|_H / F)$$

- K is a kernel function $K(x,\sigma) = \left(x^2 + \sigma^2\right)^{-1/2}$
- The affinity matrix is therefore A $\approx$ PP^T



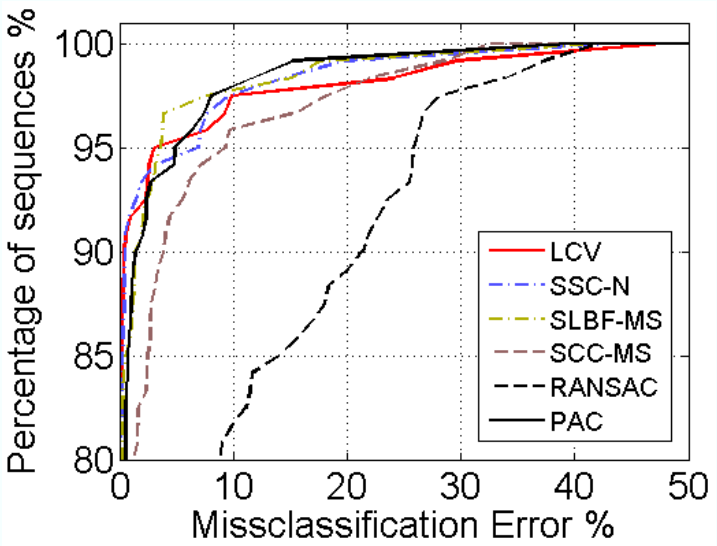
P matrix (300 columns)

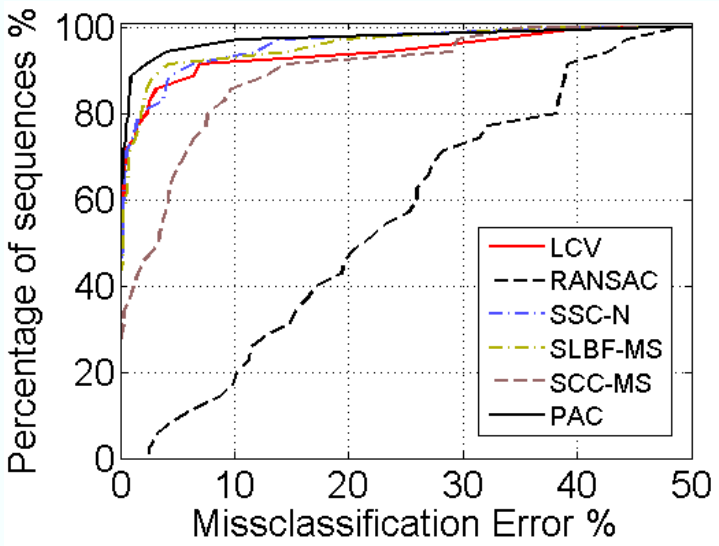


Affinity matrix A

Multi-way affinities: A real example

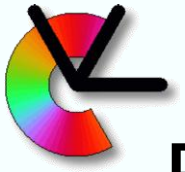
- The results. The different methods differ on the way they define their affinities. All use multi-way affinities





Method	RANSAC	SCC-MS	SLBF-MS	SSC-N	PAC	LCV
Average time (sec)	0.387	1.264	10.83	165	952.25	0.93
Total time (sec)	60	196	1680	25620	147600	145
Average error (%)	9.48	2.70	1.35	1.36	1.24	1.86

The average and total runtime on the full Hokpins155 dataset



Extensions: Large affinity matrices The Nyström method

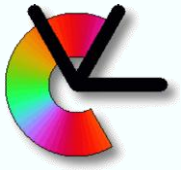
- In some cases *n* may be very large
  - Example: *n* = number of pixels in the image
- Forming $n \times n$ matrix **A** and then doing spectral clustering on **A** becomes infeasible
- Use the *Nyström method* as a means for a numerical approximation of the clustering problem
  - Proposed by Fowlkes, *et al*, PAMI (2004)

The Nyström method

Basic idea of the Nyström method (1924)

- Given an $n \times n$ affinity matrix **A**
- Subsample it to form $n' \times n'$ matrix **A**' How?
- Then, also **A**' is an affinity matrix, but for a subset of the original points
- Spectral clustering on **A**' will reveal tentative clusters in this subset
  - In this case: we need only **U**' holding the *k* largest eigenvectors of **A**'
- Extend these to include also the original points

How?



The Nyström method

Sampling:

- In some applications in makes sense to consider pair-wise affinities between points that are “close”
  - For example: pixels that are close in the image
- This, however, discourages the use of long-range affinities that sometimes are present in images
  - For example: motion segmentation
- Better option: do a sub-sampling of the points in a regular or pseudo-random way
  - For example: consider only every *r*-th point: *n*’ = *n* / *r*
  - **A**’ is (*n* / *r*) $\times$ (*n* / *r*)

$$\mathbf{A} \text{ is a permutation } \Pi \text{ of } \begin{pmatrix} \mathbf{A}' & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{pmatrix}$$

The Nyström method

Nyström extension of eigenvectors:

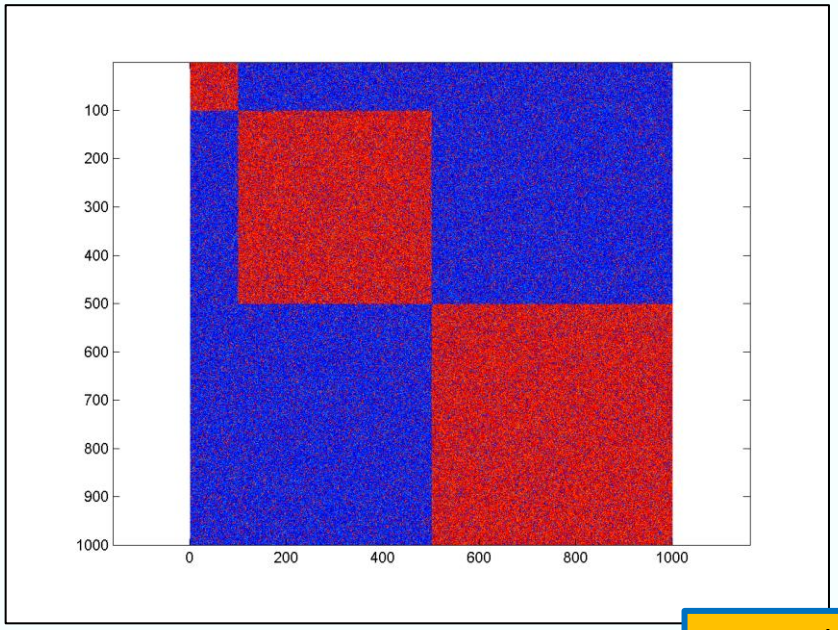
- Let **U'** hold the *k* largest eigenvector of **A'** , each with eigenvalue λ_i, $i = 1, ..., k$
- Extend $n' \times k$ matrix **U'** to $n \times k$ matrix $\mathbf{U_e}$
  - an approximation of **U** up to Π

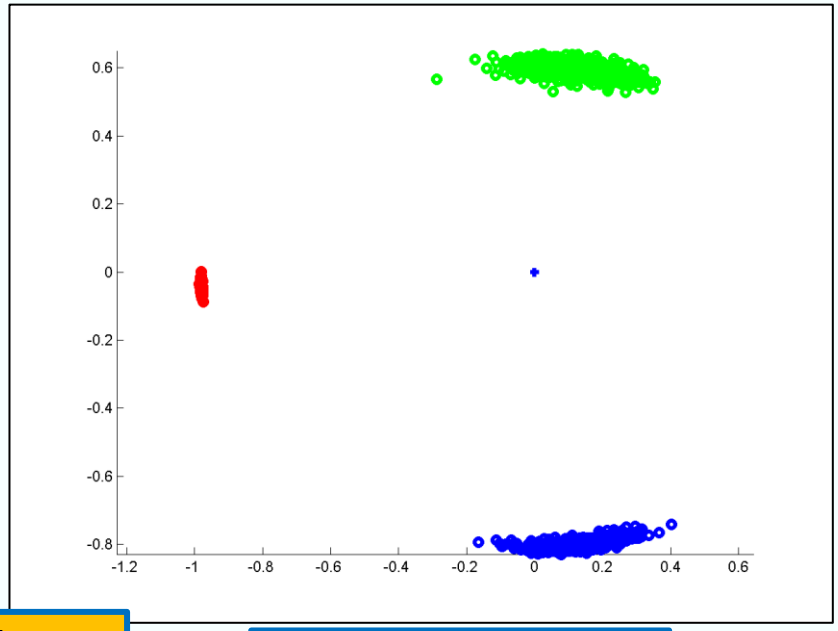
  as follows

$$\mathbf{u}_{e,i} = \frac{1}{n'\lambda_i} \begin{pmatrix} \mathbf{A}' \\ \mathbf{B}^T \end{pmatrix} \mathbf{U}'$$

B = affinities between the *n*-*n*' missing points in **A**' and the *n*' points in **A**'

Nyström method: numerical example



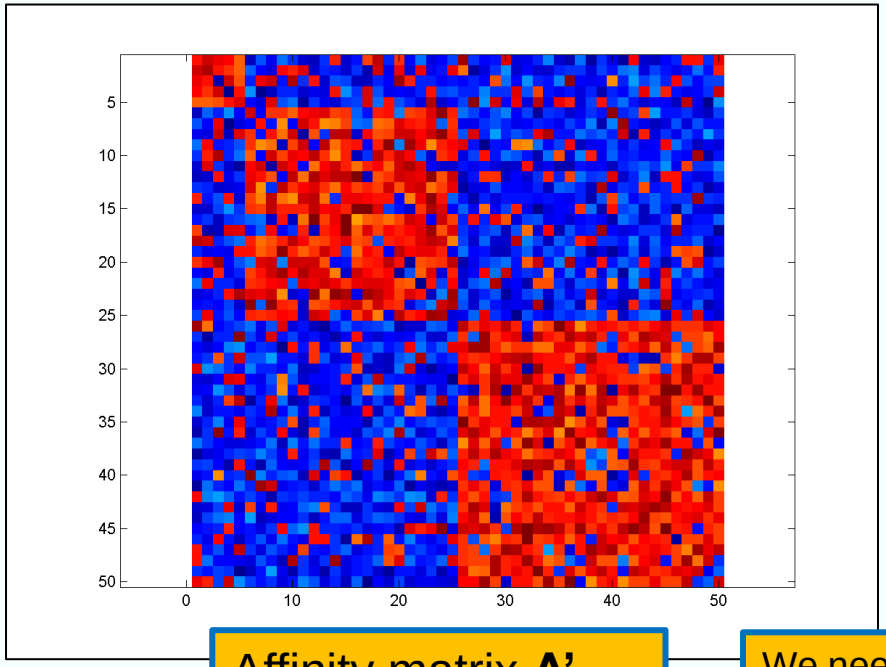


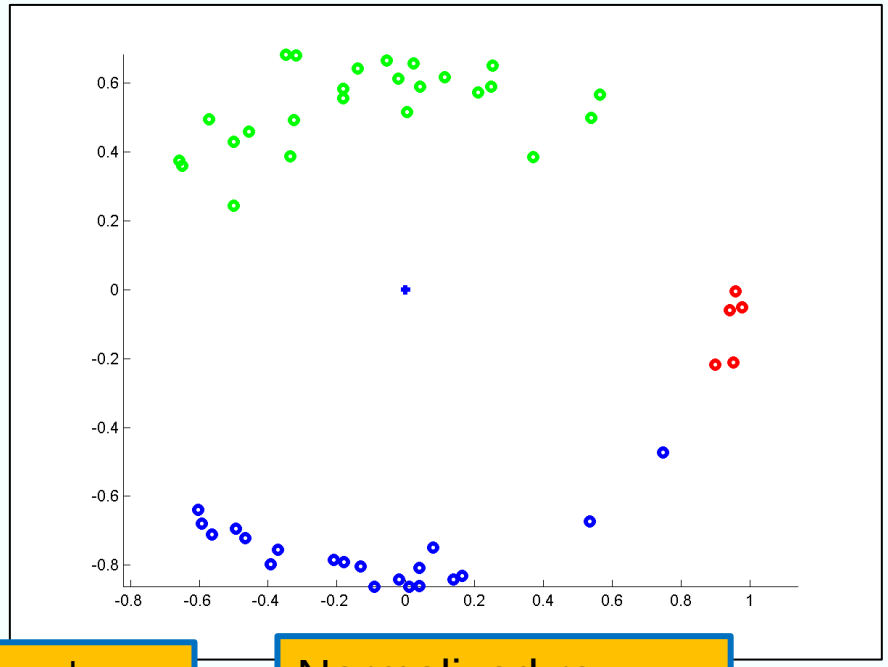
Affinity matrix **A**
1000 points

We need to solve a 1000×1000 eigenvalue problem to get here

Normalized row space of **U**

Nyström method: numerical example



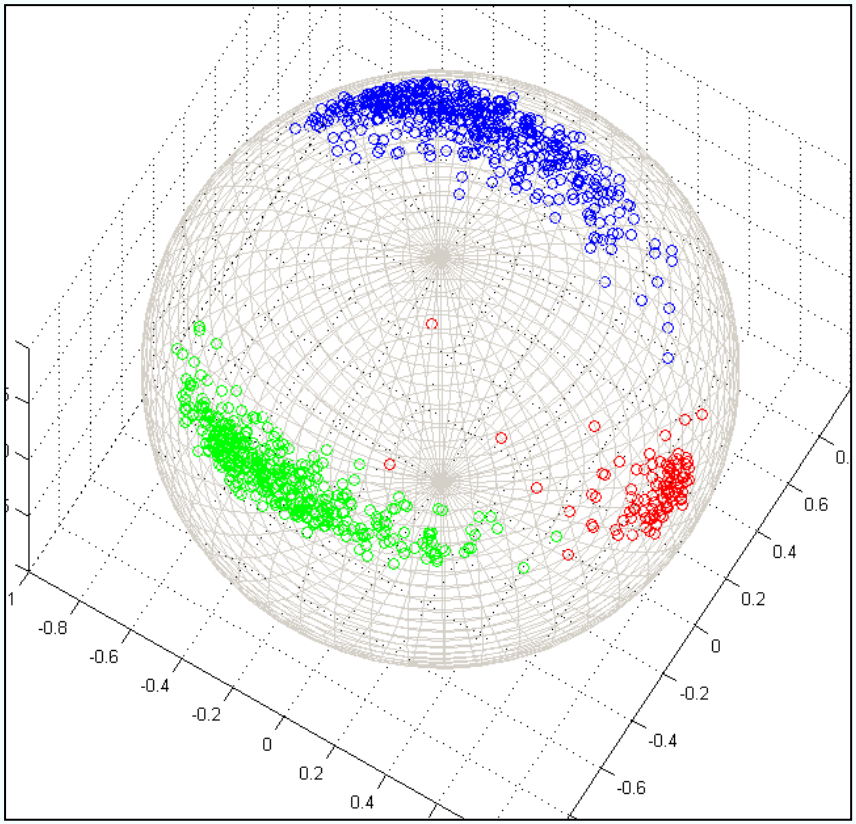


Affinity matrix **A'**
50 points

We need to solve a 50×50 eigenvalue problem to get here

Normalized row space of **U'**

Nyström method: numerical example



Normalized row space of $\mathbf{U}_e$ = approximation of **U**

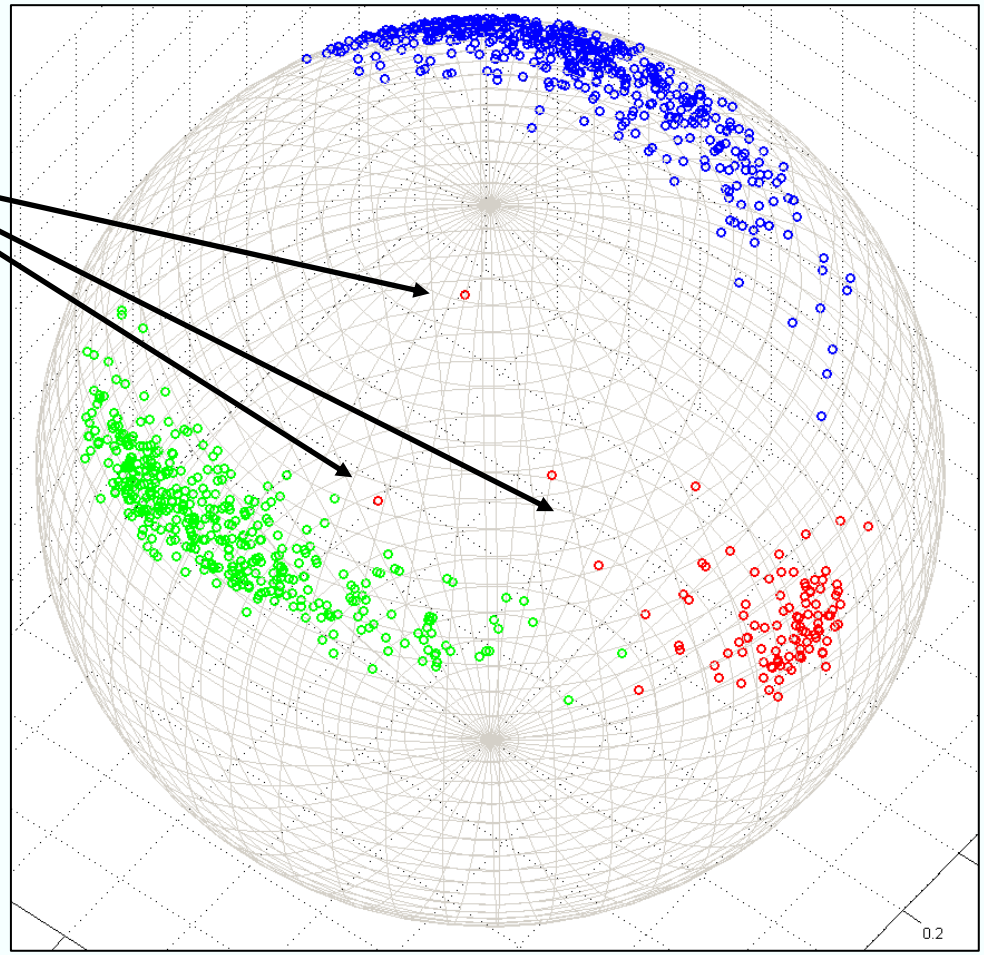
Nyström alternatives:

- Fast approximate Spectral clustering (Yan et al. KDD 2009)

Extensions: K-means

- Generally **non-spherical clusters** and **distributed on hyperspheres**

What happens to these points?



- K-means with Euclidean distance is not the best choice
- Also the **shape** of the clusters needs to be taken into consideration

Extensions: K-means

- Standard K-means tries to **minimise**

$$E = \sum_x \sum_k \left|\left| x - \mu_{k(x)} \right|\right|^2, \qquad \mu_{k(x)}$$ is the cluster mean

- The appropriate geodesic distance on the sphere is the cosine similarity

$$D(i,j) = \frac{x_i^T x_j}{\cancel{||x_i|| ||x_j||}}$$ Unit sphere!

- Thus K-means on the unit hyper-sphere instead **maximises**

$$E = \sum_x \sum_k x^T \mu_{k(x)}$$

- How is the mean defined on the sphere?

Extensions: K-means

- The mean of a cluster on the hyper-sphere
- Extrinsic (depends on the embedding space)
  - Just take the average of the cosine similarities. Simple
- Intrinsic (depends on the manifold only)
  - The **Fréchet** mean (discrete)

$$\mu_{k(x)} = \arg\min_{x \in M} \left(\frac{1}{n} \sum_i D(x_i, x_j)^2 \right)$$

point guaranteed to lie on the manifold but requires nonlinear optimisation on the spherical manifold

- Simpler alternative approximation:
  - Take the average of the cosine similarities and normalise

$$\mu_{k(x)} \approx \sum_{x \in k} x \, / \, || \sum_{x \in k} x ||$$

Extensions: K-means

- But we have non-spherical clusters:
  - Use the sq. **Mahalanobis** distance between a point and a cluster

$$D_M(x,k) = (x - \mu_k)^T \Sigma^{-1} (x - \mu_k),$$ Σ is the covariance matrix

  - On the hyper-sphere it becomes

$$D_M(x) = \mu_k x^T \Sigma^{-1} \mu_k x$$

  - What is the covariance on the manifold?
    - Quite complicated (see X. Pennec). Defined on the tangent space at the mean point of the cluster
    - Use the extrinsic formulation instead but with cosine similarities

A K-Mahalanobis algorithm on the hyper-sphere

1. Initialise k clusters with at least k points each (k is the number of dims)
2. Calculate mean of each cluster k $\mu_{k(x)} \approx \sum_{x \in k} x / || \sum_{x \in k} x ||$
3. Calculate covariance matrix of each cluster k
4. Calculate Mahalanobis distances of each point x_j to each cluster mean μ_k

$$D_M(x) = \mu_k x^T \Sigma^{-1} \mu_k x$$

5. Assign point x_j to cluster with minimum distance
6. Goto 2 until convergence

- Does it work any better? Depends
  - If the clusters are compact then K-Means and K-Mahalanobis on the hyper-sphere should be similar
  - If the clusters are dispersed and non-isotropic then the K-Mahalanobis on the hyper-sphere should be better
  - **Remember: When we search for parameters the clusters tend to be dispersed and far away**

Putting some ideas together

- A real segmentation problem…or “segmenting green from green”
- Task: Segment the 4 leaves

- Challenge: Everything looks the same! Variations only on small scales near pixel level.

- Assumptions:
  - We know that the plant is approximately on the center of the image.
  - We know there are 4 leaves
  - They are approximately elliptical

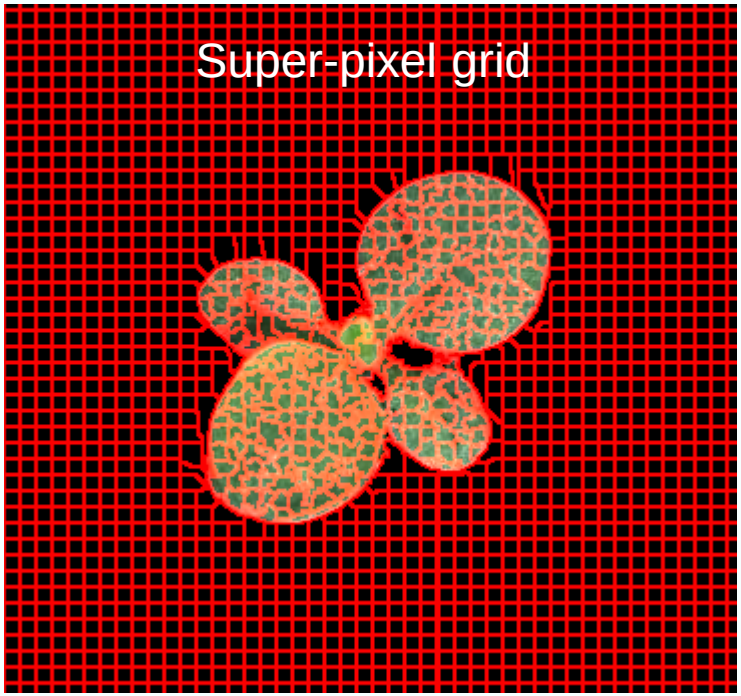
- Notions:
  - Multiple-affinities
  - Pairwise affinities
  - Automatic-parameter tuning
  - Problem-specific cluster quality

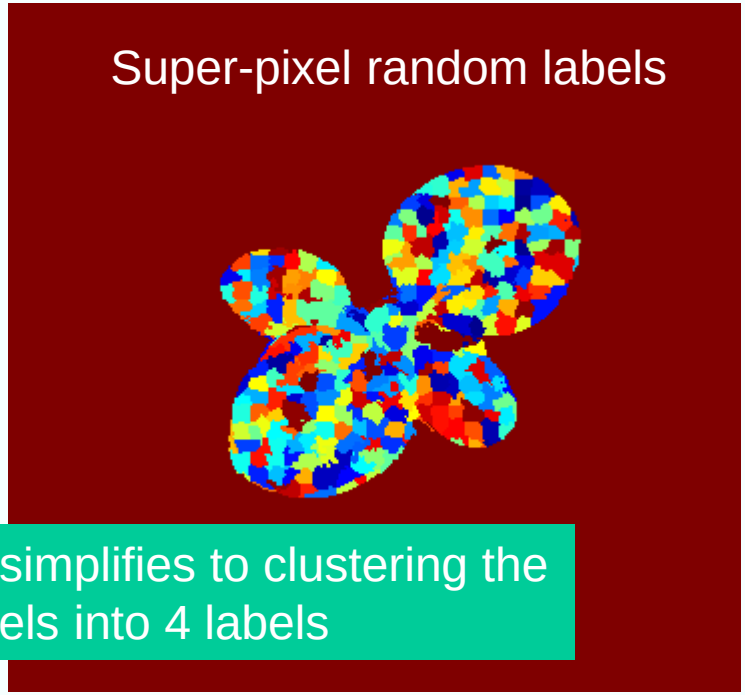
Leaf segmentation

- First step: The plant is green! So threshold the green stuff from the background. Simplifies the problem

Leaf segmentation

- Second step: Working on pixel levels is very expensive. But we need to capture texture variations on the smaller scales.
- At large scales everything looks the same
- Subdivide the image into small regions
- Instead of regular patches use super-pixels. They can adapt to the local shape variations. Preserves boundaries

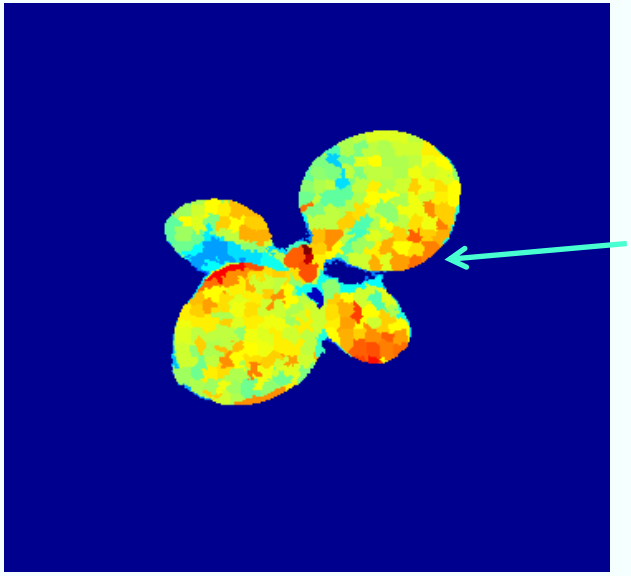




The task simplifies to clustering the super-pixels into 4 labels

Leaf segmentation: Affinity combination

- Extract a texture descriptor (e.g. Weibull distribution) at each sub-pixel patch s_i
- Pairwise texture affinity is defined as $A_T(i,j) = exp(-R(s_i, s_j)/\sigma_T^2)$
- R is the Rao distance between the distributions of two patches



Small scale patches are good for differentiating between nearby patches

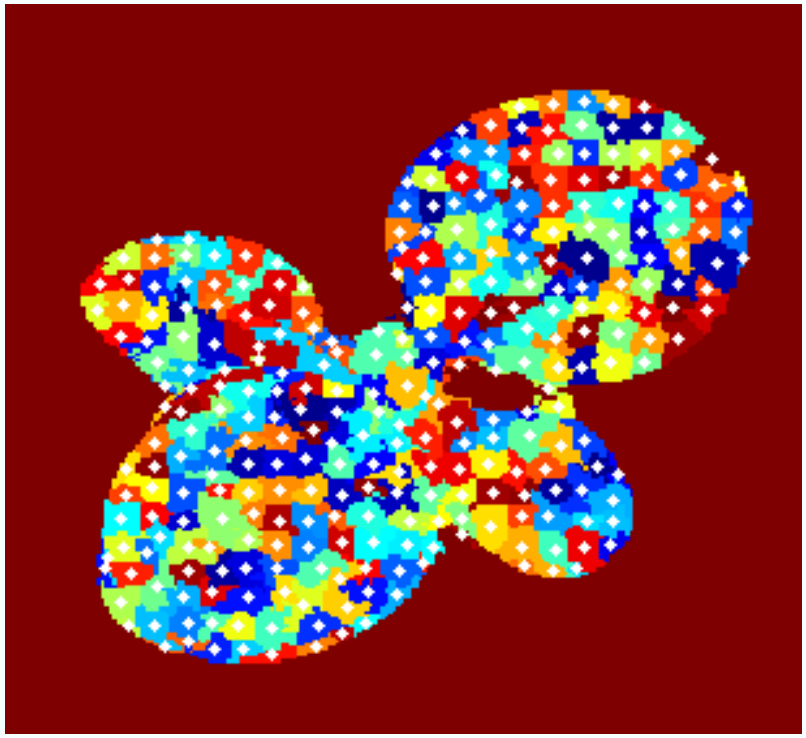
Especially around borders

Larger patches are not that good

Leaf segmentation: Affinity combination

- Texture does not help in distant patches. We need an additional affinity.
- Euclidean distance of super-pixel centroids

$$A_E(i,j) = exp(-||C_{s_i} - Cs_j||^2/\sigma_E^2)$$



- Affinity combination via weighted **Hadamard product**

$$A = A_T . * A_E^W$$

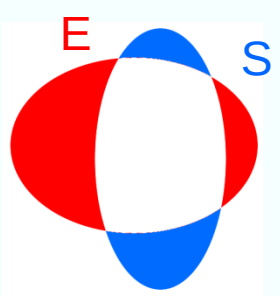
- **The weight W is the parameter we are looking for**

Leaf segmentation: Parameter tuning via problem specific quality measure

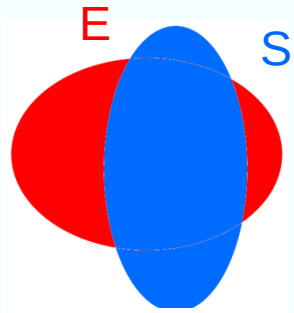
- Even with the combination of texture + distance affinities the problem is quite hard. We need to find a good combination parameter

- Parameter tuning
  - We know that the leaves are elliptical. Thus fit a geometric model (ellipse) and check residual
  - Quality measure is the total overlap error between the ellipses E and the segments S. Both are binary images

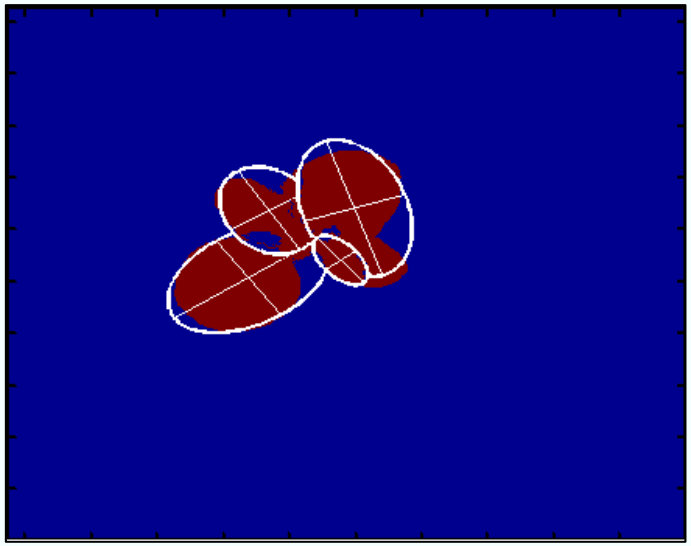
$$q = \frac{E \cup S - E \cap S}{E \cup S}$$



$E \cup S - E \cap S$



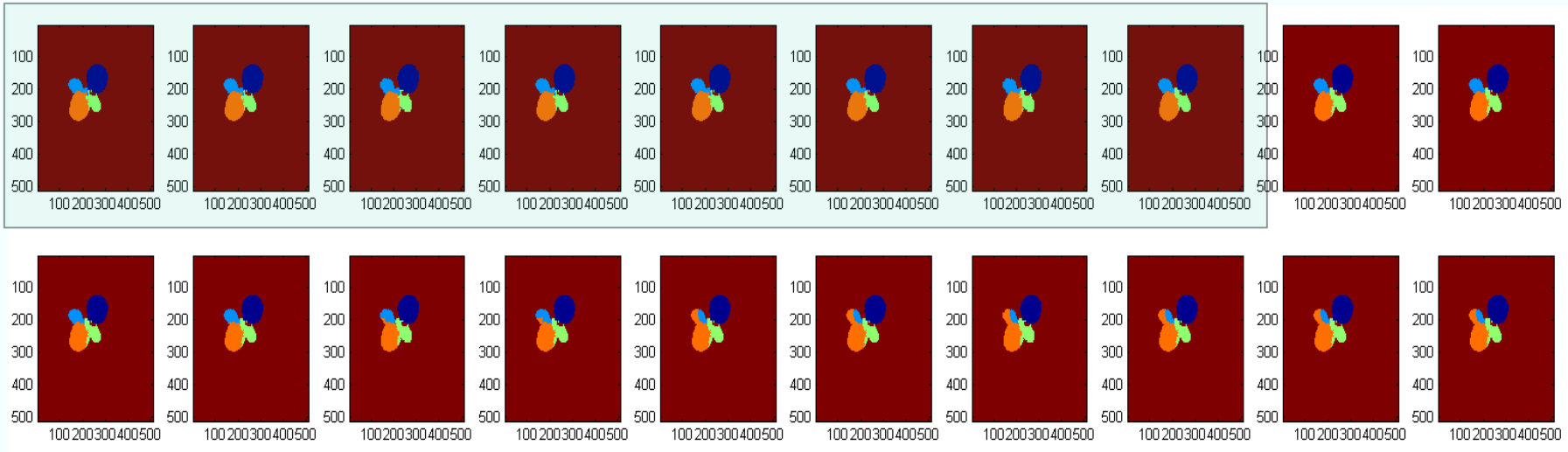
$E \cup S$



Leaf segmentation

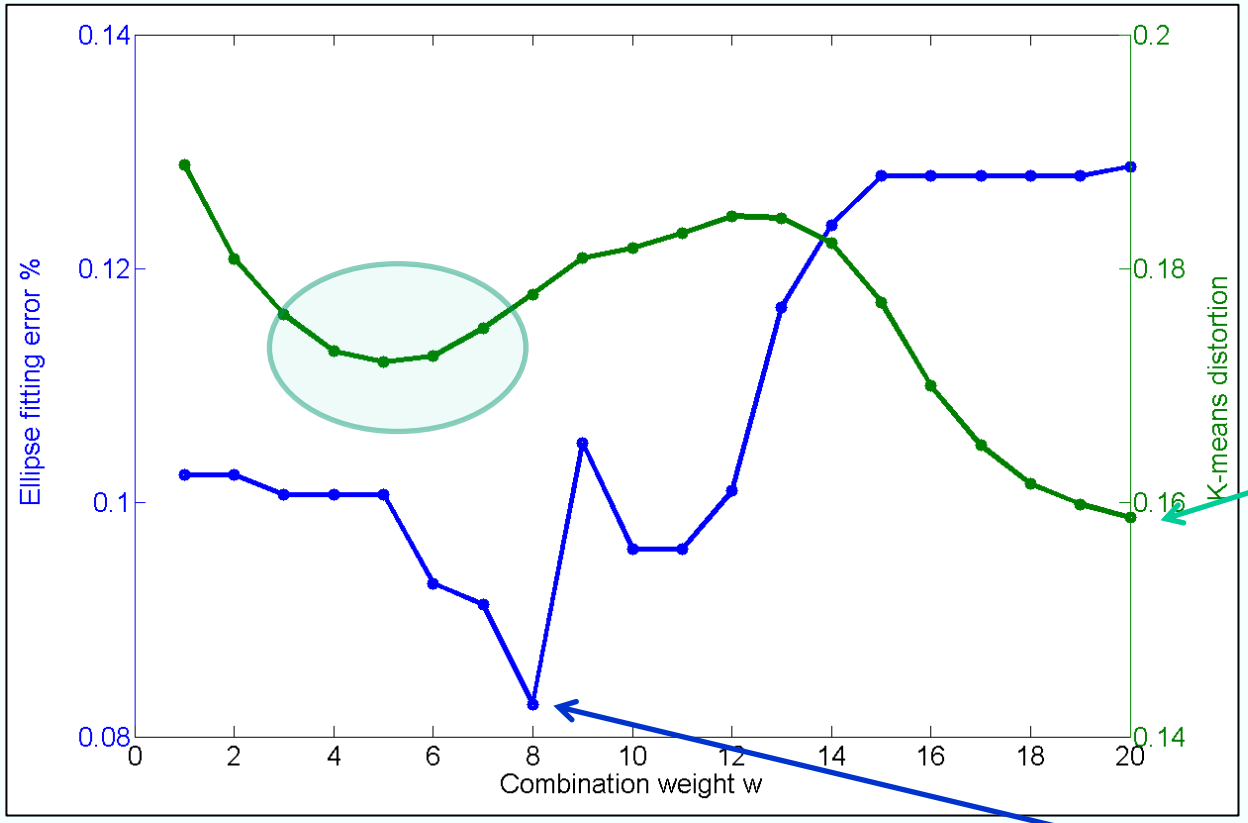
- Searching for the combination parameter ***w***

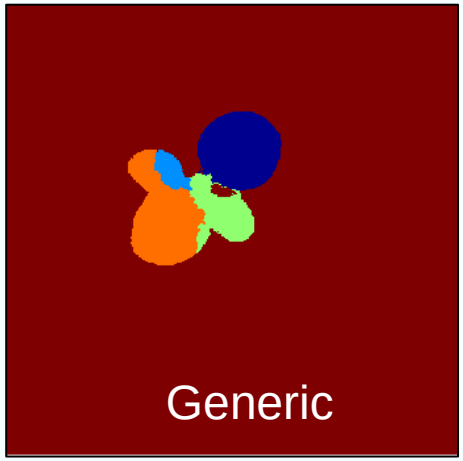
Reasonable solutions

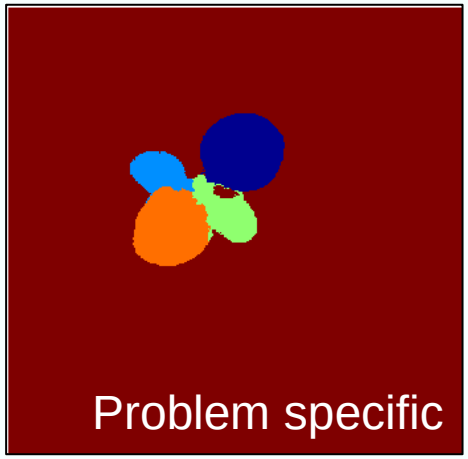


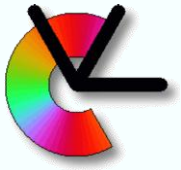
Leaf segmentation

- K-means distortion (generic) vs geometric (problem specific)









Summary

- What have we learned in this introductory course?
- **Lecture 1:** Core ideas of SC and connection to graphs
- **Lecture 2:** Intuitive explanation of SC mechanics and different SC algorithms
- **Lecture 3:** Practical issues (Parameter tuning, multiple-vews, multi-way affinities, large affinity matrices, real applications)