Repetition, Consolidation, and Applications

Topics Covered So Far

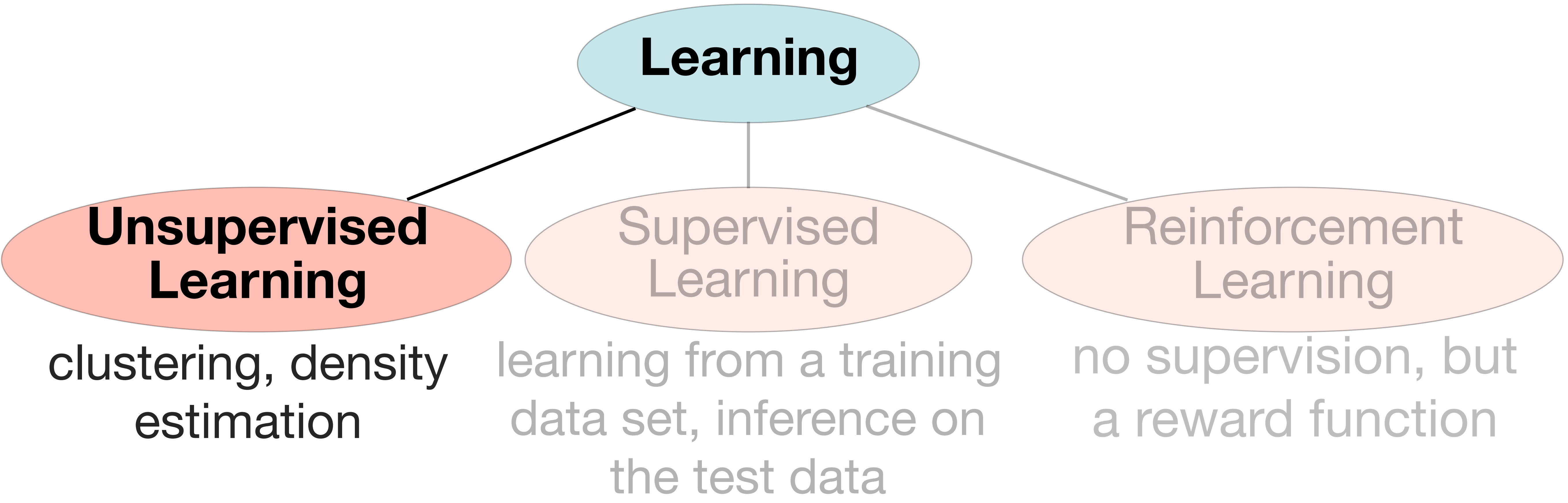
- Clustering and Expectation Maximization
- Kernel Methods and GPs
- Boosting and Bagging
- Graphical Models
- Hidden Markov Models
- Deep Learning
- Metric Learning

3. Clustering

Motivation

- Supervised learning is good for interaction with humans, but labels from a supervisor are sometimes hard to obtain
- Clustering is **unsupervised** learning, i.e. it tries to learn only from the data
- Main idea: find a similarity measure and group similar data objects together
- Clustering is a very old research field, many approaches have been suggested
- Main problem in most methods: how to find a good number of clusters

Categories of Learning


Learning
Unsupervised Learning
clustering, density estimation
Supervised Learning
learning from a training data set, inference on the test data
Reinforcement Learning
no supervision, but a reward function

In unsupervised learning, there is no **ground truth** information given.

Most Unsupervised Learning methods are based on **Clustering**.

K-means Clustering

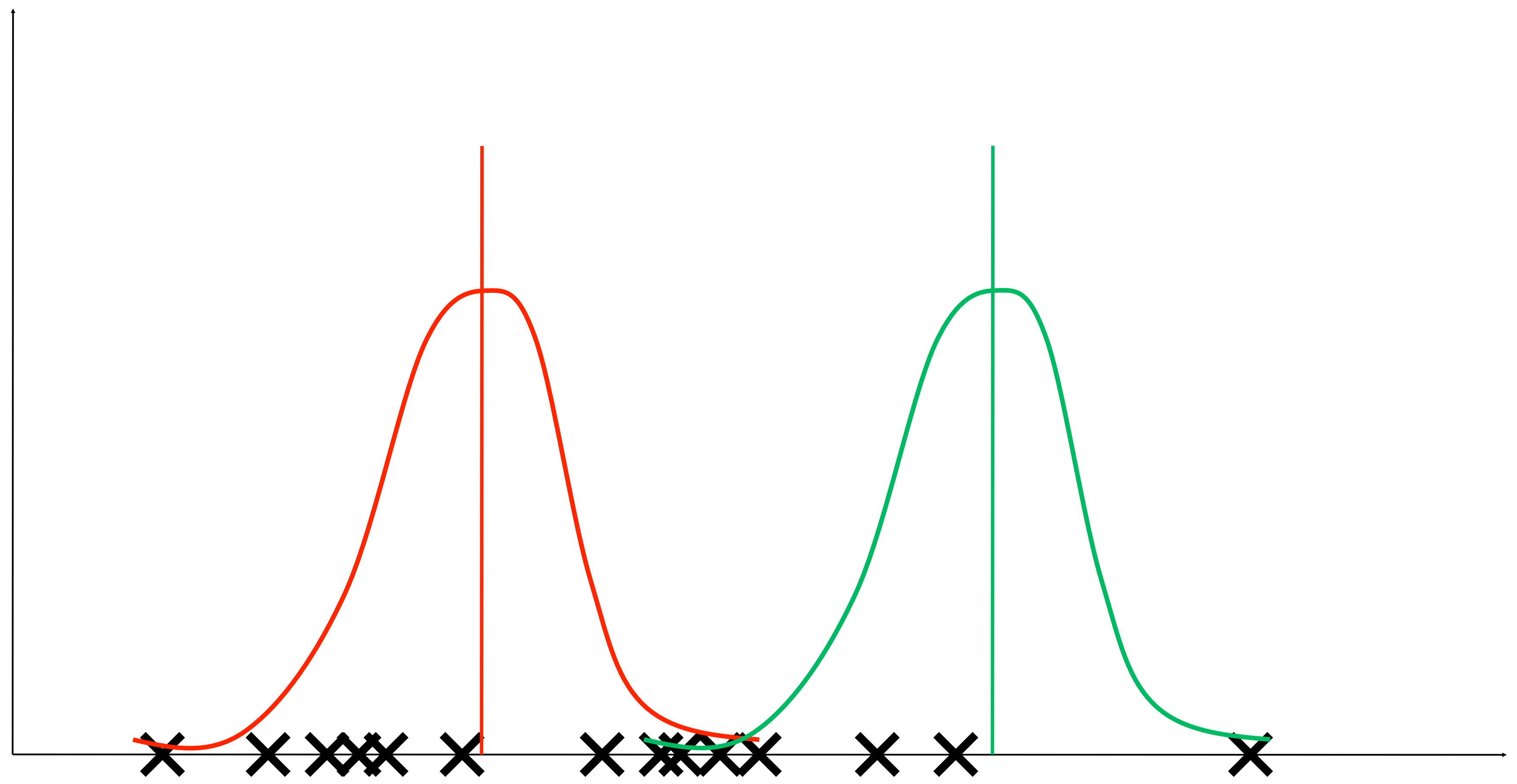
- Given: data set $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, number of clusters K
- Goal: find cluster centers $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$ so that

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

  is minimal, where $r_{nk} = 1$ if $\mathbf{x}_n$ is assigned to $\boldsymbol{\mu}_k$
- Idea: compute r_{nk} and $\boldsymbol{\mu}_k$ iteratively
- Start with some values for the cluster centers
- Find optimal assignments r_{nk}
- Update cluster centers using these assignments
- Repeat until assignments or centers don't change

K-means Clustering

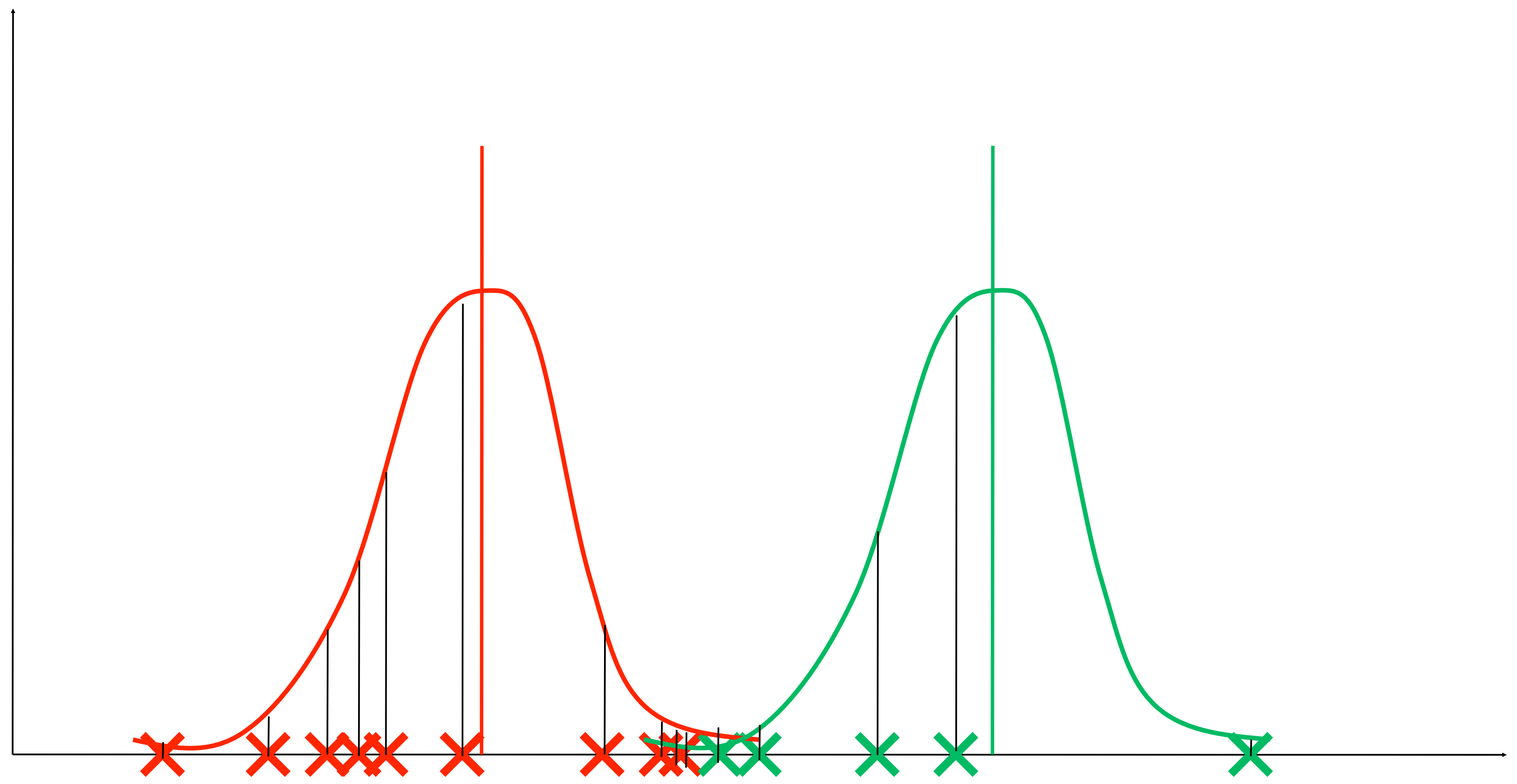
Initialize cluster means: $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$



K-means Clustering

Find optimal assignments:

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\| \\ 0 & \text{otherwise} \end{cases}$$

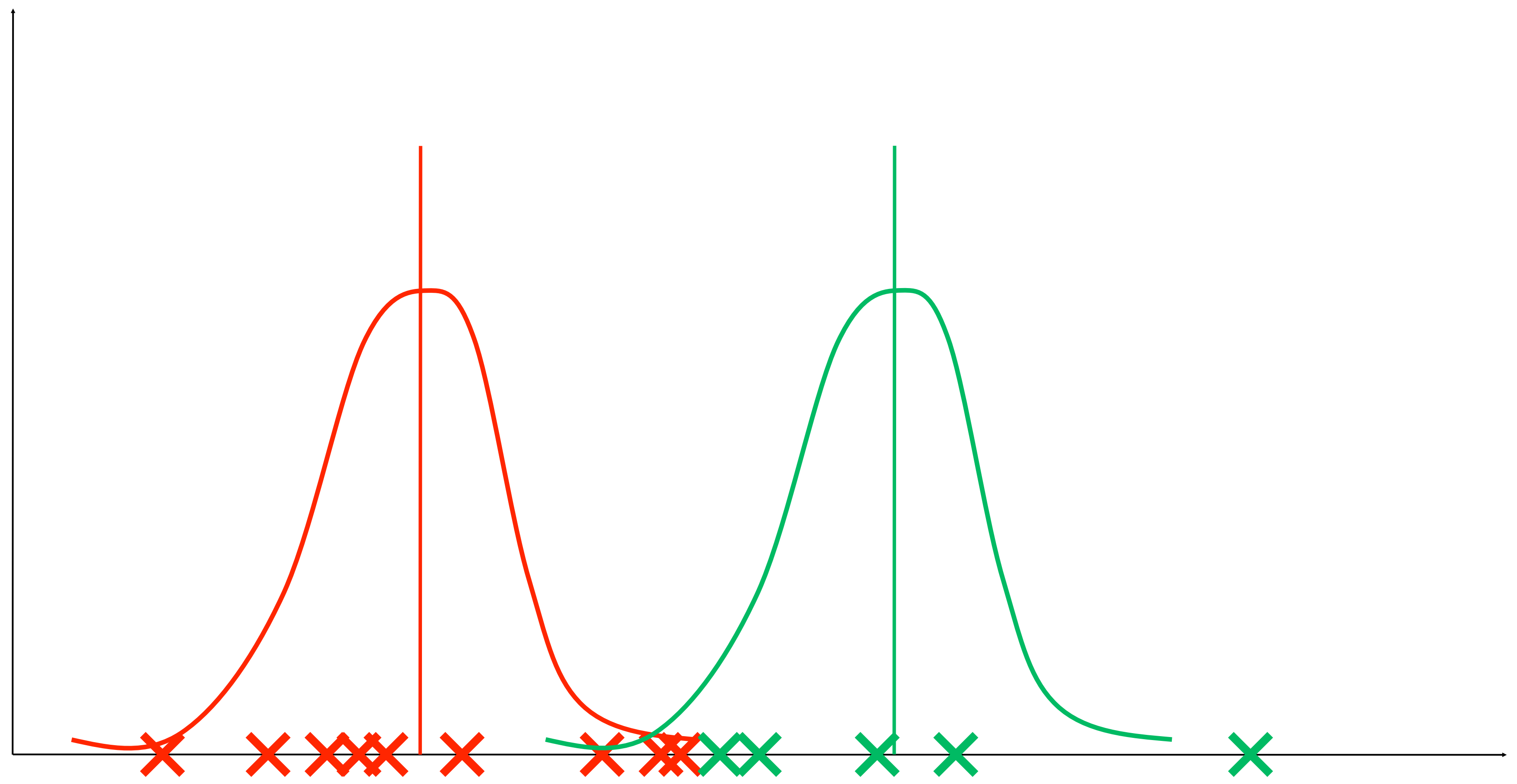


K-means Clustering

Find new optimal means:

$$\frac{\partial J}{\partial \boldsymbol{\mu}_k} = 2 \sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k) \overset{!}{=} 0$$

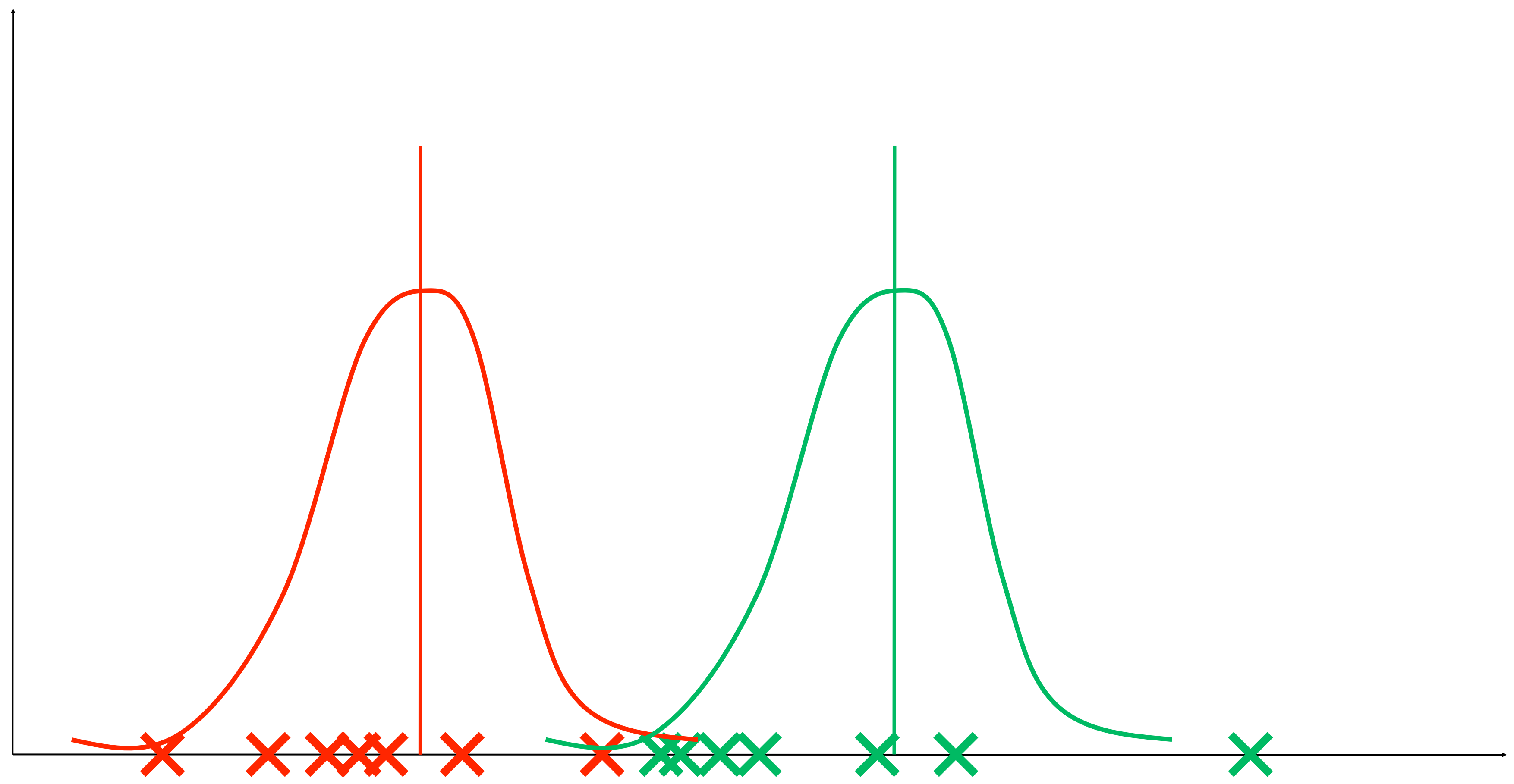
$$\Rightarrow \boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} r_{nk}\mathbf{x}_n}{\sum_{n=1}^{N} r_{nk}}$$



K-means Clustering

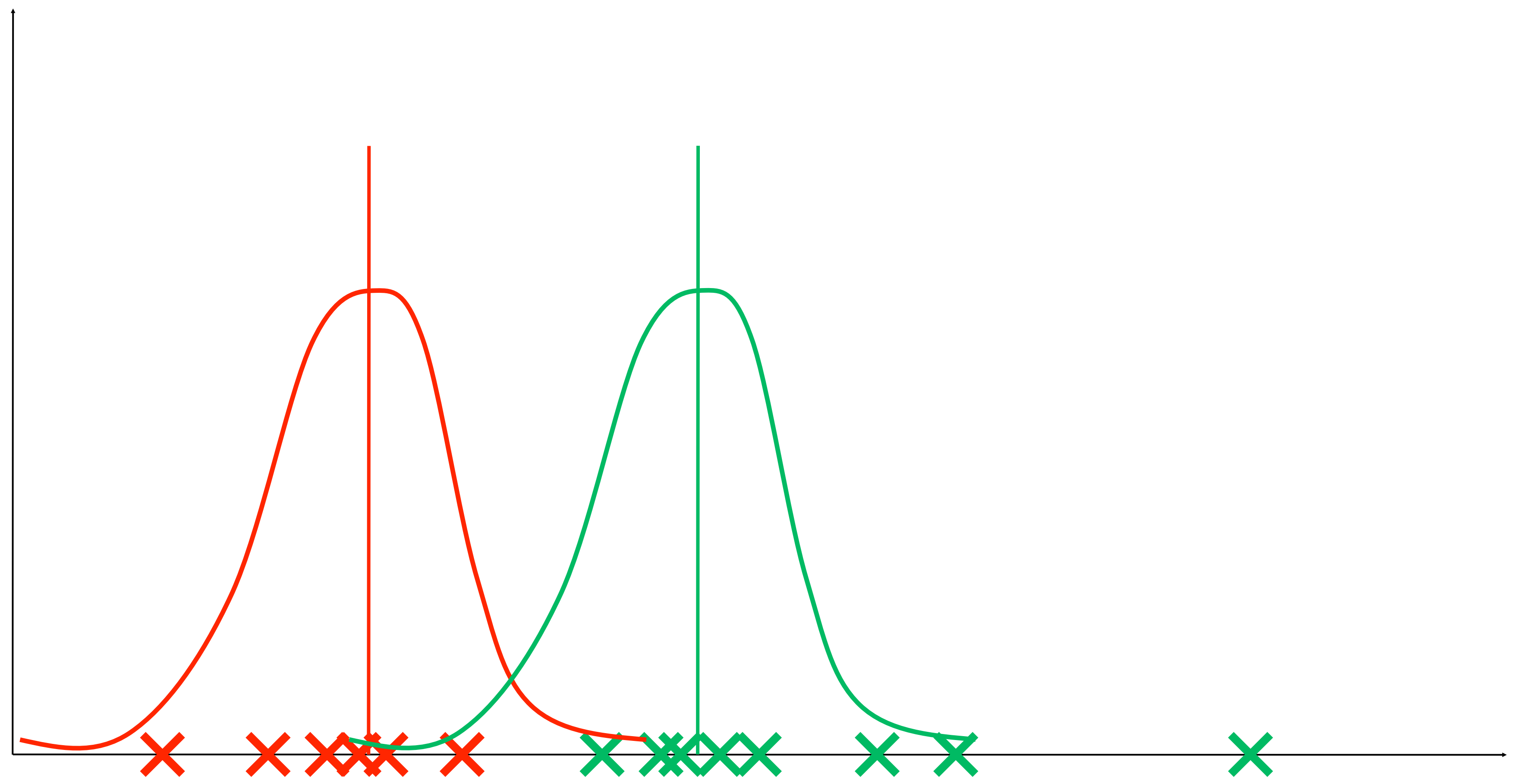
Find new optimal assignments:

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\| \\ 0 & \text{otherwise} \end{cases}$$

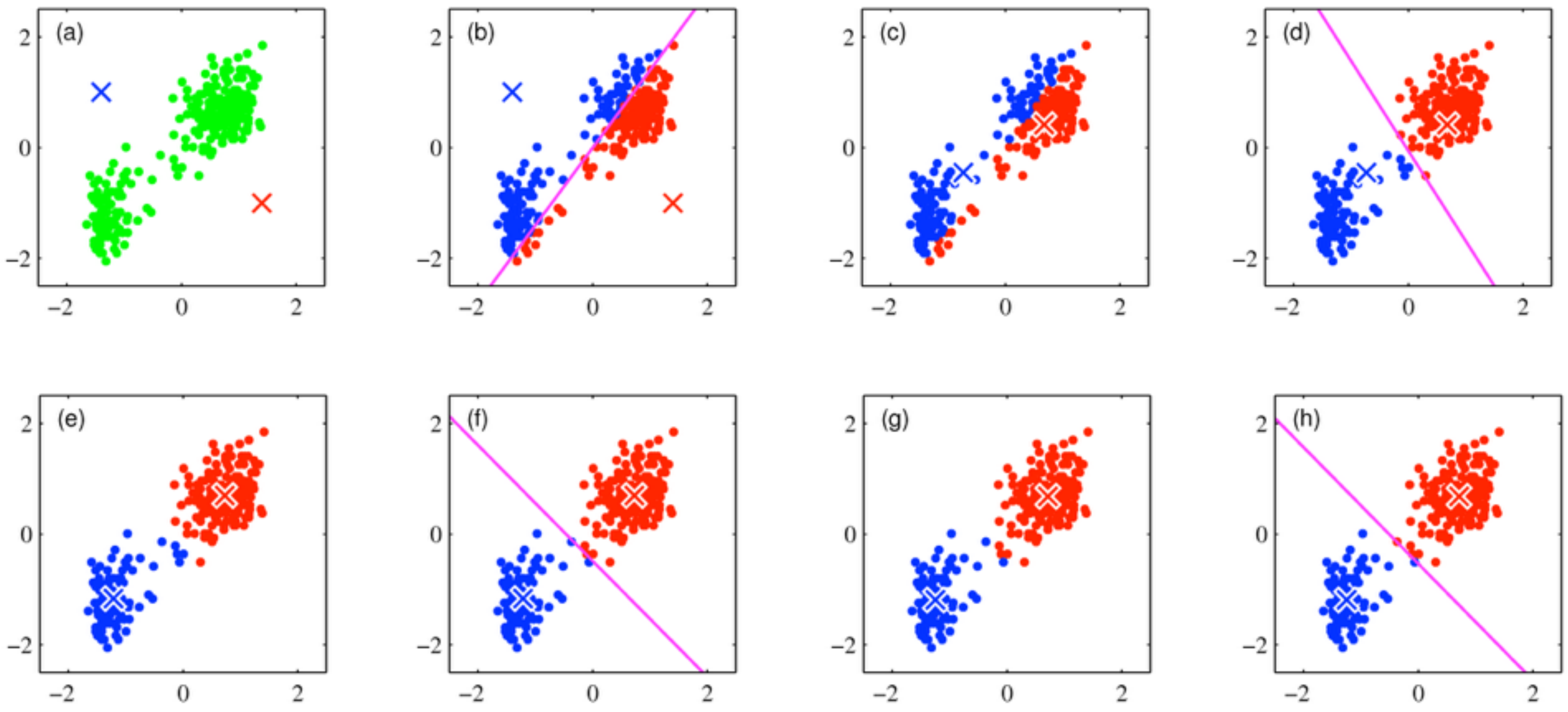


K-means Clustering

Iterate these steps until means and assignments do not change any more



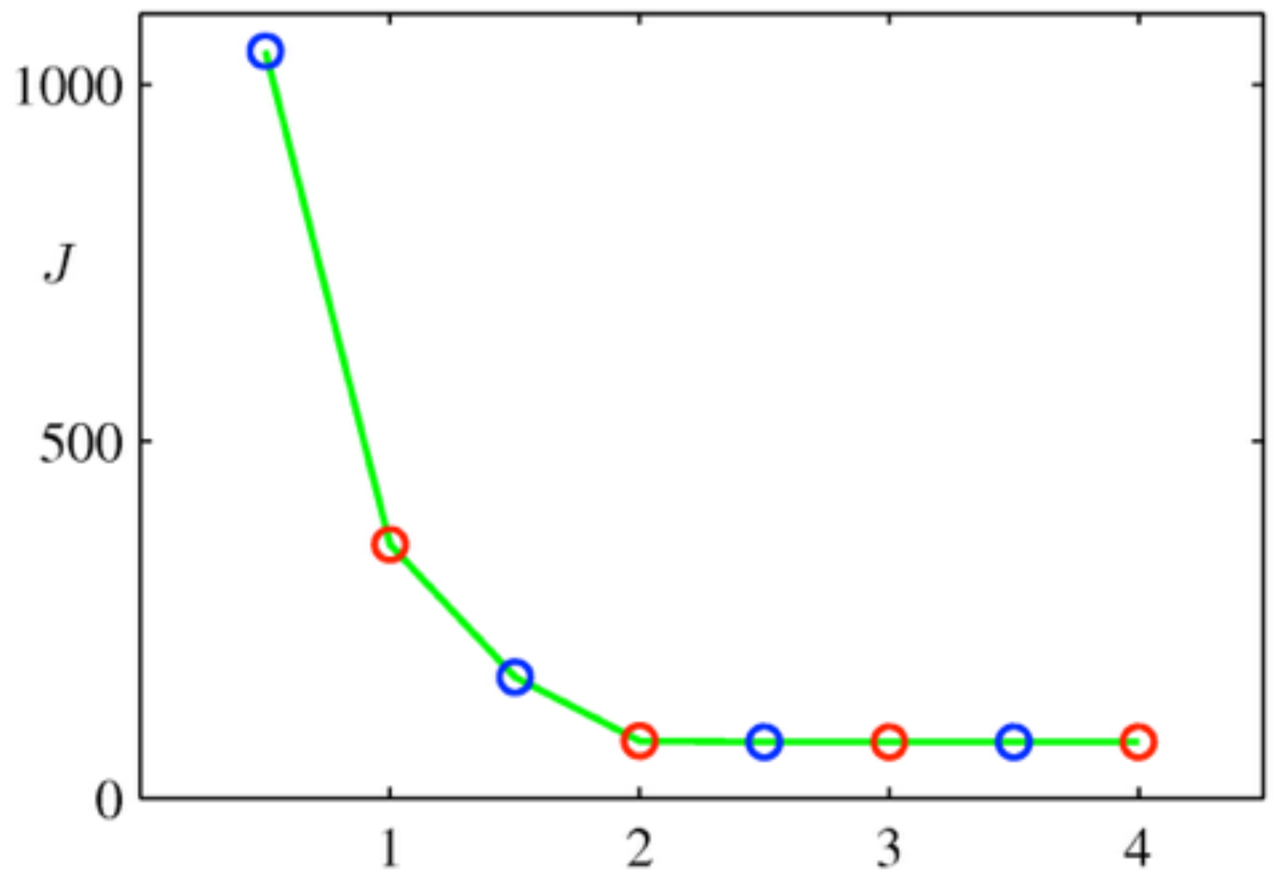
2D Example



- Real data set
- Random initialization

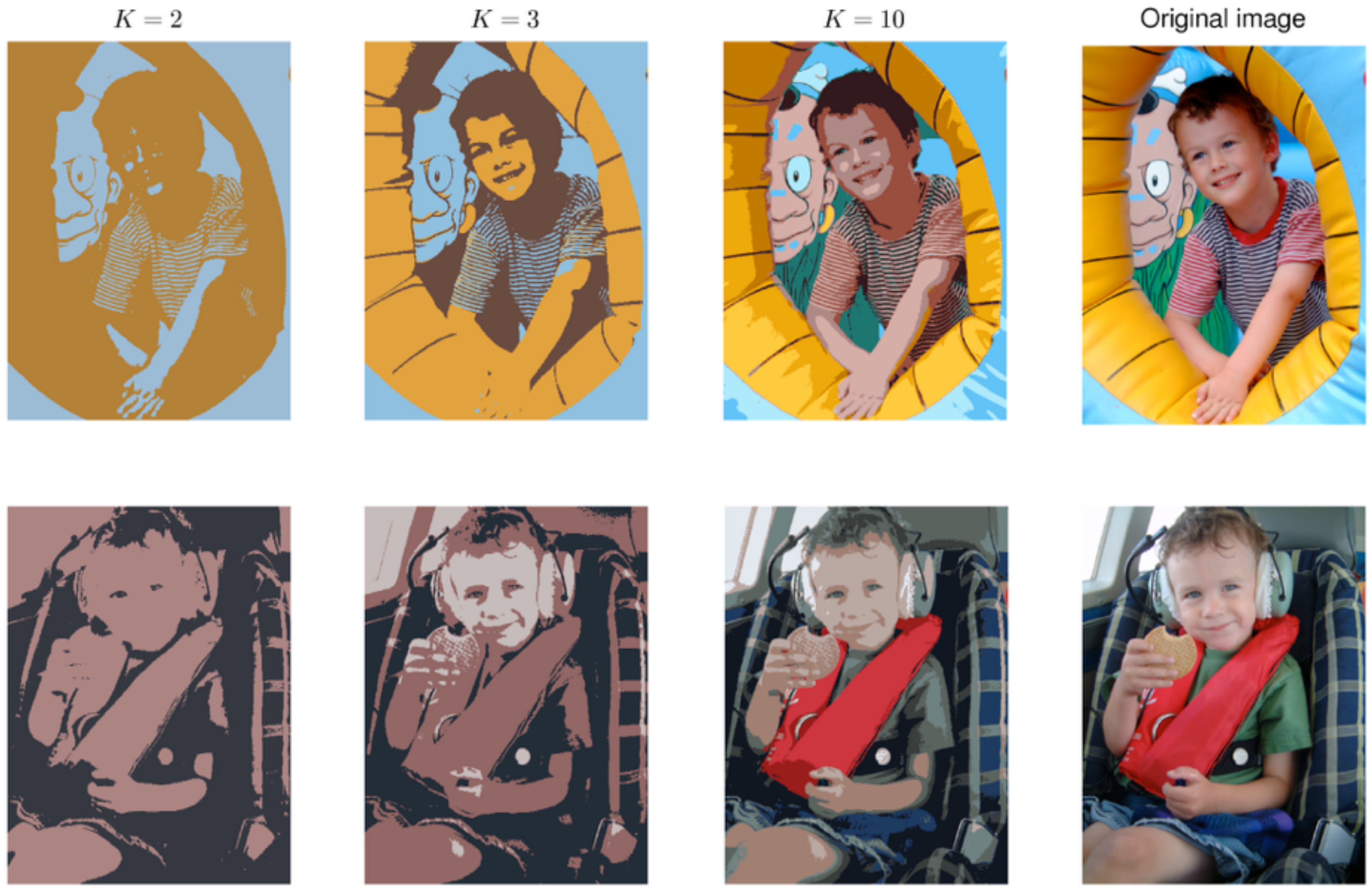
- Magenta line is "decision boundary"

The Cost Function



- After every step the cost function J is minimized
- Blue steps: update assignments
- Red steps: update means
- Convergence after 4 rounds

K-means for Segmentation



K-Means: Additional Remarks

- K-means converges always, but the minimum is not guaranteed to be a global one
- There is an **online** version of K-means
    - After each addition of $\mathbf{x}_n$, the nearest center $\boldsymbol{\mu}_k$ is updated:

$$\boldsymbol{\mu}_k^{\text{new}} = \boldsymbol{\mu}_k^{\text{old}} + \eta_n(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{old}})$$

- The ***K*-medoid** variant:
    - Replace the Euclidean distance by a general measure *V*.

$$\tilde{J} = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \mathcal{V}(\mathbf{x}_n, \boldsymbol{\mu}_k)$$

Mixtures of Gaussians

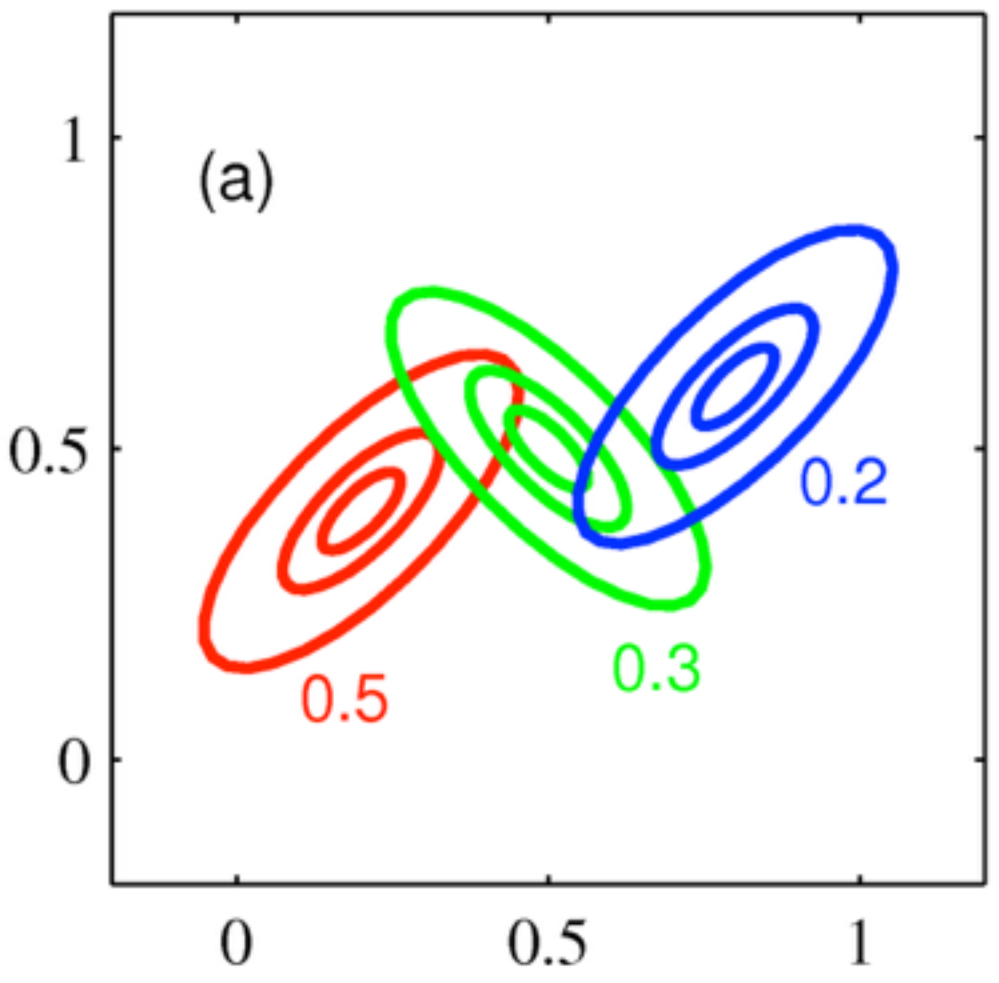
- Assume that the data consists of K clusters
- The data within each cluster is Gaussian
- For any data point $\mathbf{x}$ we introduce a K-dimensional binary random variable $\mathbf{z}$ so that:

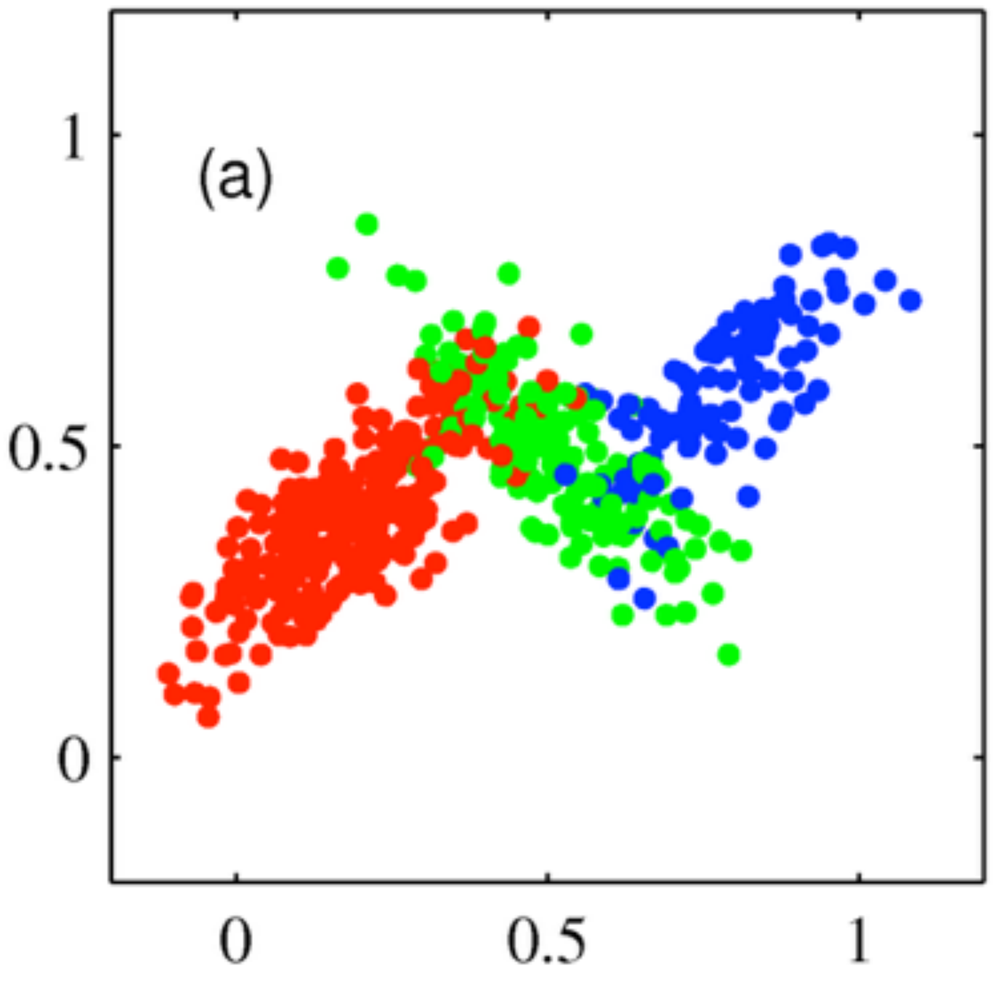
$$p(\mathbf{x}) = \sum_{k=1}^{K} \underbrace{p(z_k = 1)}_{=:\pi_k} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k)$$

where

$$z_k \in \{0, 1\}, \quad \sum_{k=1}^{K} z_k = 1$$

A Simple Example





- Mixture of three Gaussians with mixing coefficients
- Left: all three Gaussians as contour plot
- Right: samples from the mixture model, the red component has the most samples

Parameter Estimation

- From a given set of training data $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ we want to find parameters $(\pi_{1,\dots,K}, \boldsymbol{\mu}_{1,\dots,K}, \Sigma_{1,\dots,K})$ so that the likelihood is maximized (MLE):

$$p(\mathbf{x}_1, \dots, \mathbf{x}_N \mid \pi_{1,\dots,K}, \boldsymbol{\mu}_{1,\dots,K}, \Sigma_{1,\dots,K}) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \Sigma_k)$$

or, applying the logarithm:

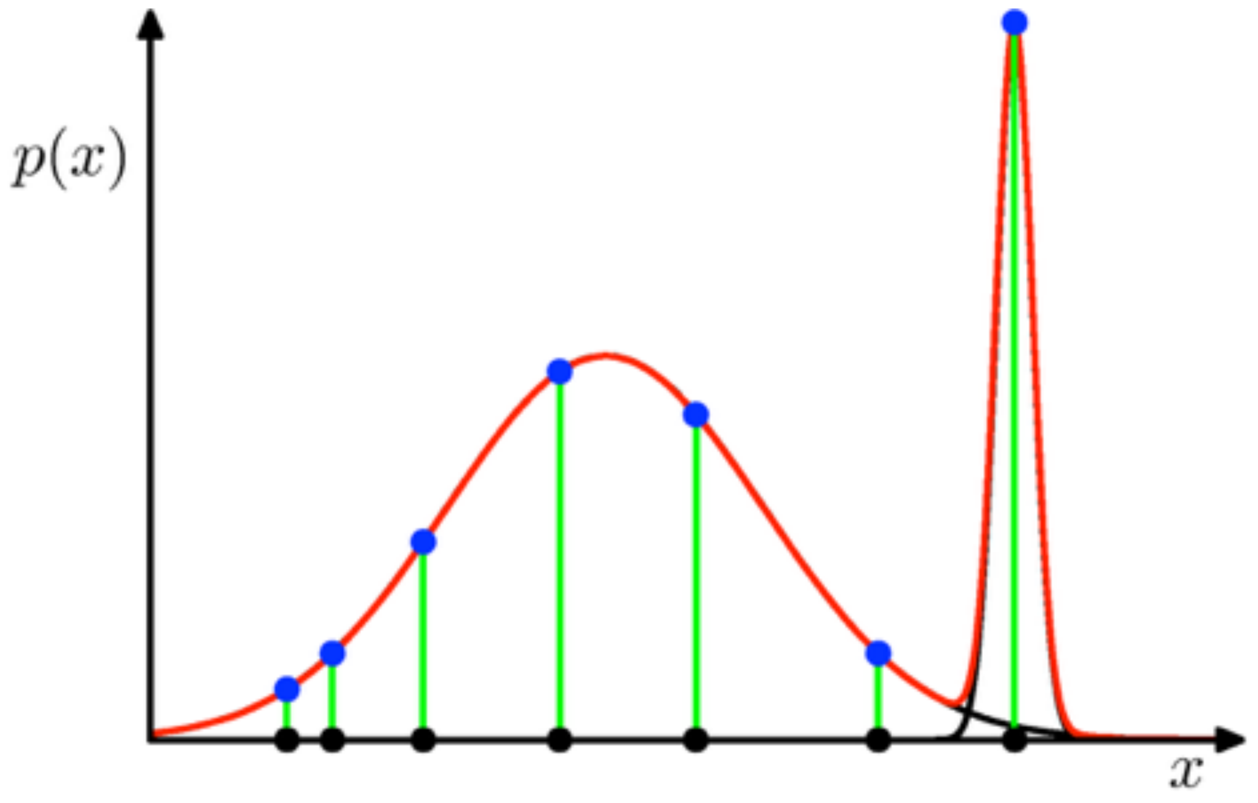
$$\log p(X \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma) = \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \Sigma_k)$$

- However: this is not as easy as maximum-likelihood for single Gaussians!

Problems with MLE for Gaussian Mixtures

- Assume that for one k the mean $\boldsymbol{\mu}_k$ is exactly at a data point $\mathbf{x}_n$
  - For simplicity: assume that $\Sigma_k = \sigma_k^2 I$
  - Then: $$\mathcal{N}(\mathbf{x}_n \mid \mathbf{x}_n, \sigma_k^2 I) = \frac{1}{\sqrt{2\pi}\sigma_k^D}$$
  - This means that the overall log-likelihood can be maximized arbitrarily by letting $\sigma_k \to 0$ (**overfitting**)
- Another problem is the **identifiability**:
  - The order of the Gaussians is not fixed, therefore:
  - There are $K!$ equivalent solutions to the MLE problem

Overfitting with MLE for Gaussian Mixtures



- One Gaussian fits exactly to one data point
- It has a very small variance, i.e. contributes strongly to the overall likelihood
- In standard MLE, there is no way to avoid this!

Expectation-Maximization

- EM is an elegant and powerful method for MLE problems with latent variables
- Main idea: model parameters and latent variables are estimated iteratively, where average over the latent variables (expectation)
- A typical example application of EM is the Gaussian Mixture model (GMM)
- However, EM has many other applications
- First, we consider EM for GMMs

Expectation-Maximization for GMM

- First, we define the **responsibilities:**

$$\gamma(z_{nk}) = p(z_{nk} = 1 \mid \mathbf{x}_n) \qquad z_{nk} \in \{0, 1\}$$

$$\sum_k z_{nk} = 1$$

Expectation-Maximization for GMM

- First, we define the **responsibilities:**

$$\gamma(z_{nk}) = p(z_{nk} = 1 \mid \mathbf{x}_n)$$

$$= \frac{\pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \Sigma_j)}$$

Expectation-Maximization for GMM

- First, we define the **responsibilities:**

$$\gamma(z_{nk}) = p(z_{nk} = 1 \mid \mathbf{x}_n)$$

$$= \frac{\pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \Sigma_j)}$$

- Next, we derive the log-likelihood wrt. to $\boldsymbol{\mu}_k$:

$$\frac{\partial \log p(X \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma)}{\partial \boldsymbol{\mu}_k} \overset{!}{=} \mathbf{0}$$

Expectation-Maximization for GMM

- First, we define the **responsibilities:**

$$\gamma(z_{nk}) = p(z_{nk} = 1 \mid \mathbf{x}_n)$$

$$= \frac{\pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \Sigma_j)}$$

- Next, we derive the log-likelihood wrt. to $\boldsymbol{\mu}_k$:

$$\frac{\partial \log p(X \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma)}{\partial \boldsymbol{\mu}_k} \overset{!}{=} \mathbf{0}$$

and we obtain: $$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})}$$

Expectation-Maximization for GMM

- We can do the same for the covariances:

$$\frac{\partial \log p(X \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma)}{\partial \Sigma_k} \overset{!}{=} \mathbf{0}$$

and we obtain:

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})}$$

- Finally, we derive wrt. the mixing coefficients π_k :

$$\frac{\partial \log p(X \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma)}{\partial \pi_k} \overset{!}{=} \mathbf{0} \qquad \text{where:} \qquad \sum_{k=1}^{K} \pi_k = 1$$

Expectation-Maximization for GMM

- We can do the same for the covariances:

$$\frac{\partial \log p(X \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma)}{\partial \Sigma_k} \overset{!}{=} \mathbf{0}$$

and we obtain:

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})}$$

- Finally, we derive wrt. the mixing coefficients π_k :

$$\frac{\partial \log p(X \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma)}{\partial \pi_k} \overset{!}{=} \mathbf{0} \quad \text{where:} \quad \sum_{k=1}^{K} \pi_k = 1$$

and the result is: $\pi_k = \frac{1}{N} \sum_{n=1}^{N} \gamma(z_{nk})$

Algorithm Summary

1. Initialize means $\boldsymbol{\mu}_k$ covariance matrices Σ_k and mixing coefficients $\boldsymbol{\pi}_k$
2. Compute the initial log-likelihood $\log p(X \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma)$

3. E-Step. Compute the responsibilities:

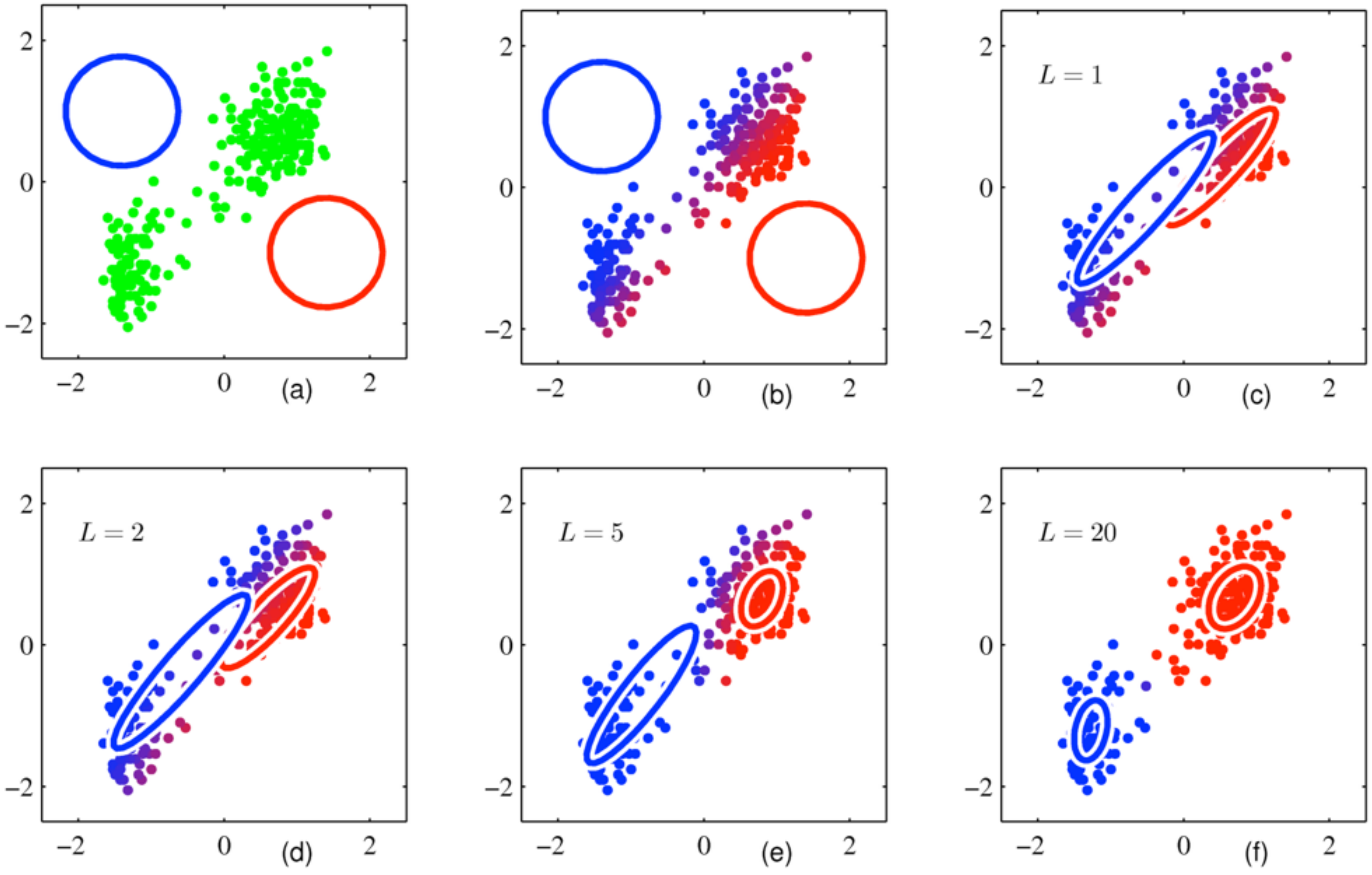
$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \Sigma_j)}$$

4. M-Step. Update the parameters:

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})} \quad \Sigma_k^{\text{new}} = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})^T}{\sum_{n=1}^{N} \gamma(z_{nk})} \quad \pi_k^{\text{new}} = \frac{1}{N} \sum_{n=1}^{N} \gamma(z_{nk})$$

5. Compute log-likelihood; if not converged go to 3.

The Same Example Again



Observations

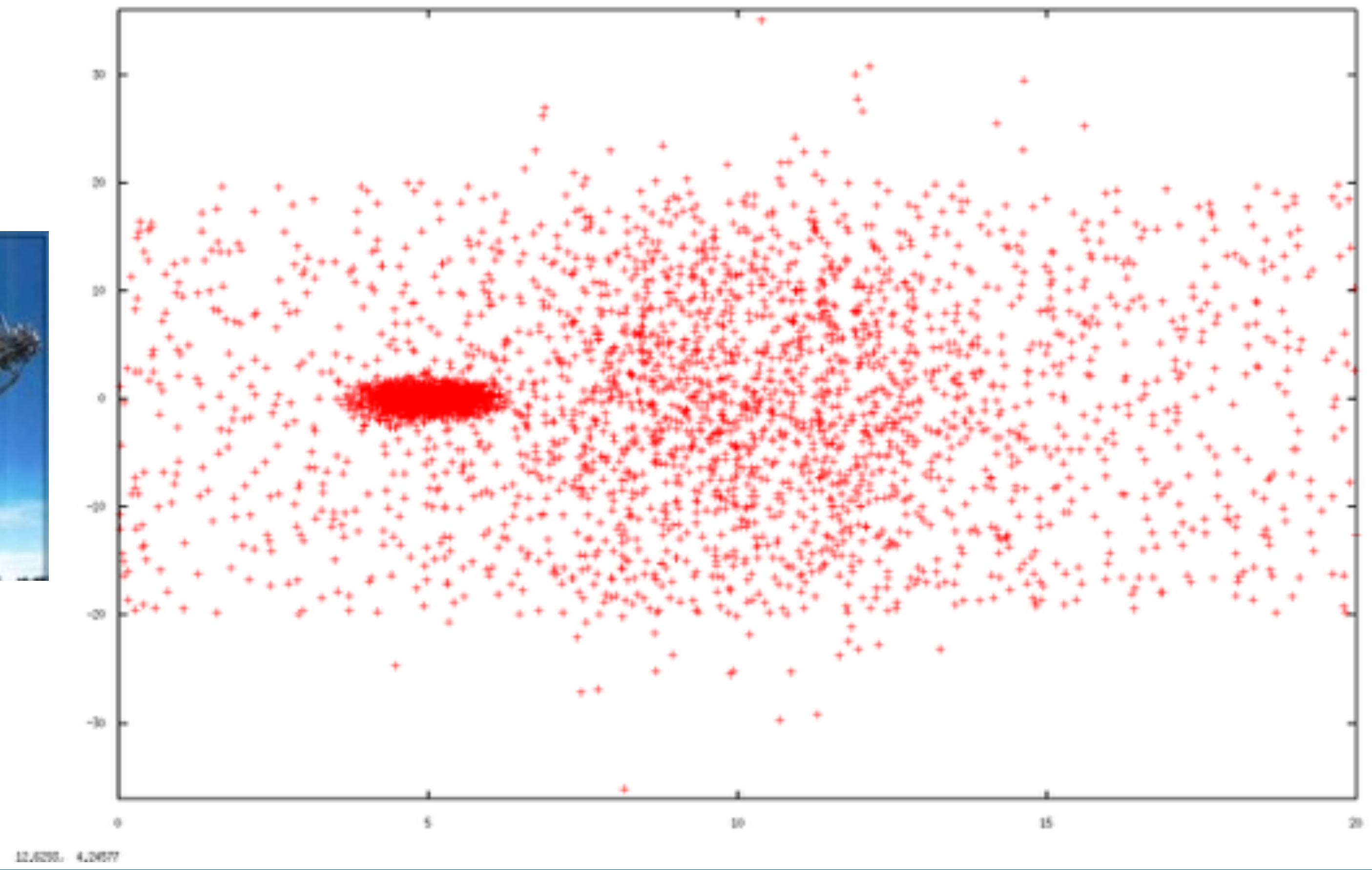
- Compared to K-means, points can now belong to both clusters (**soft assignment**)
- In addition to the cluster center, a covariance is estimated by EM
- Initialization is the same as used for K-means
- Number of iterations needed for EM is much higher
- Also: each cycle requires much more computation
- Therefore: start with K-means and run EM on the result of K-means (covariances can be initialized to the sample covariances of K-means)
- EM only finds a **local** maximum of the likelihood!

Variants of EM

- Instead of maximizing the log-likelihood, we can use EM to maximize a **posterior** when a prior is given (MAP instead of MLE) ⇒ less overfitting
- In Generalized EM, the M-step only increases the lower bound instead of maximization (useful if standard M-step is intractable)
- Similarly, the E-step can be generalized in that the optimization wrt. *q* is not complete
- Furthermore, there are incremental versions of EM, where data points are given sequentially and the parameters are updated after each data point.

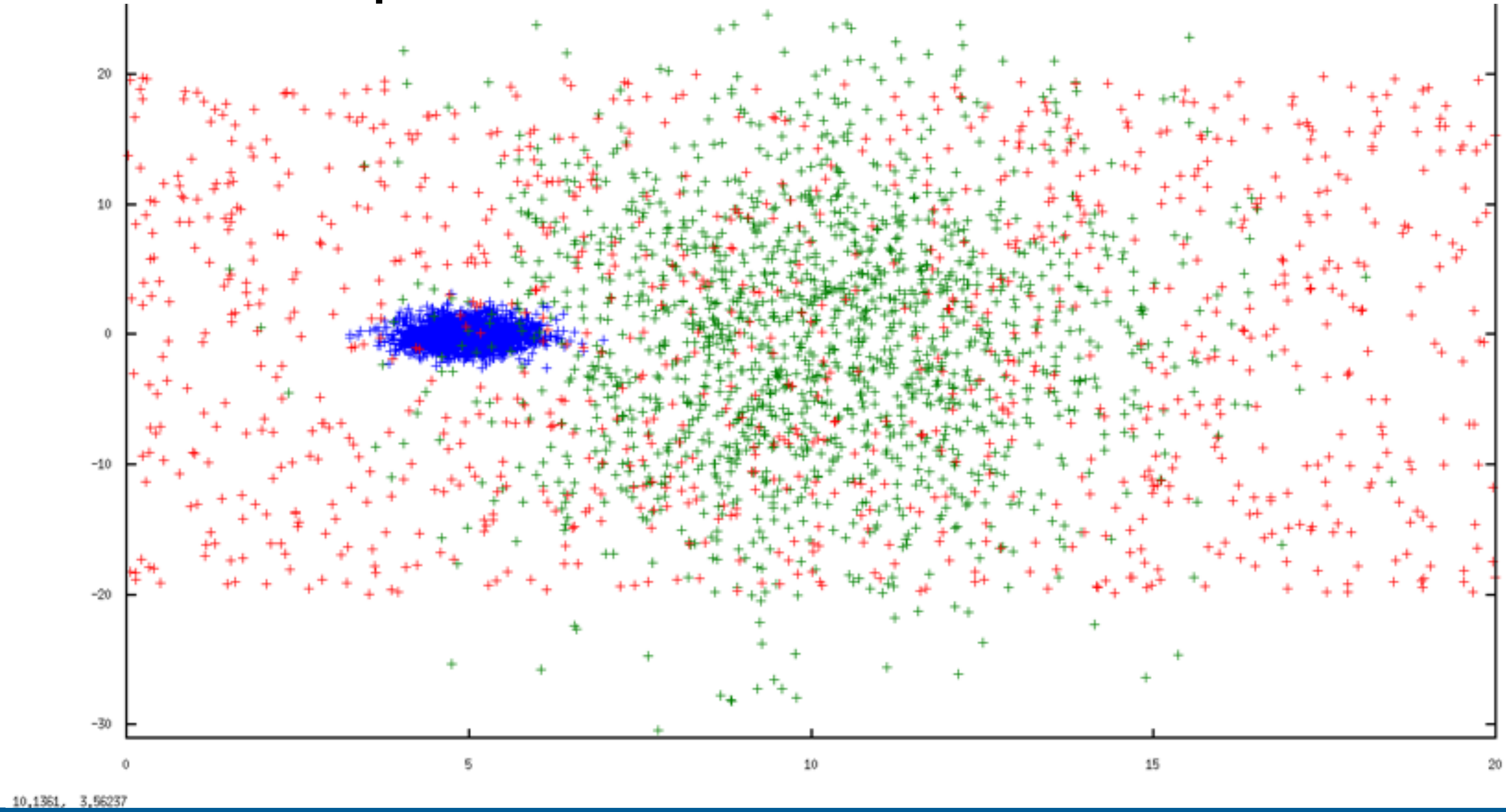
Example 1: Learn a Sensor Model

- A Radar range finder on a metallic target will returns 3 types of measurement:
  - The distance to target
  - The distance to the wall behind the target
  - A completely random value



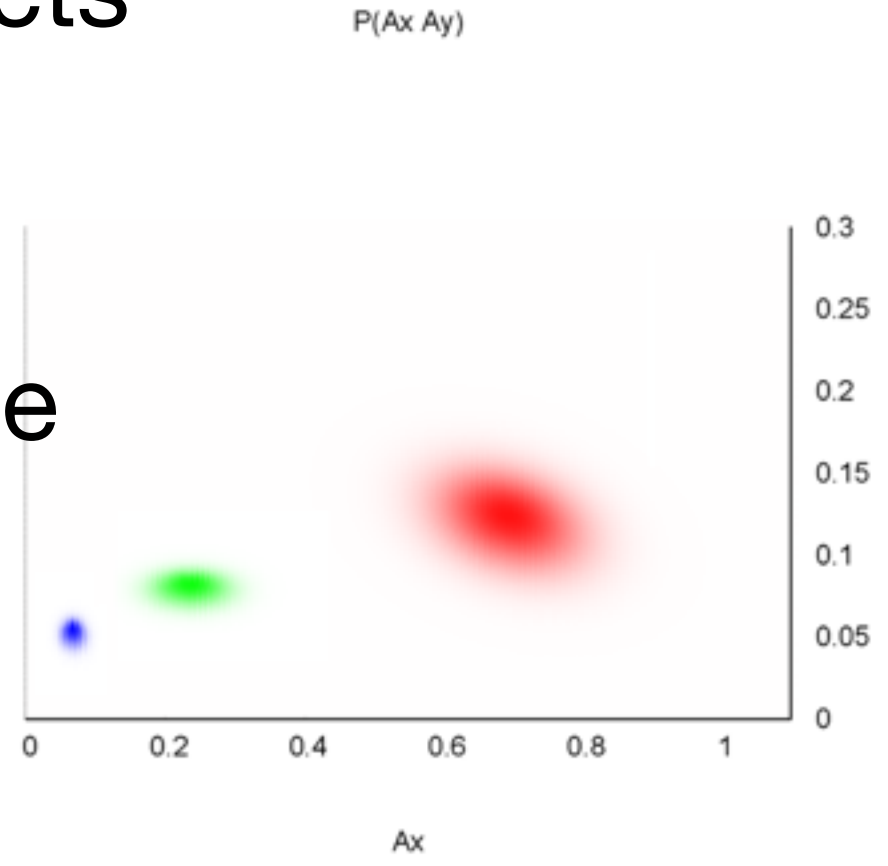
Example 1: Learn a Sensor Model

- Which point corresponds to from which model?
- What are the different model parameters?
- Solution: Expectation-Maximization



Example 2: Environment Classification

- From each image, the robot extracts features: => points in nD space
- K-means only finds the cluster centers, not their extent and shape
- The centers and covariances can be obtained with EM



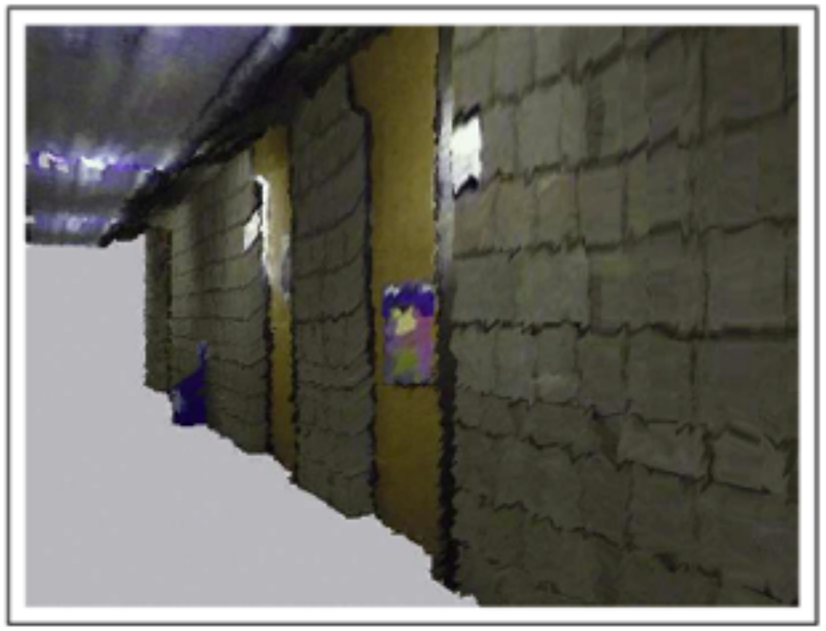
Example 3: Plane Fitting in 3D

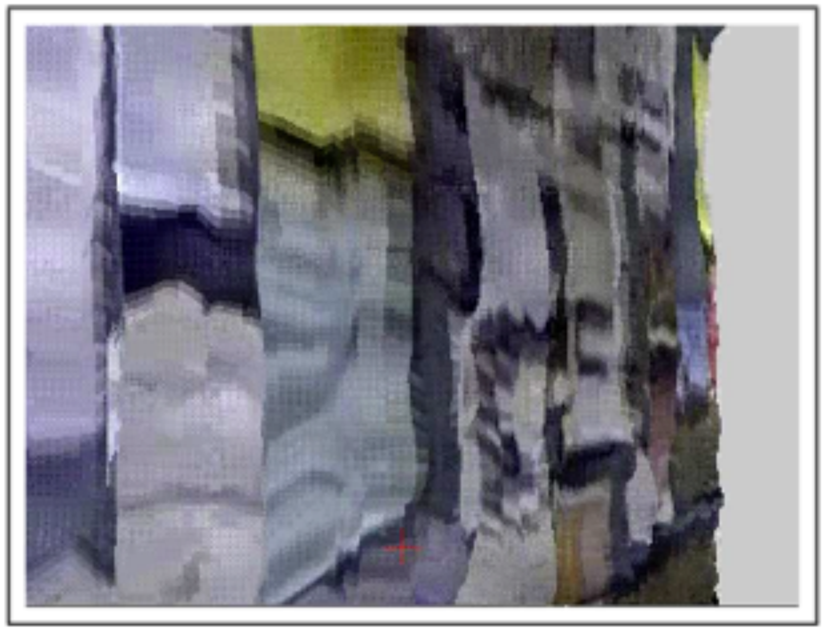
- Has been done in this paper
- Given a set of 3D points, fit planes into the data
- Idea: Model parameters θ are normal vectors and distance to origin for a set of planes
- Gaussian noise model: $p(z \mid \theta) = \mathcal{N}(\underset{\substack{\uparrow \\ \text{point-to-plane} \\ \text{distance}}}{d}(\mathbf{z}, \theta) \mid 0, \underset{\substack{\uparrow \\ \text{noise} \\ \text{variance}}}{\sigma})$
- Introduce latent correspondence variables C_{ij} and maximize the expected log-lik.:

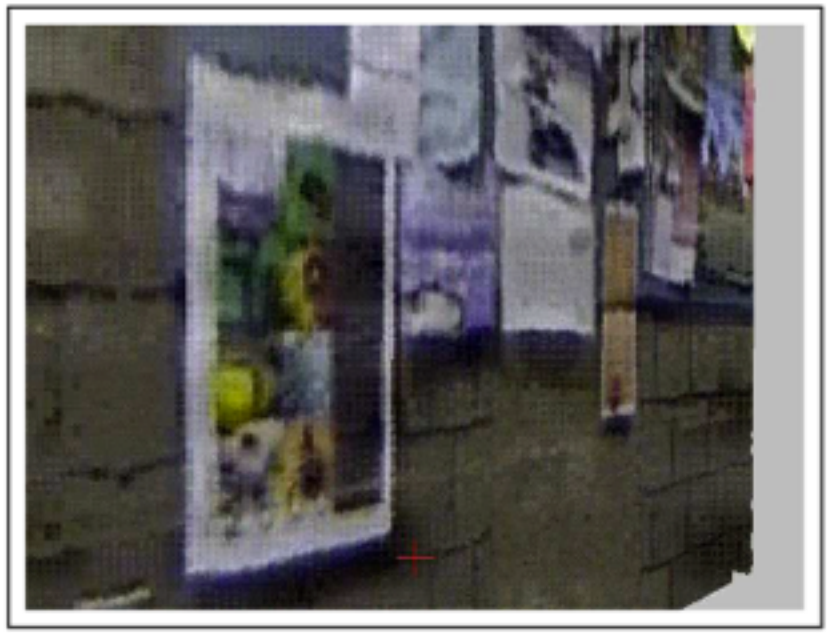
$$\mathbb{E}[\log p(Z, C \mid \theta)]$$

- Maximization can be done in closed form

Example 3: Plane Fitting in 3D







Affinity Propagation

- Often, we are only given a **similarity matrix for** the data points
- The idea of Affinity Propagation is to determine cluster centers (“exemplars”) that explain other data points in an optimal way
- This is similar to k-medoids, but the algorithm is more robust against local minima
- **Idea:** each data point must choose another data point as its exemplar; some points will choose themselves as exemplar
- The number of clusters is then found automatically

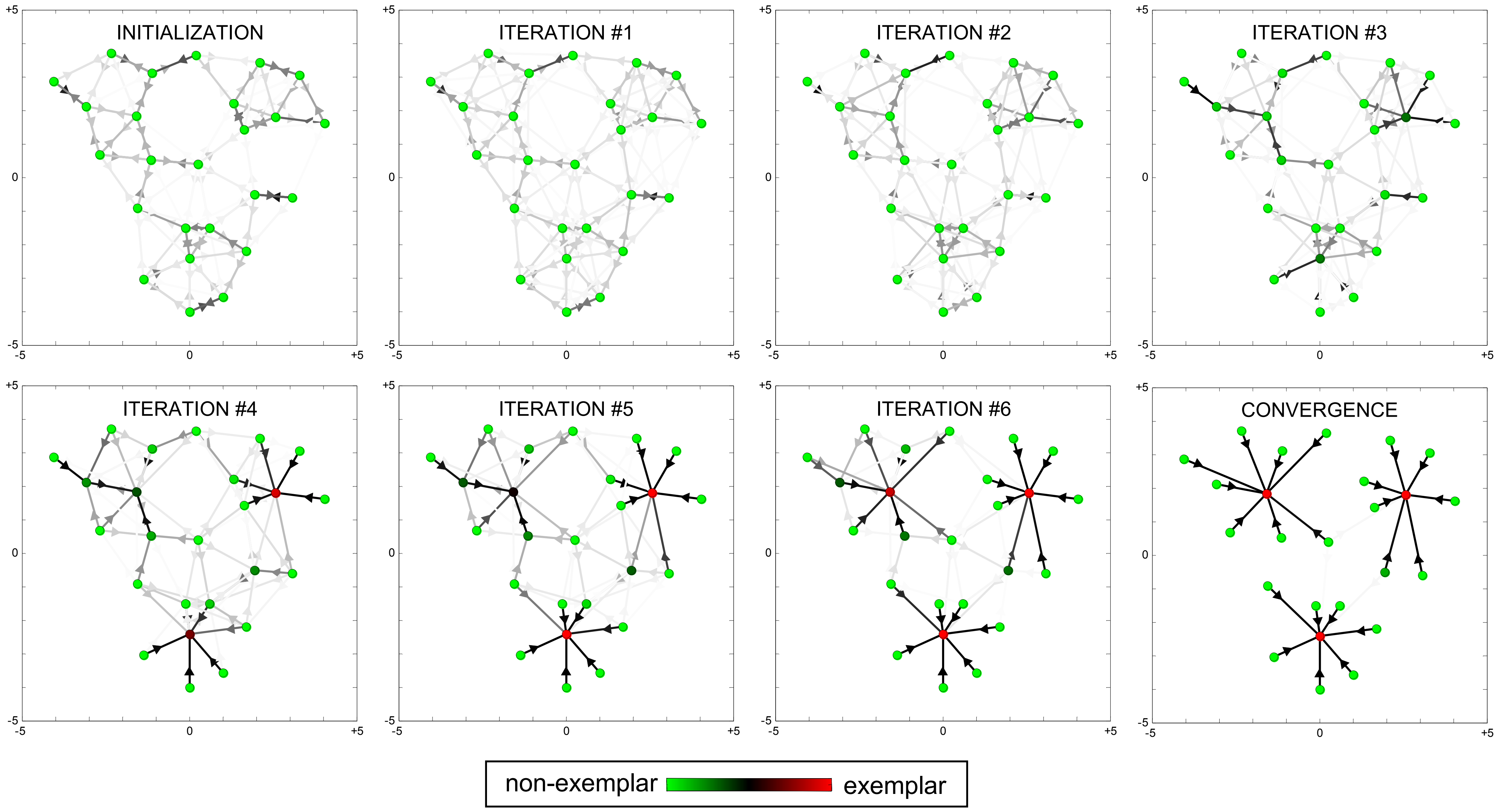
Affinity Propagation

- Input: similarity values s(i,j)
- Initialize the responsibilities r(i,j), and the availabilities a(i,j) to 0
- do until convergence:
  - recompute the responsibilities:
    $$r(i,j) = s(i,j) - \max_{j' \neq j}\{a(i,j') + s(i,j')\}$$
  - recompute the availabilities:
    $$a(i,j) = \min\left\{0, r(j,j) + \sum_{i' \notin \{i,j\}} \max\{0, r(i',j)\}\right\}$$
- the j that maximizes r(i,j) + a(i,j) is the exemplar of i

Affinity Propagation

- Intuitively:
    - responsibility measures how much i thinks that j would be a good exemplar
    - availability measures how strongly j things it should be an exemplar for i
- The algorithm can be shown to be equivalent to max-product loopy belief propagation
- Convergence is not guaranteed, but with “damping” oscillations can be avoided
- The number of clusters can be controlled by the “self-similarity”

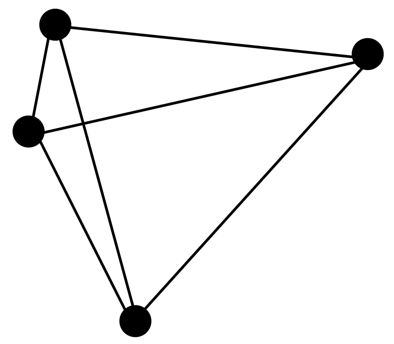
Affinity Propagation



- Colours: how much each point wants to be an exemplar
- Edge strengths: how much a point wants to belong to a cluster

Spectral Clustering

- Consider an undirected graph that connects all data points
- The edge weights are the similarities (“closeness”)
- We define the weighted degree d_i of a node as the sum of all outgoing edges



$$W = \begin{array}{|c|c|c|c|}\hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array}$$

$$d_i = \sum_{j=1}^{N} w_{ij}$$

$$D = \begin{array}{|c|c|c|c|}\hline d_1 & & & \\ \hline & d_2 & & \\ \hline & & d_3 & \\ \hline & & & d_4 \\ \hline \end{array}$$

Spectral Clustering

- The Graph Laplacian is defined as:

$$L = D - W$$

- This matrix has the following properties:
  - the 1 vector is eigenvector with eigenvalue 0

Spectral Clustering

- The Graph Laplacian is defined as:

$$L = D - W$$

- This matrix has the following properties:
  - the 1 vector is eigenvector with eigenvector 0
  - the matrix is symmetric and positive semi-definite

Spectral Clustering

- The Graph Laplacian is defined as:

$$L = D - W$$

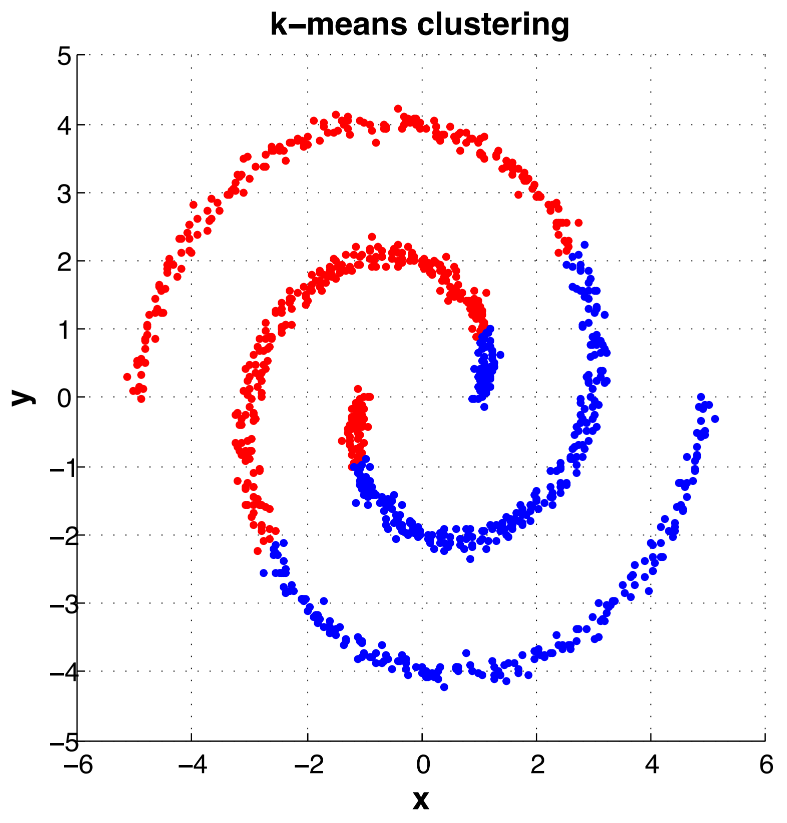
- This matrix has the following properties:
  - the 1 vector is eigenvector with eigenvector 0
  - the matrix is symmetric and positive semi-definite
- With these properties we can show:

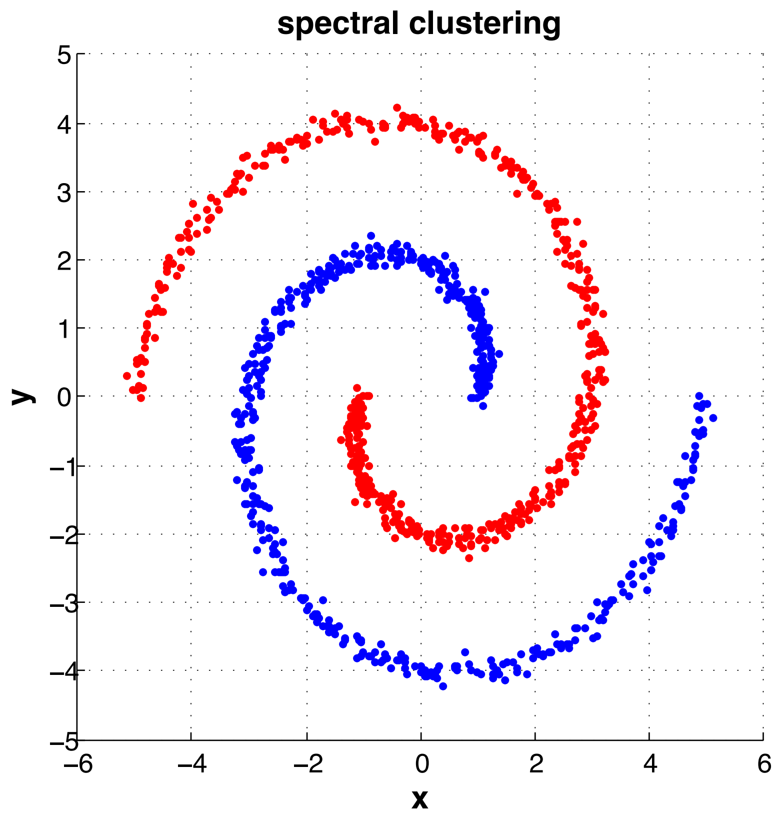
Theorem: The set of eigenvectors of L with eigenvalue 0 is spanned by the indicator vectors $\mathbf{1}_{A_1}, \ldots, \mathbf{1}_{A_K}$, where A_k are the K connected components of the graph.

The Algorithm

- Input: Similarity matrix W
- Compute L = D - W
- Compute the eigenvectors that correspond to the K smallest eigenvalues
- Stack these vectors as columns in a matrix U
- Treat each row of U as a K-dim data point
- Cluster the N rows with K-means clustering
- The indices of the rows that correspond to the resulting clusters are those of the original data points.

An Example





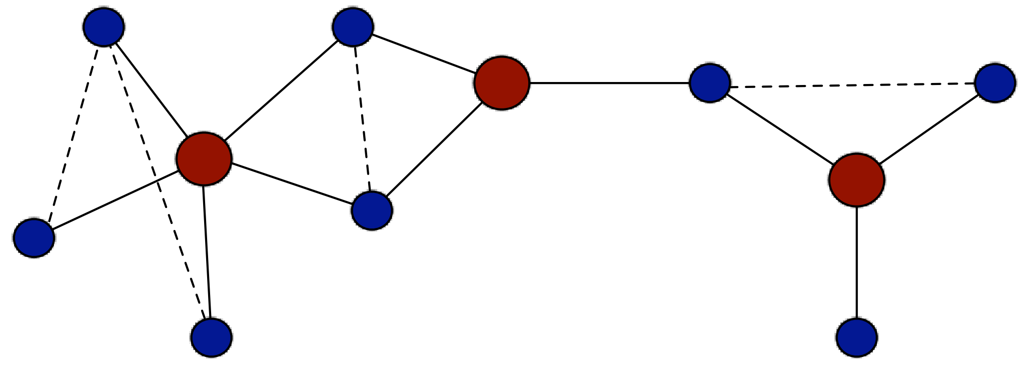
- Spectral clustering can handle complex problems such as this one
- The complexity of the algorithm is $O(N^3)$, because it has to solve an eigenvector problem
- But there are efficient variants of the algorithm

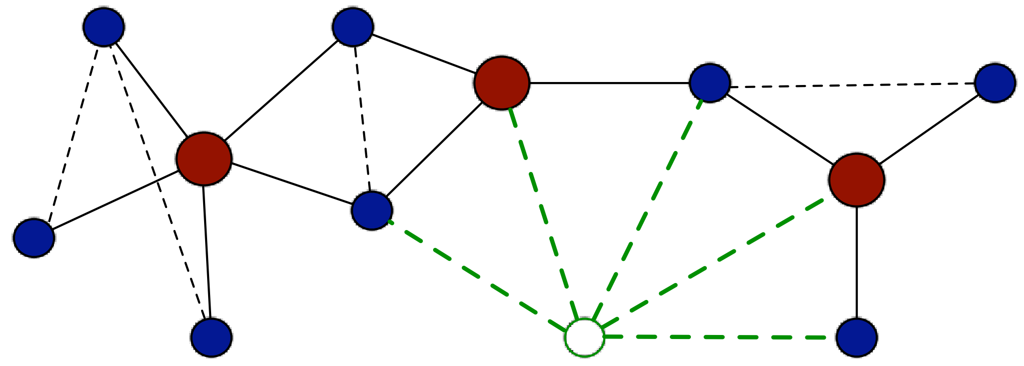
Further Remarks

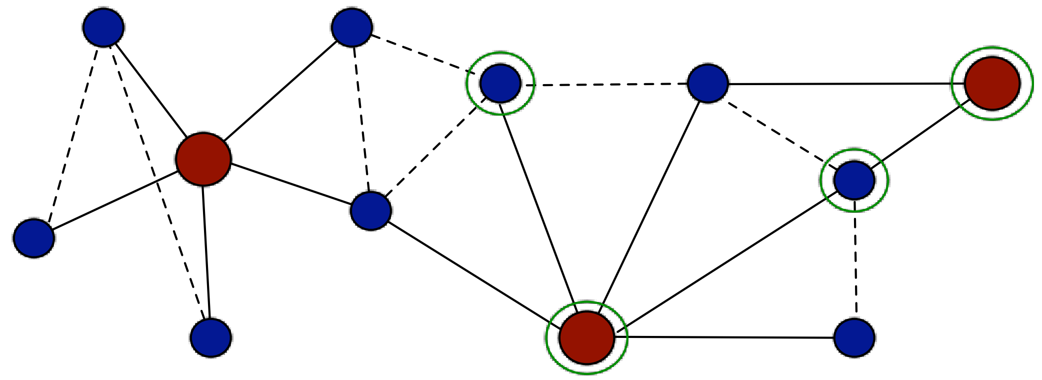
- To account for nodes that are highly connected, we can use a normalized version of the graph Laplacian
- Two different methods exist:
  - $L_{rw} = D^{-1}L = I - D^{-1}W$
  - $L_{sym} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$
- These have similar eigenspaces than the original Laplacian L
- Clustering results tend to be better than with the unnormalized Laplacian

Online Star Clustering

- clusters consist of **centers** and **satellites**, connected to each other by edges
- **normalized cosine distance** is used to compute the similarities between features
- number of clusters is inferred automatically and depends on a **similarity threshold** σ
- new elements are inserted **incrementally** without rearranging the entire data structure
- insertion time is asymptotically **linear** in the size of the graph
- star-subgraph geometry ensures high expected
  satellite similarity, implying dense clustering


star cluster graph

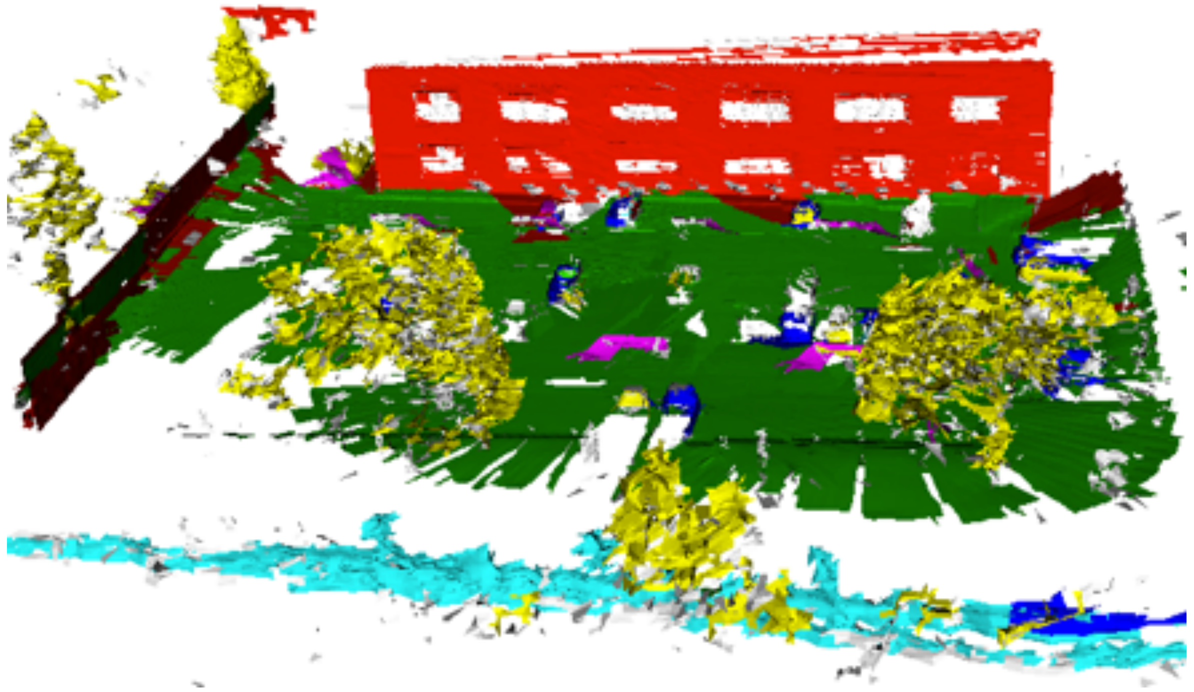

after insertion


new cluster centers

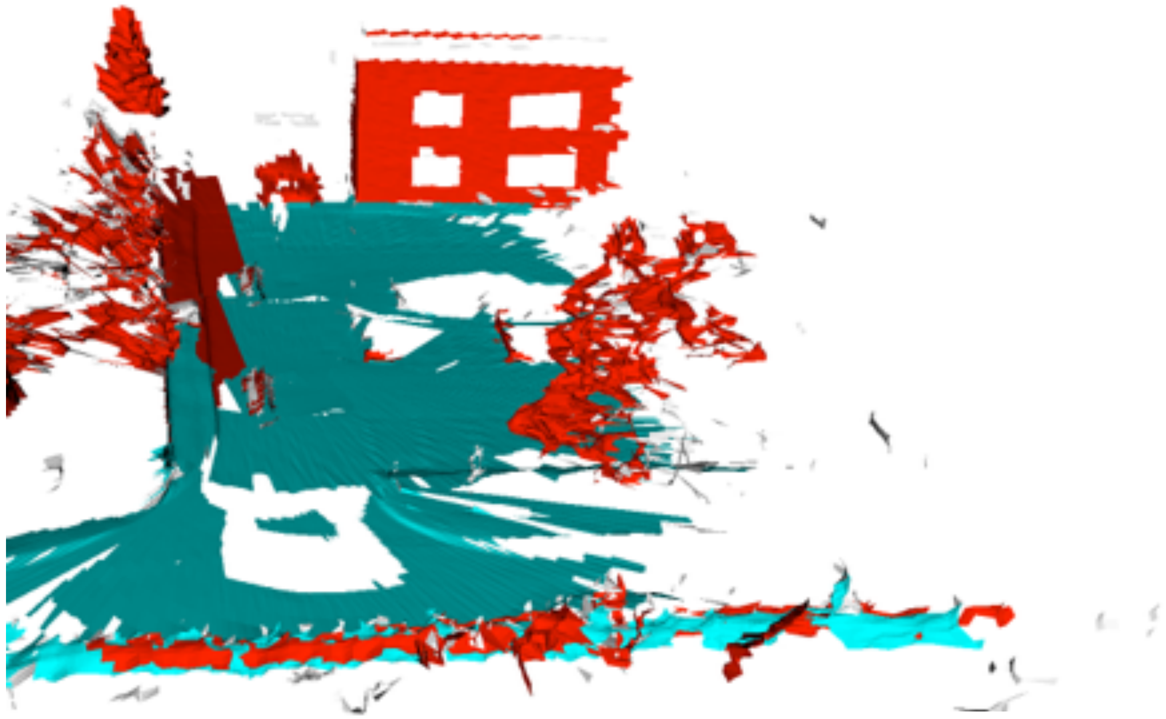
Example 4: Online Scene Labeling

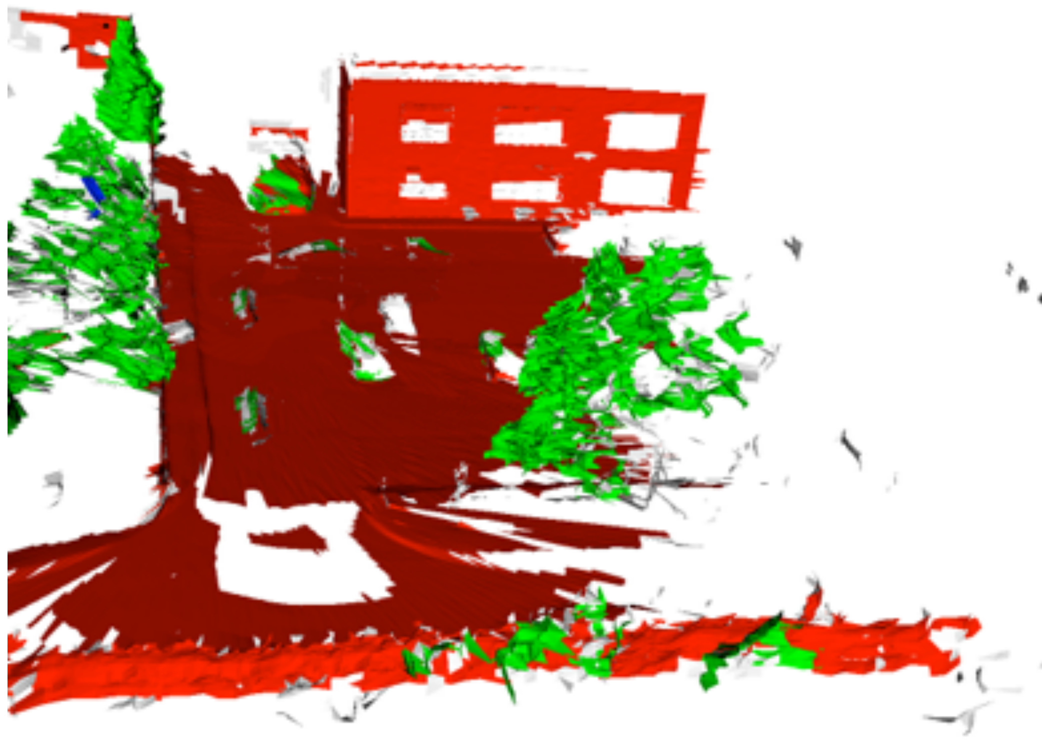
Given: 3D Point Cloud Data

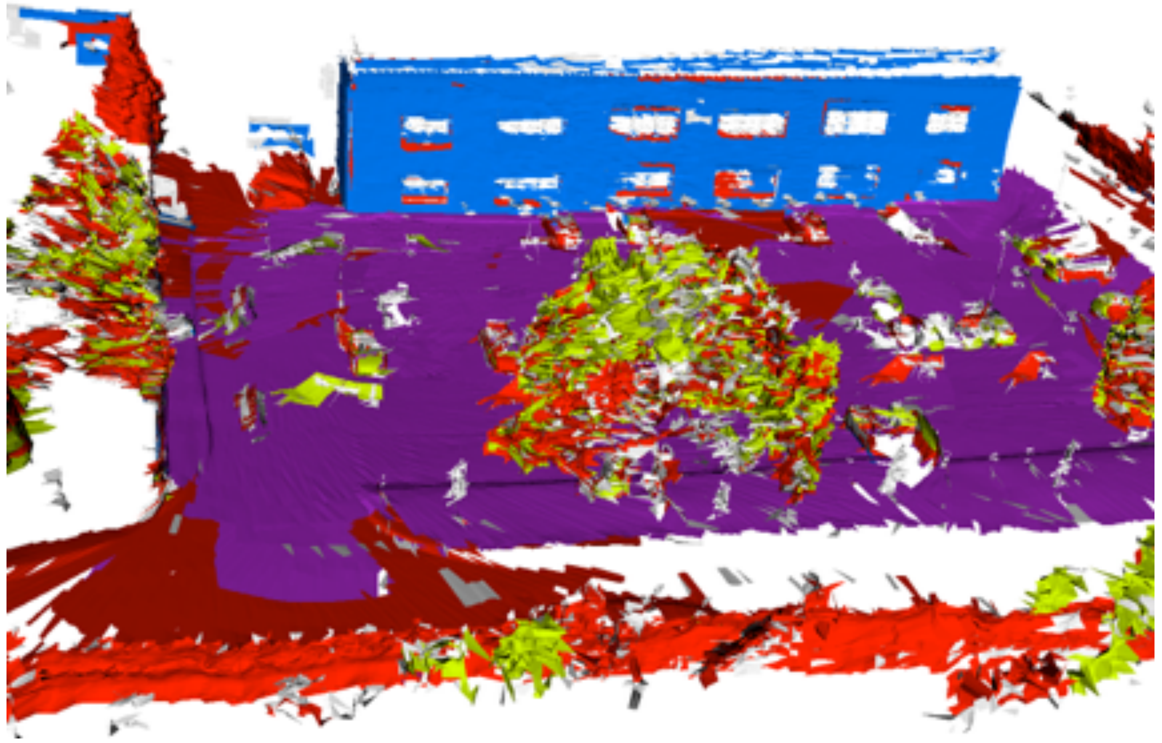
Aim: Clustering



Example 4: Online Scene classification


after 2 point clouds: 2 discovered clusters


after 4 point clouds: 3 discovered clusters

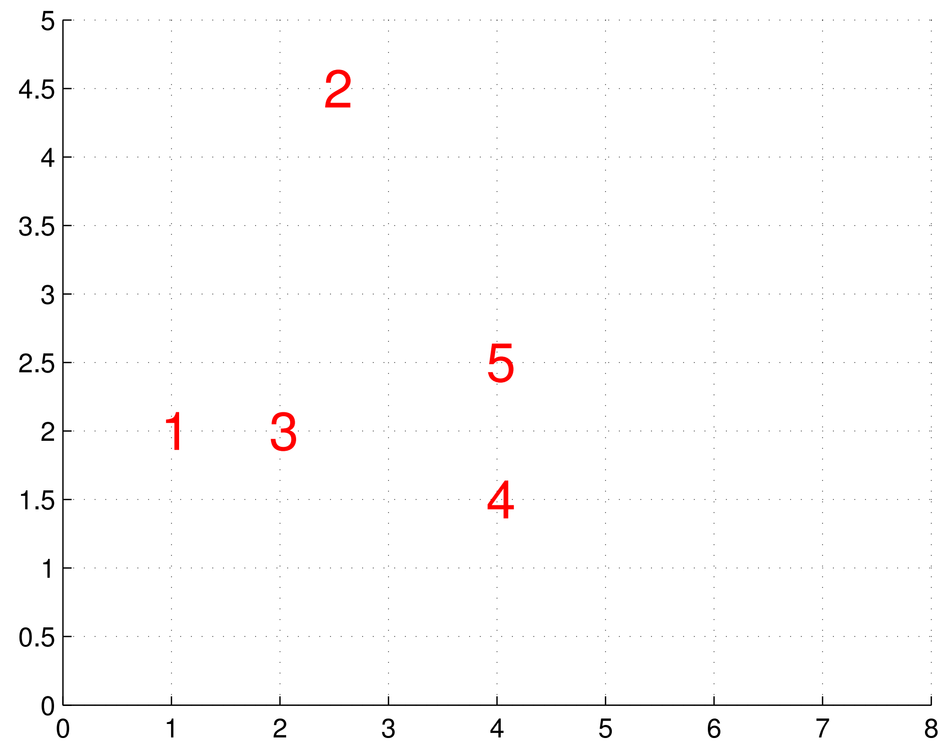

after 17 point clouds: 6 discovered clusters

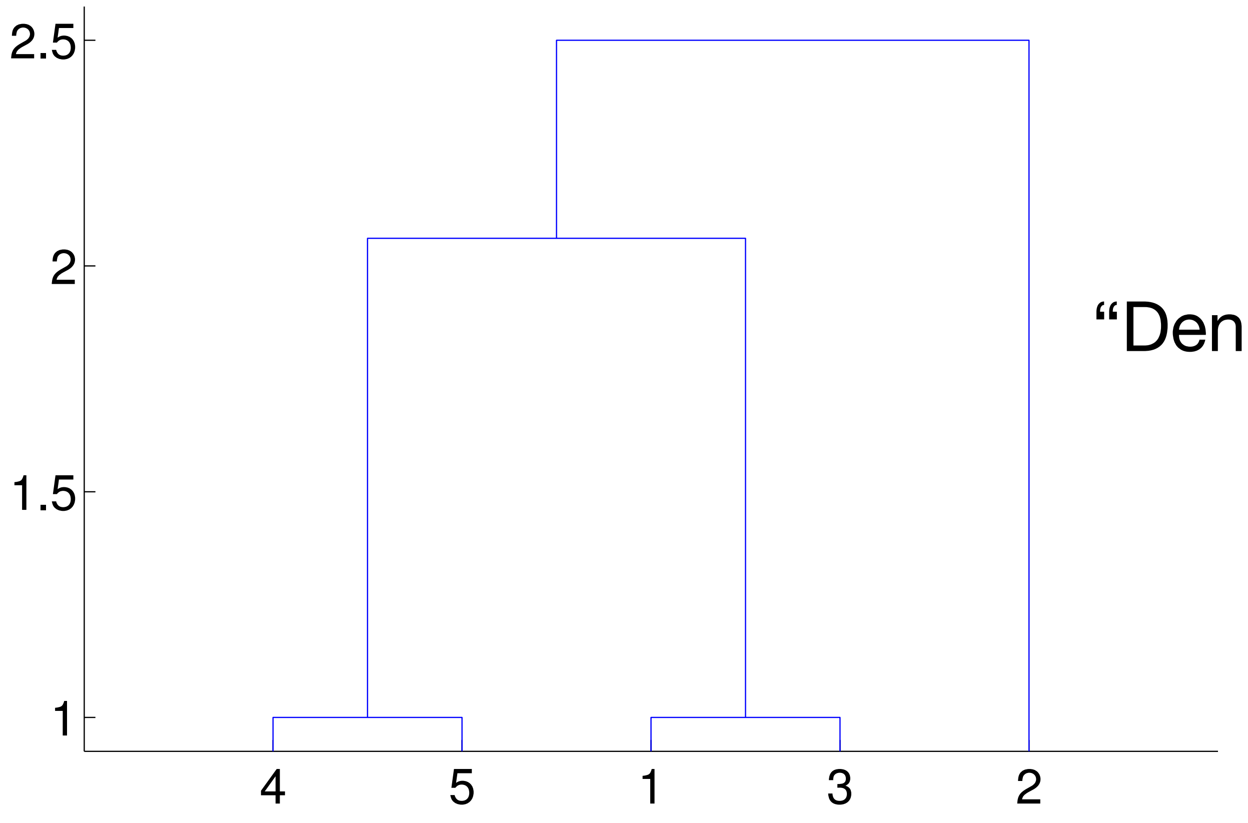
Hierarchical Clustering

- Often, we want to have nested clusters instead of a “flat” clustering
- Two possible methods:
  - “bottom-up” or agglomerative clustering
  - “top-down” or divisive clustering
- Both methods take a dissimilarity matrix as input
- Bottom-up grows merges points to clusters
- Top-down splits clusters into sub-clusters
- Both are heuristics, there is no clear objective function
- They always produce a clustering (also for noise)

Agglomerative Clustering

- Start with N clusters, each contains exactly one data point
- At each step, merge the two most similar groups
- Repeat until there is a single group





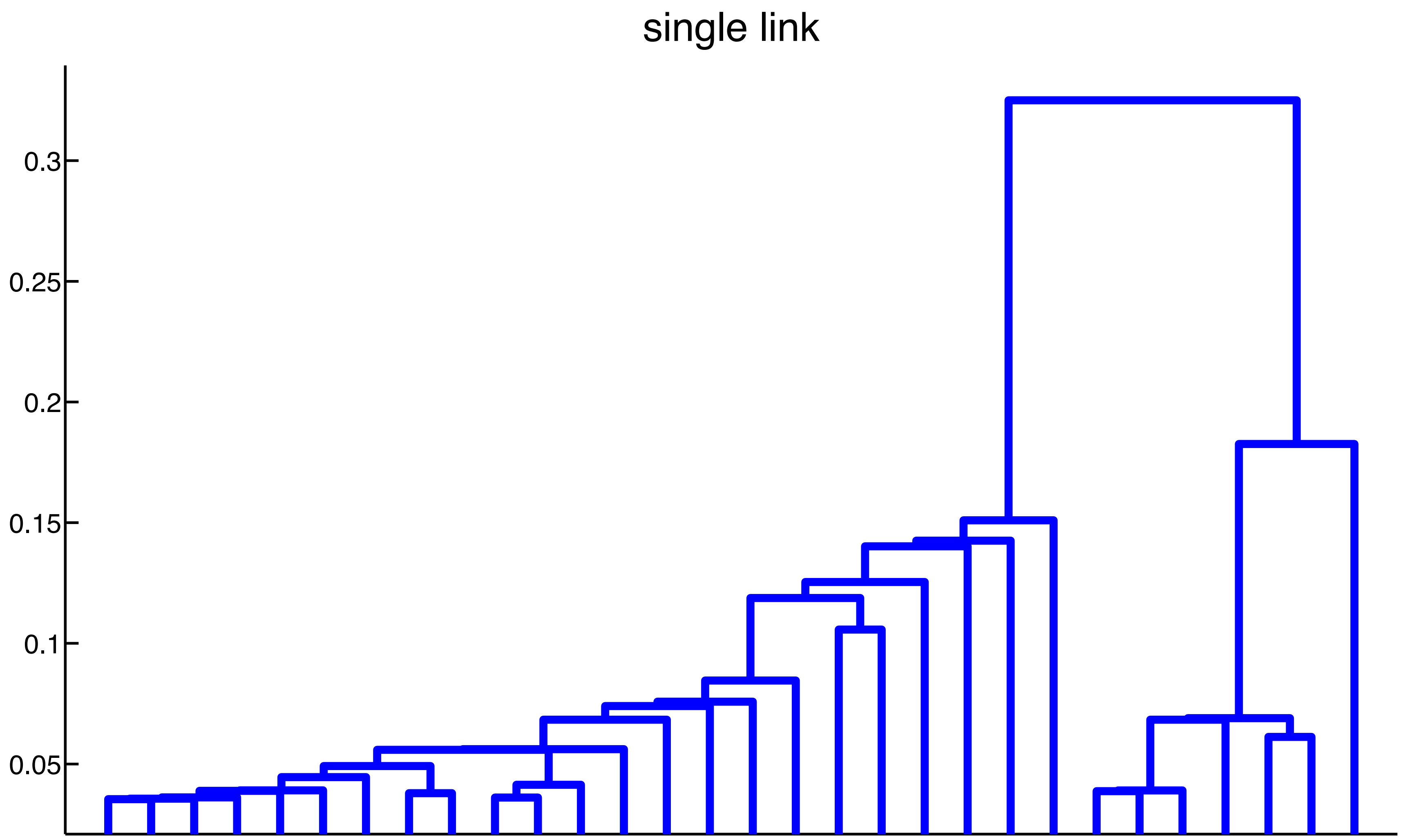
"Dendrogram"

Linkage

- In agglomerative clustering, it is important to define a distance measure between two clusters
- There are three different methods:
  - Single linkage: considers the two closest elements from both clusters and uses their distance
  - Complete linkage: considers the two farthest elements from both clusters
  - Average linkage: uses the average distance between pairs of points from both clusters
- Depending on the application, one linkage should be preferred over the other

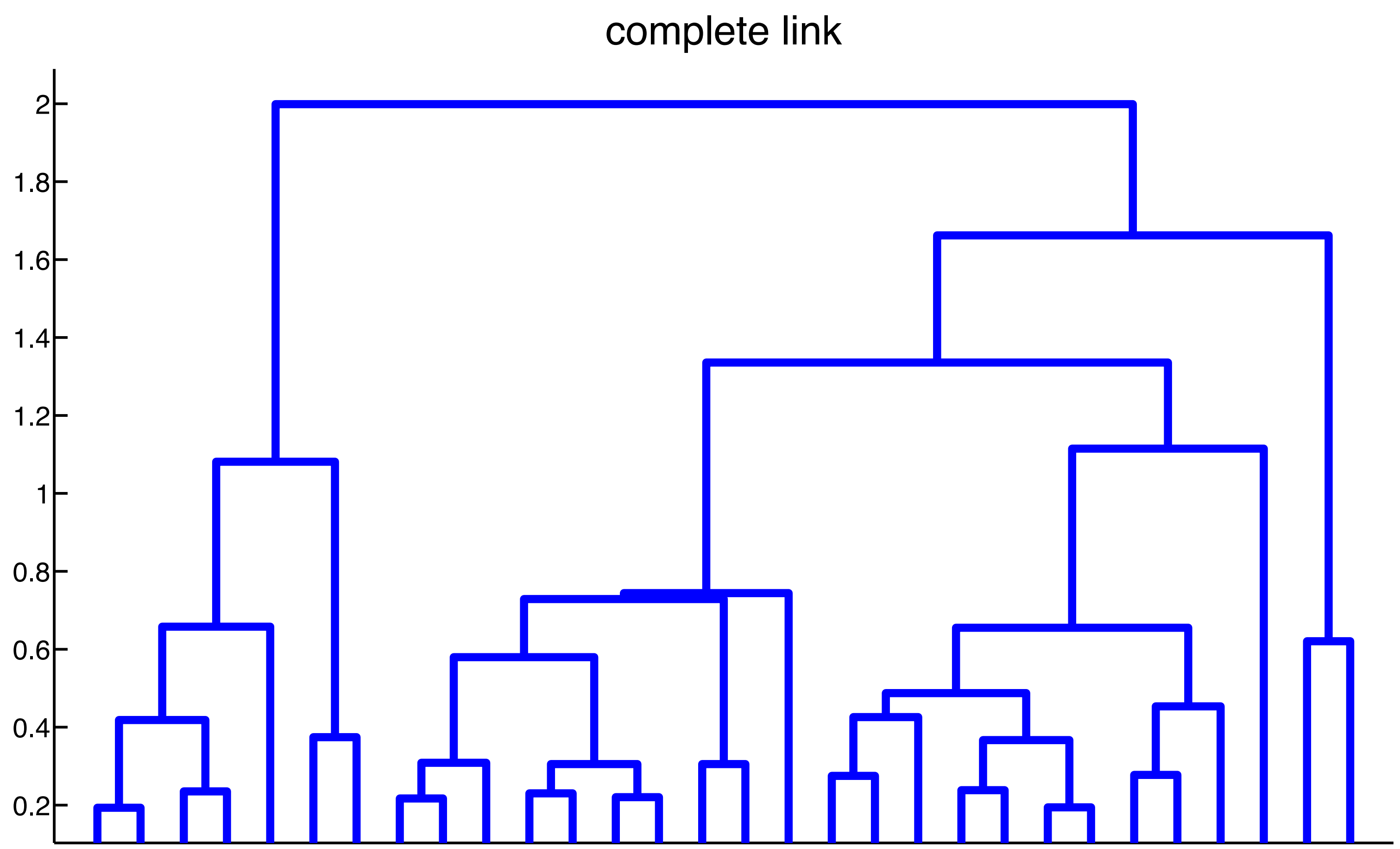
Single Linkage

- The distance is based on $d_{SL}(G,H) = \min_{i \in G, i' \in H} d_{i,i'}$
- The resulting dendrogram is a minimum spanning tree, i.e. it minimizes the sum of the edge weights
- Thus: we can compute the clustering in O(N^2) time



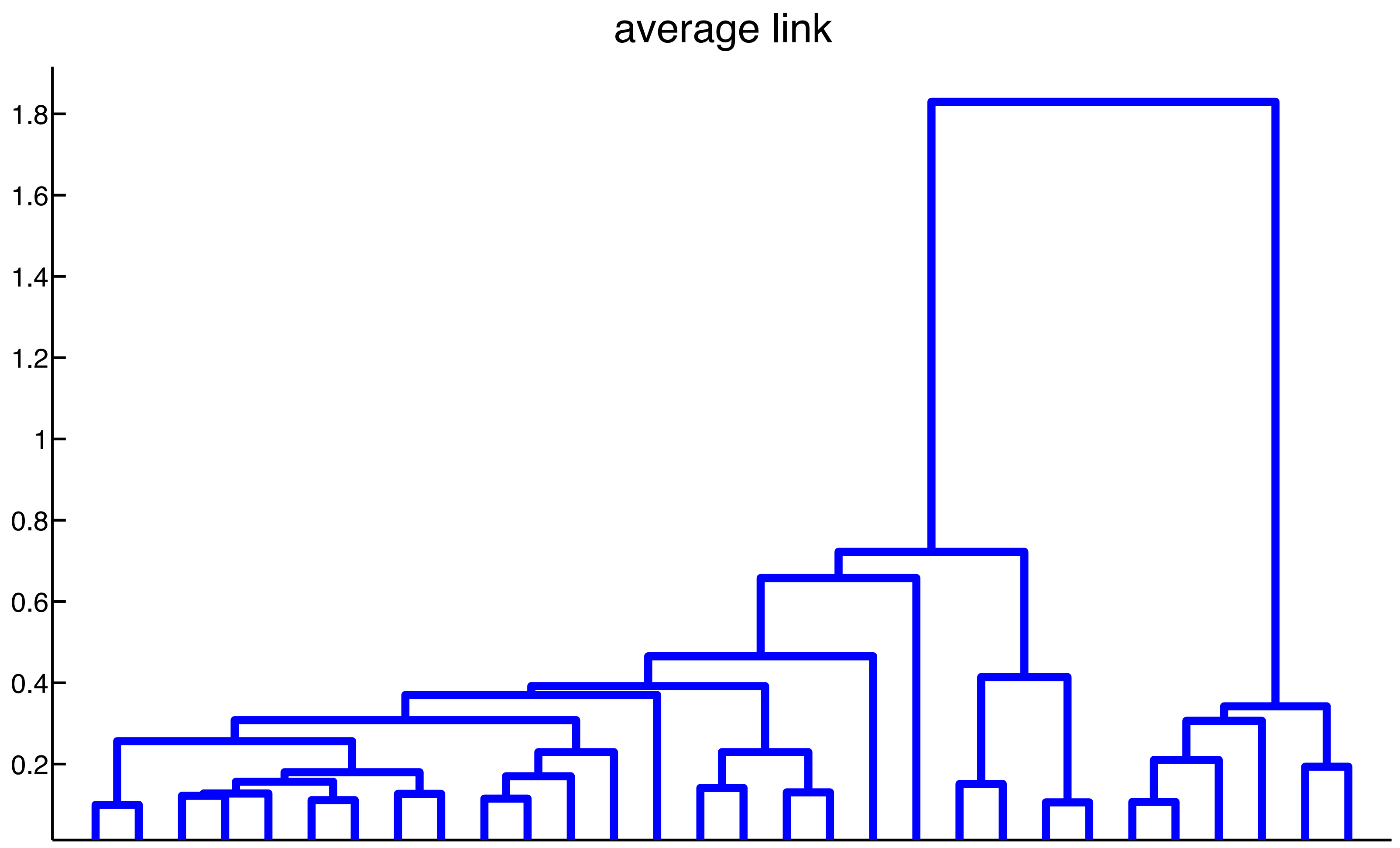
Complete Linkage

- The distance is based on $d_{CL}(G,H) = \max_{i \in G, i' \in H} d_{i,i'}$
- Complete linkage fulfills the **compactness property**, i.e. all points in a group should be similar to each other
- Tends to produce clusters with smaller diameter



Average Linkage

- The distance is based on $d_{avg}(G,H) = \frac{1}{n_G n_H} \sum_{i \in G} \sum_{i' \in H} d_{i,i'}$
- Is a good compromise between single and complete linkage
- However: sensitive to changes on the meas. scale



Divisive Clustering

- Start with all data in a single cluster
- Recursively divide each cluster into two child clusters
- Problem: optimal split is hard to find
- Idea: use the cluster with the largest diameter and use K-means with K = 2
- Or: use minimum-spanning tree and cut links with the largest dissimilarity
- In general two advantages:
  - Can be faster
  - More globally informed (not myopic as bottom-up)

Choosing the Number of Clusters

- As in general, choosing the number of clusters is hard
- When a dendrogram is available, a gap can be detected in the lengths of the links
- This represents the dissimilarity between merged groups
- However: in real data this can be hard to detect
- There are Bayesian techniques to address this problem (Bayesian hierarchical clustering)

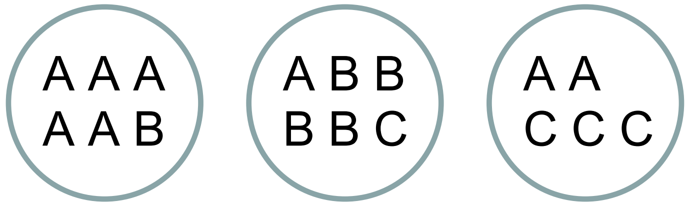
Evaluation of Clustering Algorithms

- Clustering is unsupervised: evaluation of the output is hard, because no ground truth is given
- Intuitively, points in a cluster should be similar and points in different clusters dissimilar
- However, better methods use external information, such as labels or a reference clustering
- Then we can compare clusterings with the labels using different metrics, e.g.
  - purity
  - mutual information

Purity

- Define N_{ij} the number of objects in cluster i that are in class j
- Define $N_i = \sum_{j=1}^{C} N_{ij}$ number of objects in cluster i
- $p_{ij} = \frac{N_{ij}}{N_i}$ $\quad p_i = \max_j p_{ij}$ "Purity"
- overall purity

$$\sum_i \frac{N_i}{N} p_i$$



Purity = 0.71

- Purity ranges from 0 (bad) to 1 (good)
- But: a clustering with each object in its own cluster has a purity of 1

Mutual Information

- Let U and V be two clusterings
- Define the probability that a randomly chosen point belongs to cluster u_i in U and to v_j in V

$$p_{UV}(i,j) = \frac{|u_i \cap v_j|}{N}$$

- Also: The prob. that a point is in u_i $\quad p_U(i) = \frac{|u_i|}{N}$

$$\mathbb{I}(U,V) = \sum_{i=1}^{R} \sum_{j=1}^{C} p_{UV}(i,j) \log \frac{p_{UV}(i,j)}{p_U(i) p_V(j)}$$

- This can be normalized to account for many small clusters with low entropy

Summary

- Several Clustering methods exist:
  - K-means clustering and Expectation-Maximization, both based on Gaussian Mixture Models
  - K-means uses hard assignments, whereas EM uses soft assignments and estimates also the covariances
  - Spectral clustering uses the graph Laplacian and performs an eigenvector analysis
- Major Problem:
  - most clustering algorithms require the number of clusters to be given

4. Kernel Methods

Motivation

- Usually learning algorithms assume that some kind of feature function is given
- Reasoning is then done on a feature vector of a given (finite) length
- But: some objects are hard to represent with a fixed-size feature vector, e.g. text documents, molecular structures, evolutionary trees
- Idea: use a way of measuring similarity without the need of features, e.g. the edit distance for strings
- This we will call a **kernel function**

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T KK\mathbf{a} - \mathbf{a}^T K\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T K\mathbf{a}$$

This is called the **dual formulation.**

The solution to the dual problem is:

$$\mathbf{a} = (K + \lambda I_N)^{-1}\mathbf{t}$$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T K K\mathbf{a} - \mathbf{a}^T K\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T K\mathbf{a}$$

$$\mathbf{a} = (K + \lambda I_N)^{-1}\mathbf{t}$$

This we can use to make **predictions**:

$$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \mathbf{a}^T\Phi\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(K + \lambda I_N)^{-1}\mathbf{t}$$

(now $\mathbf{x}$ is unknown and $\mathbf{a}$ is given from training)

Dual Representation

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T (K + \lambda I_N)^{-1} \mathbf{t}$$

where:

$$\mathbf{k}(\mathbf{x}) = \begin{pmatrix} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}) \\ \vdots \\ \phi(\mathbf{x}_N)^T \phi(\mathbf{x}) \end{pmatrix} \quad K = \begin{pmatrix} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_N) \end{pmatrix}$$

Thus, y is expressed only in terms of **dot products** between different pairs of $\phi(\mathbf{x})$, or in terms of the **kernel function**

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Representation using the Kernel

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T (K + \lambda I_N)^{-1} \mathbf{t}$$

Now we have to invert a matrix of size $N \times N$, before it was $M \times M$ where $M < N$, but:

By expressing everything with the kernel function, we can deal with very high-dimensional or even **infinite**-dimensional feature spaces!

Idea: Don’t use features at all but simply define a similarity function expressed as the kernel!

Constructing Kernels

The straightforward way to define a kernel function is to first find a basis function $\phi(\mathbf{x})$ and to define:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

This means, k is an inner product in some space $\mathcal{H}$, i.e:

1. Symmetry: $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_j), \phi(\mathbf{x}_i) \rangle = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$
2. Linearity: $\langle a(\phi(\mathbf{x}_i) + \mathbf{z}), \phi(\mathbf{x}_j) \rangle = a \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle + a \langle \mathbf{z}, \phi(\mathbf{x}_j) \rangle$
3. Positive definite: $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_i) \rangle \geq 0$, equal if $\phi(\mathbf{x}_i) = \mathbf{0}$

Can we find conditions for k under which there is a (possibly infinite dimensional) basis function into $\mathcal{H}$, where k is an inner product?

Constructing Kernels

Theorem (Mercer): If k is

1. symmetric, i.e. $k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_j, \mathbf{x}_i)$ and
2. positive definite, i.e.

$$K = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

"Gram Matrix"

is positive definite, then there exists a mapping $\phi(\mathbf{x})$ into a feature space $\mathcal{H}$ so that k can be expressed as an inner product in $\mathcal{H}$.

This means, we don't need to find $\phi(\mathbf{x})$ explicitly!

We can directly work with k **"Kernel Trick"**

Application Examples

Kernel Methods can be applied for many different problems, e.g.:

- Density estimation (unsupervised learning)
- Regression
- Principal Component Analysis (PCA)
- Classification

Most important Kernel Methods are

- Support Vector Machines
- Gaussian Processes

Kernelization

- Many existing algorithms can be converted into kernel methods
- This process is called “kernelization”

Idea:

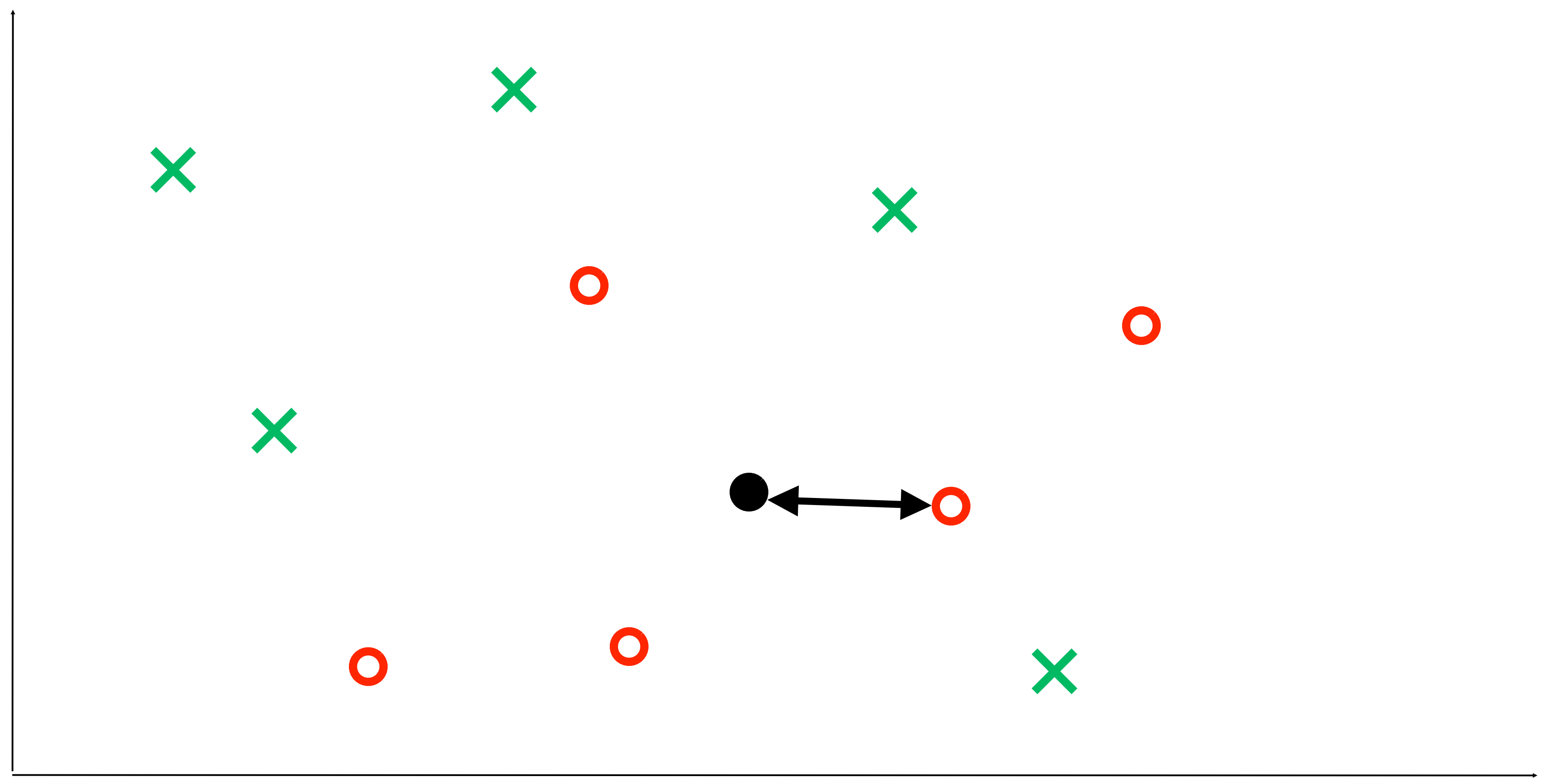
- express similarities of data points in terms of an inner product (dot product)
- replace all occurrences of that inner product by the kernel function

This is called the **kernel trick**

Example: Nearest Neighbor

- The NN classifier selects the label of the nearest neighbor in Euclidean distance

$$\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \mathbf{x}_i^T\mathbf{x}_i + \mathbf{x}_j^T\mathbf{x}_j - 2\mathbf{x}_i^T\mathbf{x}_j$$



Example: Nearest Neighbor

- The NN classifier selects the label of the nearest neighbor in Euclidean distance

$$\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \mathbf{x}_i^T \mathbf{x}_i + \mathbf{x}_j^T \mathbf{x}_j - 2\mathbf{x}_i^T \mathbf{x}_j$$

- We can now replace the dot products by a valid Mercer kernel and we obtain:

$$d(\mathbf{x}_i, \mathbf{x}_j)^2 = k(\mathbf{x}_i, \mathbf{x}_i) + k(\mathbf{x}_j, \mathbf{x}_j) - 2k(\mathbf{x}_i, \mathbf{x}_j)$$

- This is a **kernelized** nearest-neighbor classifier
- We do not explicitly compute feature vectors!

Back to Linear Regression

We had the primal and the dual formulation:

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T KK\mathbf{a} - \mathbf{a}^T K\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T K\mathbf{a}$$

with the dual solution:

$$\mathbf{a} = (K + \lambda I_N)^{-1}\mathbf{t}$$

This we can use to make **predictions (MAP)**:

$$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \mathbf{a}^T\Phi\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(K + \lambda I_N)^{-1}\mathbf{t}$$

Observations

- We have found a way to predict function values of y for new input points $\mathbf{x}$
- As we used regularized regression, we can equivalently find the **predictive distribution** by marginalizing out the parameters $\mathbf{w}$

Questions:

- Can we find a closed form for that distribution?
- How can we model the uncertainty of our prediction?
- Can we use that for classification?

Definition

Definition: A **Gaussian process** is a collection of random variables, any finite number of which have a joint Gaussian distribution.

The number of random variables can be **infinite**!

This means: a GP is a Gaussian distribution over **functions**!

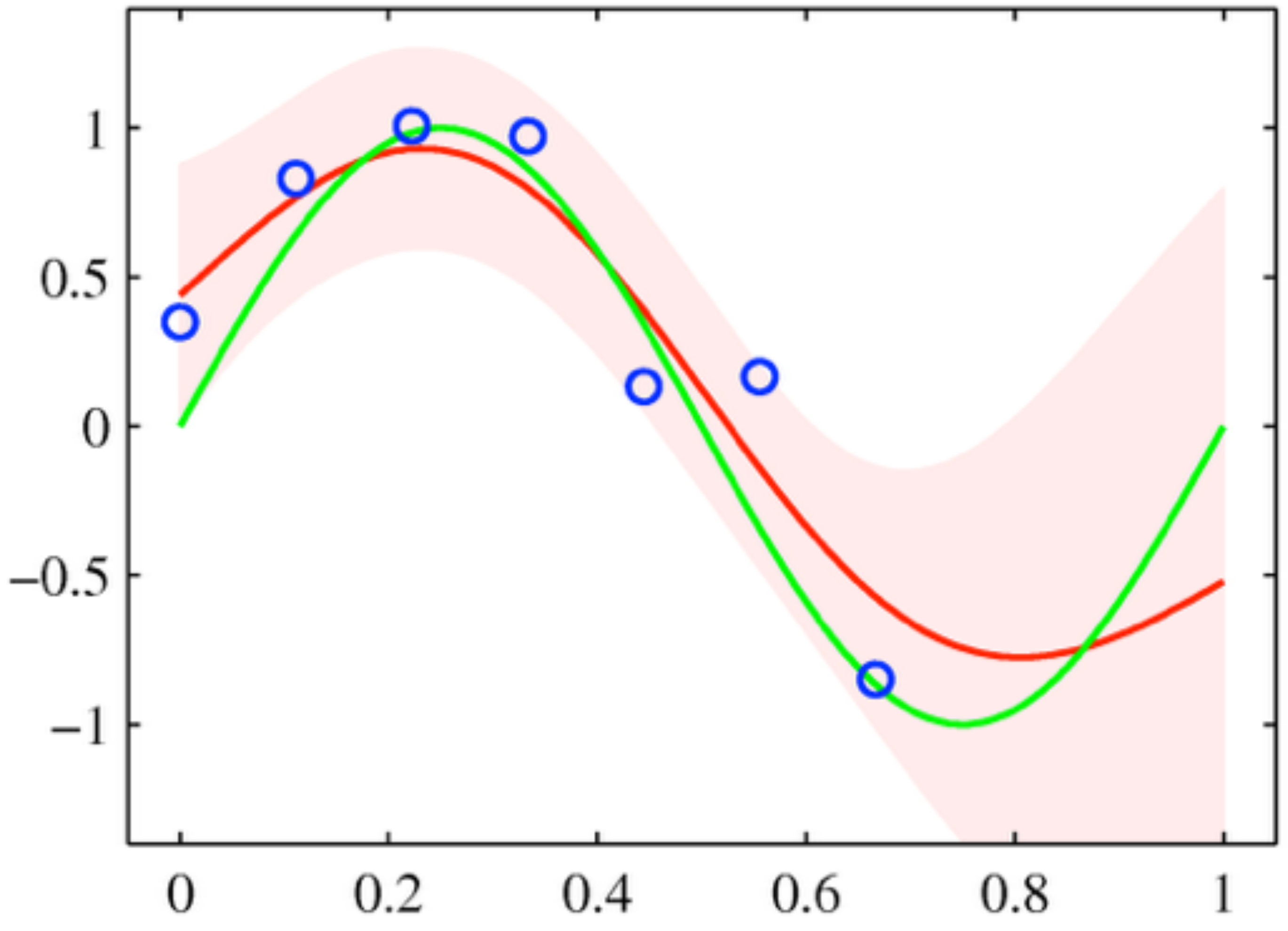
To specify a GP we need:

mean function: $m(\mathbf{x}) = \mathbb{E}[y(\mathbf{x})]$

covariance function:

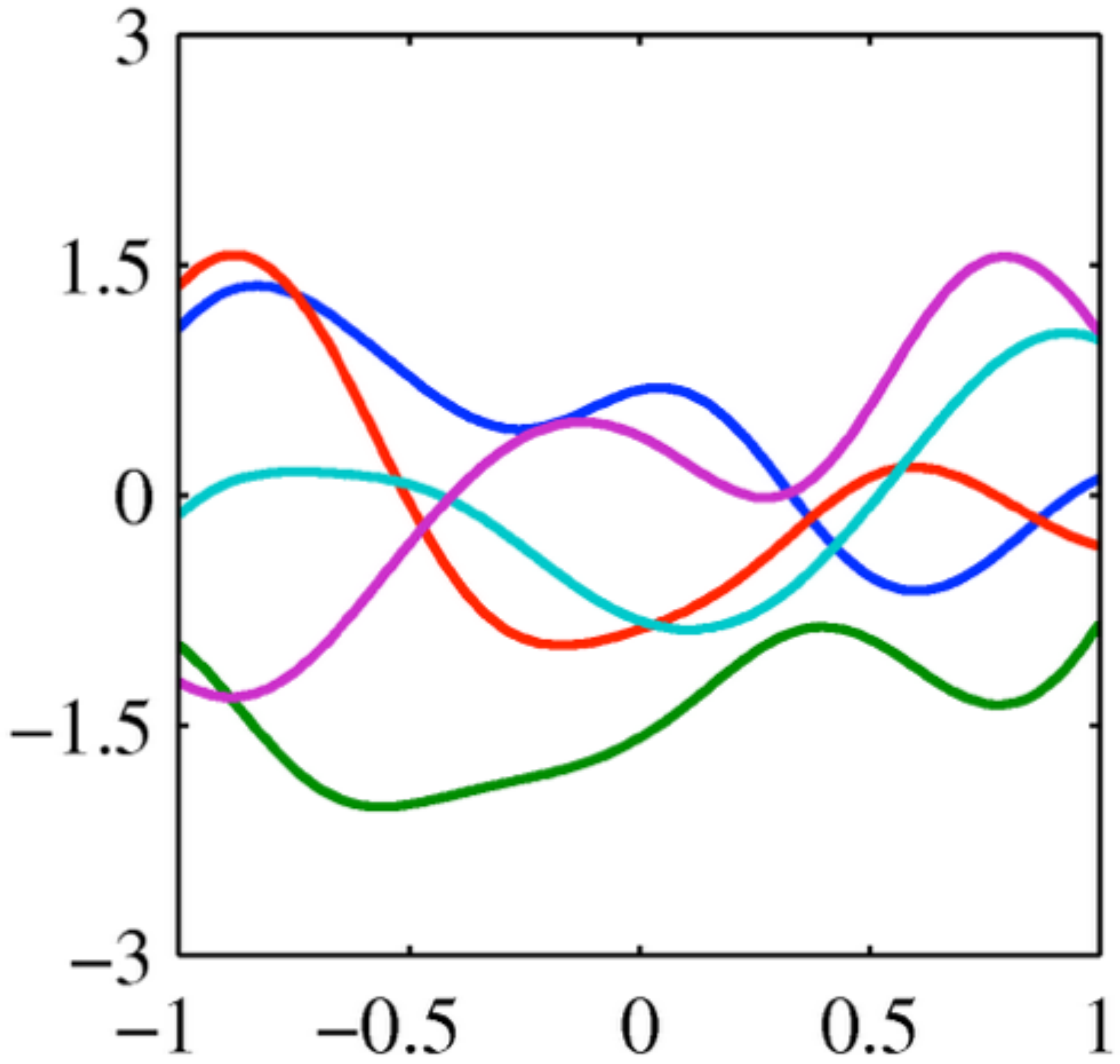
$$k(\mathbf{x}_1, \mathbf{x}_2) = \mathbb{E}[y(\mathbf{x}_1) - m(\mathbf{x}_1)y(\mathbf{x}_2) - m(\mathbf{x}_2)]$$

Example

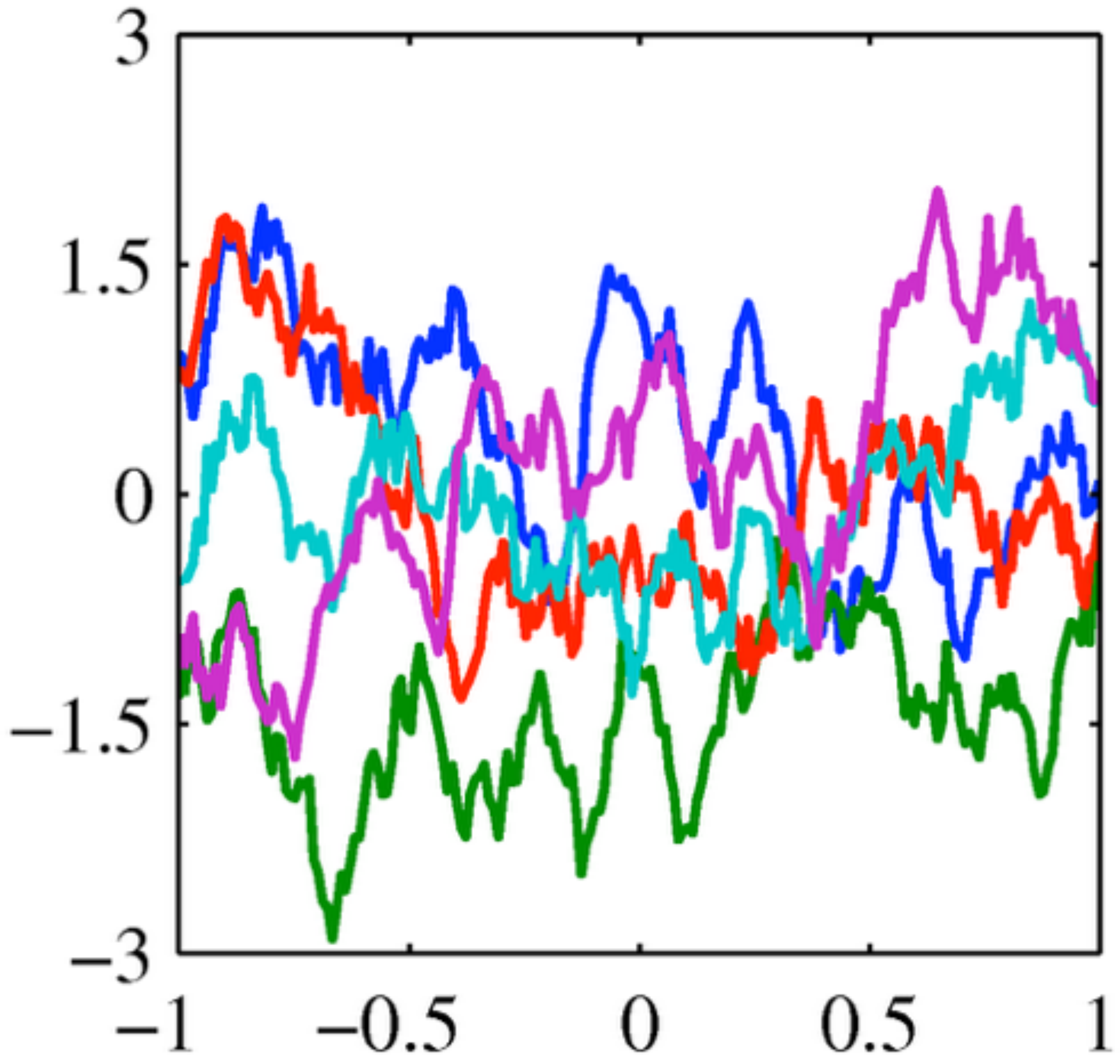


- green line: sinusoidal data source
- blue circles: data points with Gaussian noise
- red line: mean function of the Gaussian process

Sampling from a GP



Squared exponential kernel



Exponential kernel

Prediction with a Gaussian Process

Most often we are more interested in predicting new function values for given input data.

We have:

training data $\mathbf{x}_1, \dots, \mathbf{x}_N \quad y_1, \dots, y_N$

test input $\mathbf{x}_1^*, \dots, \mathbf{x}_M^*$

And we want test outputs $y_1^*, \dots, y_M^*$

The joint probability is

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{y}_* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{pmatrix}\right)$$

and we need to compute $p(\mathbf{y}^* \mid \mathbf{x}^*, X, \mathbf{y})$.

Prediction with a Gaussian Process

In the case of only one test point $\mathbf{x}^*$ we have

$$K(X, \mathbf{x}^*) = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_*) \\ \vdots \\ k(\mathbf{x}_N, \mathbf{x}_*) \end{pmatrix} = \mathbf{k}_*$$

Now we compute the conditional distribution

$$p(y^* \mid \mathbf{x}^*, X, \mathbf{y}) = \mathcal{N}(y_* \mid \mu_*, \Sigma_*)$$

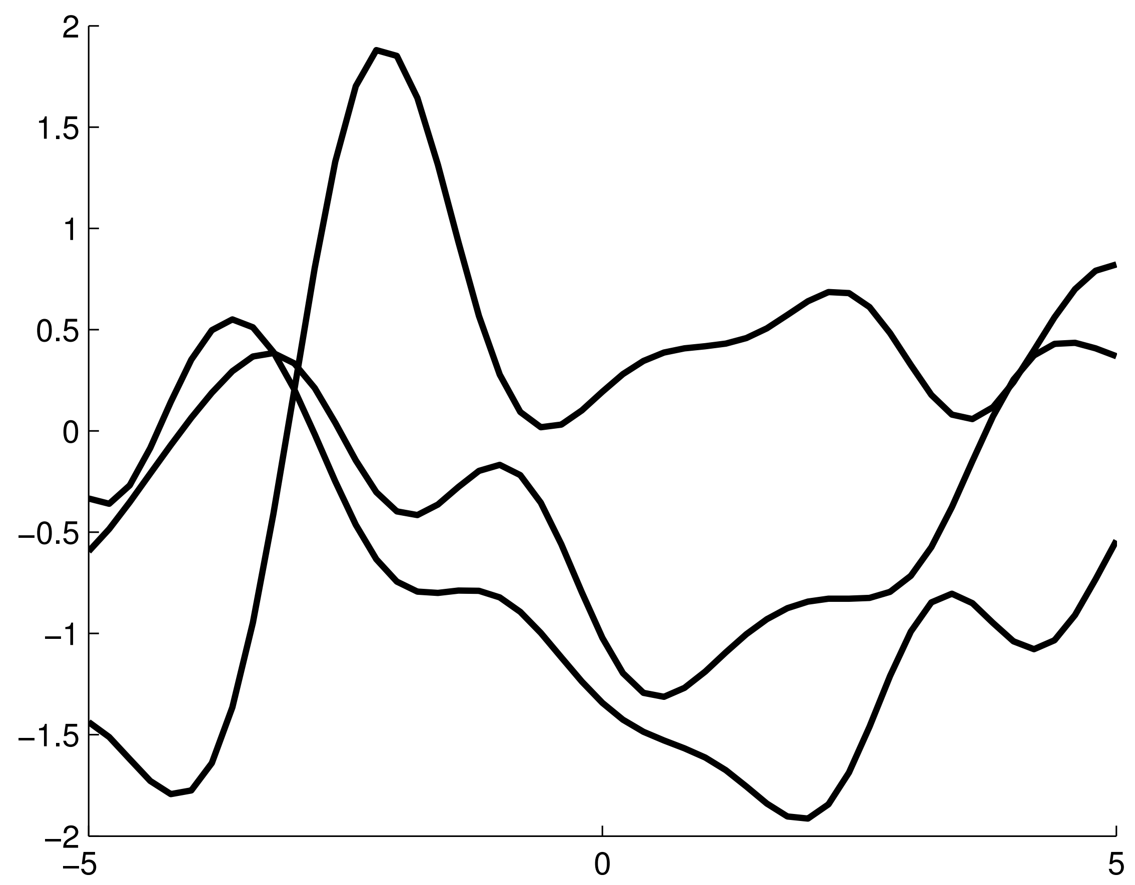
where

$$\mu_* = \mathbf{k}_*^T K^{-1} \mathbf{t}$$

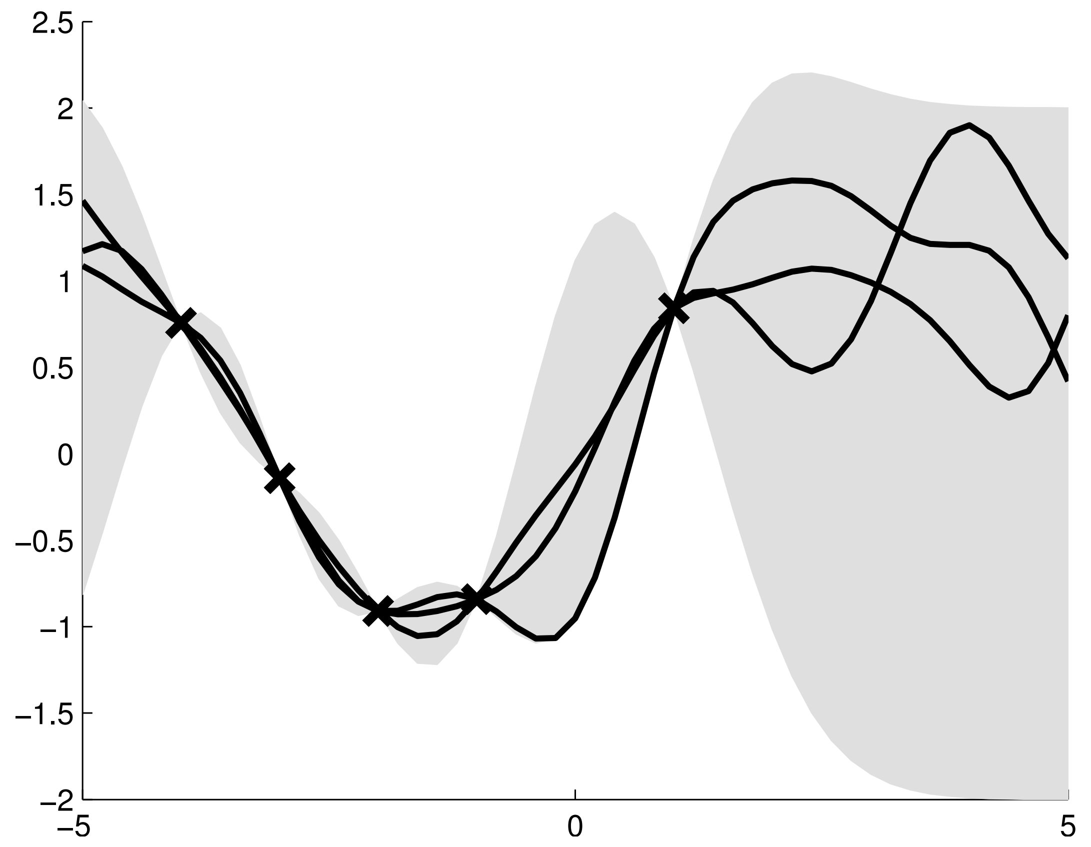
$$\Sigma_* = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T K^{-1} \mathbf{k}_*$$

This defines the **predictive distribution.**

Example



Functions sampled from a Gaussian Process prior



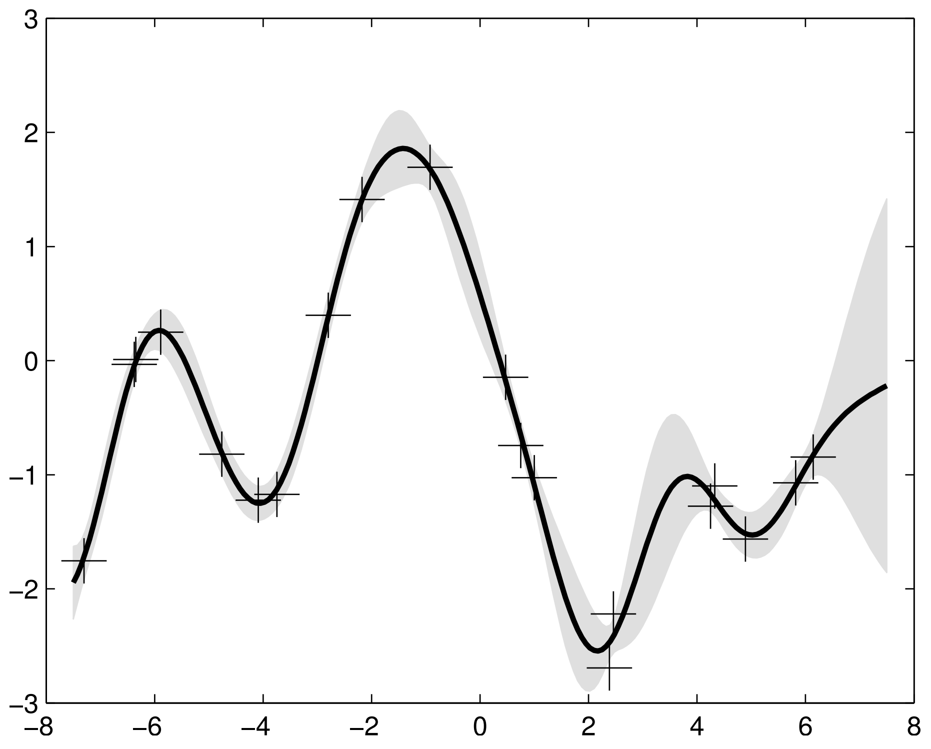
Functions sampled from the predictive distribution

The predictive distribution is itself a Gaussian process.

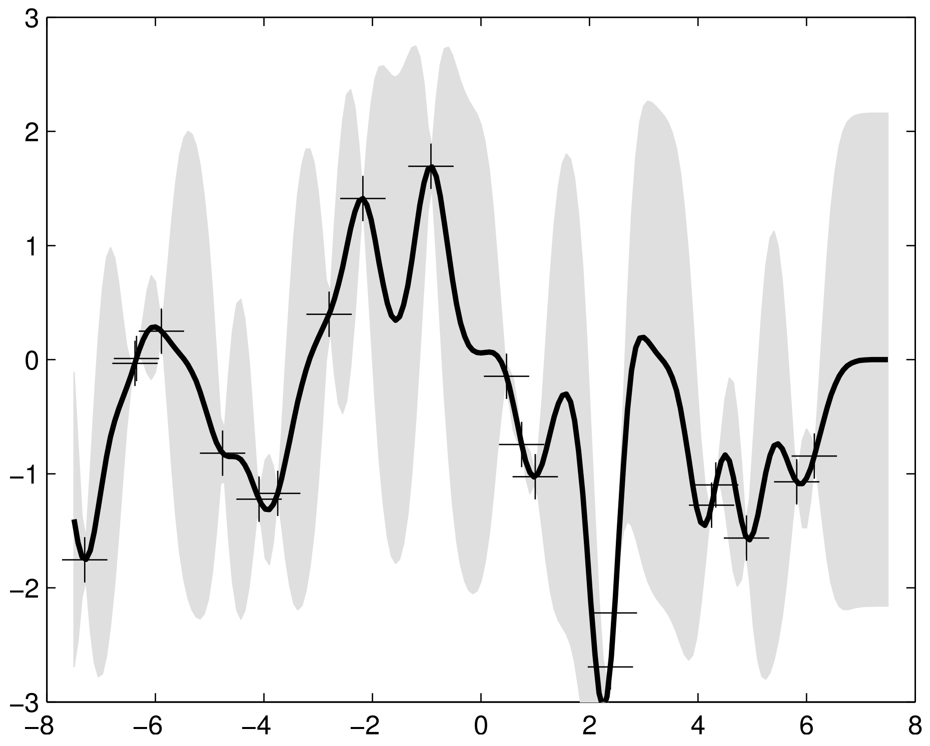
It represents the posterior after observing the data.

The covariance is low in the vicinity of data points.

Varying the Hyperparameters



$$l = \sigma_f = 1, \quad \sigma_n = 0.1$$

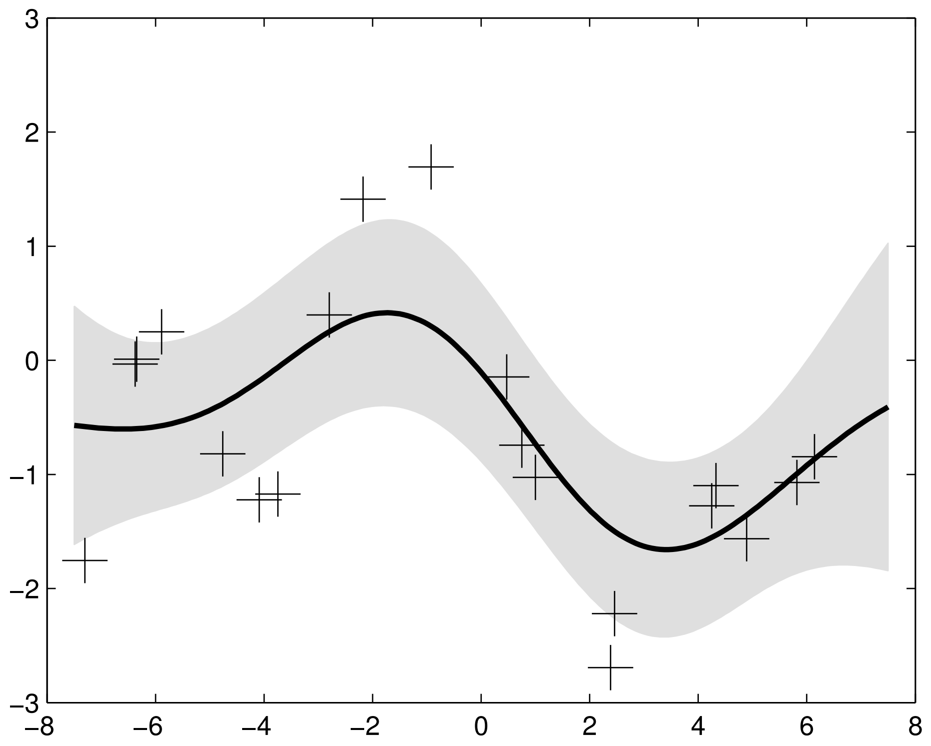


$$l = 0.3,$$

$$\sigma_f = 1.08,$$

$$\sigma_n = 0.0005$$

- 20 data samples
- GP prediction with different kernel hyper parameters



$$l = 3$$

$$\sigma_f = 1.16$$

$$\sigma_n = 0.89$$

Varying the Hyperparameters

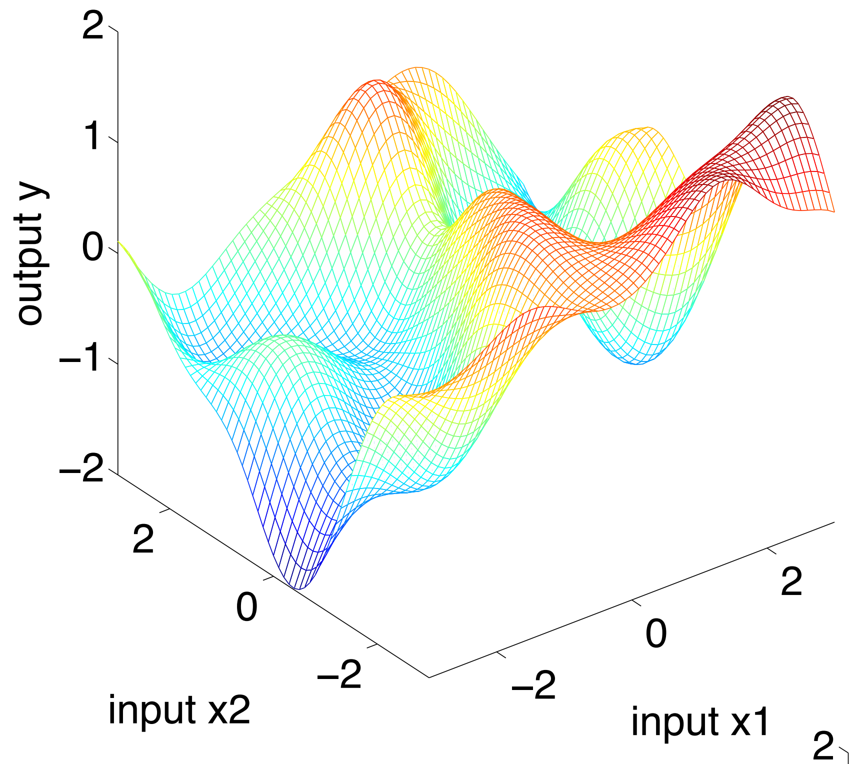
The squared exponential covariance function can be generalized to

$$k(\mathbf{x}_p, \mathbf{x}_q) = \sigma_f^2 \exp(-\frac{1}{2}(\mathbf{x}_p - \mathbf{x}_q)^T M(\mathbf{x}_p - \mathbf{x}_q)) + \sigma_n^2 \delta_{pq}$$

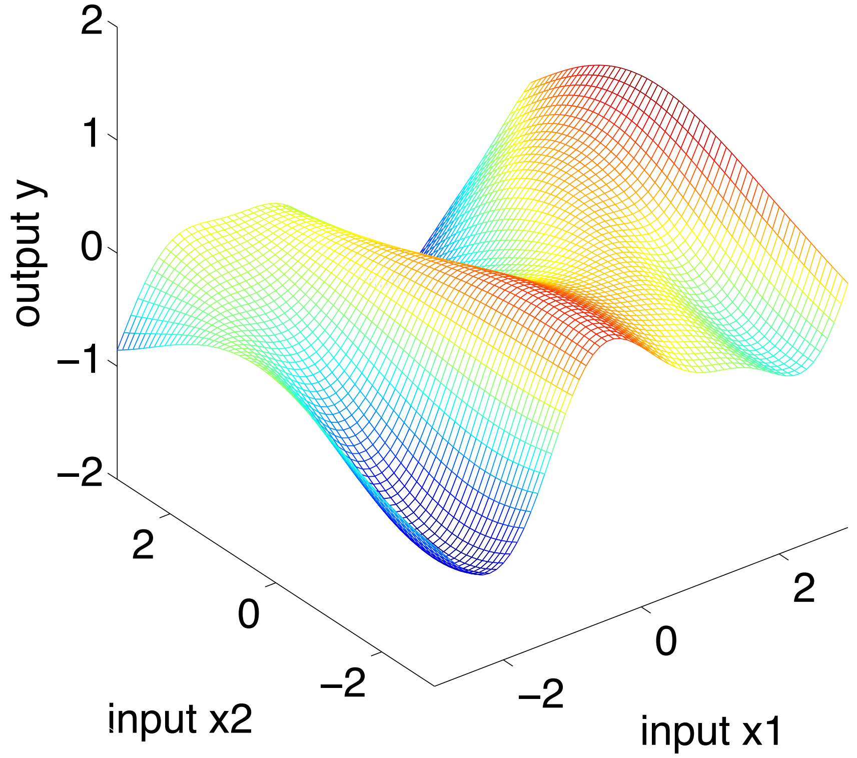
where M can be:

- $M = l^{-2} I$: this is equal to the above case
- $M = \operatorname{diag}(l_1, \ldots, l_D)^{-2}$: every feature dimension has its own length scale parameter
- $M = \Lambda\Lambda^T + \operatorname{diag}(l_1, \ldots, l_D)^{-2}$: here Λ has less than D columns

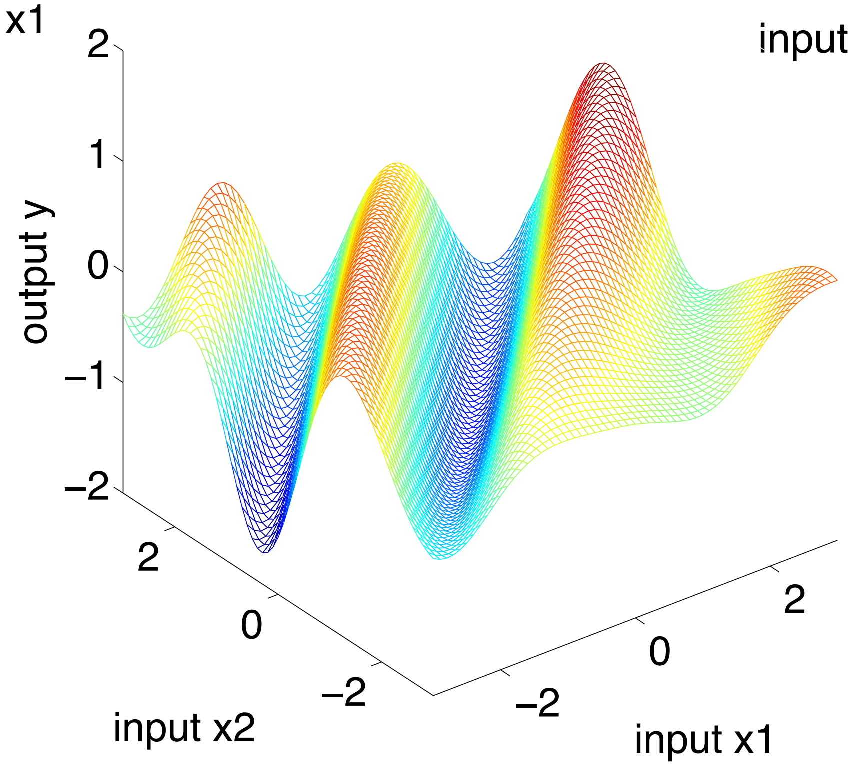
Varying the Hyperparameters



$$M = I$$



$$M = \text{diag}(1,3)^{-2}$$



$$M = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} + \text{diag}(6,6)^{-2}$$

Implementation

Algorithm 1: GP regression

```
Data: training data $(X, \mathbf{y})$, test data $\mathbf{x}_*$
Input: Hyper parameters $\sigma_f^2$, $l$, $\sigma_n^2$
$K_{ij} \leftarrow k(\mathbf{x}_i, \mathbf{x}_j)$
$L \leftarrow \texttt{cholesky}(K + \sigma_n^2 I)$
$\boldsymbol{\alpha} \leftarrow L^T \backslash (L \backslash \mathbf{y})$
$\mathbb{E}[f_*] \leftarrow \mathbf{k}_*^T \boldsymbol{\alpha}$
$\mathbf{v} \leftarrow L \backslash \mathbf{k}_*$
$\texttt{var}[f_*] \leftarrow k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T \mathbf{v}$
$\log p(\mathbf{y} \mid X) \leftarrow -\frac{1}{2}\mathbf{y}^T \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{N}{2} \log(2\pi)$
```

Training Phase

Test Phase

- Cholesky decomposition is numerically stable
- Can be used to compute inverse efficiently

Estimating the Hyperparameters

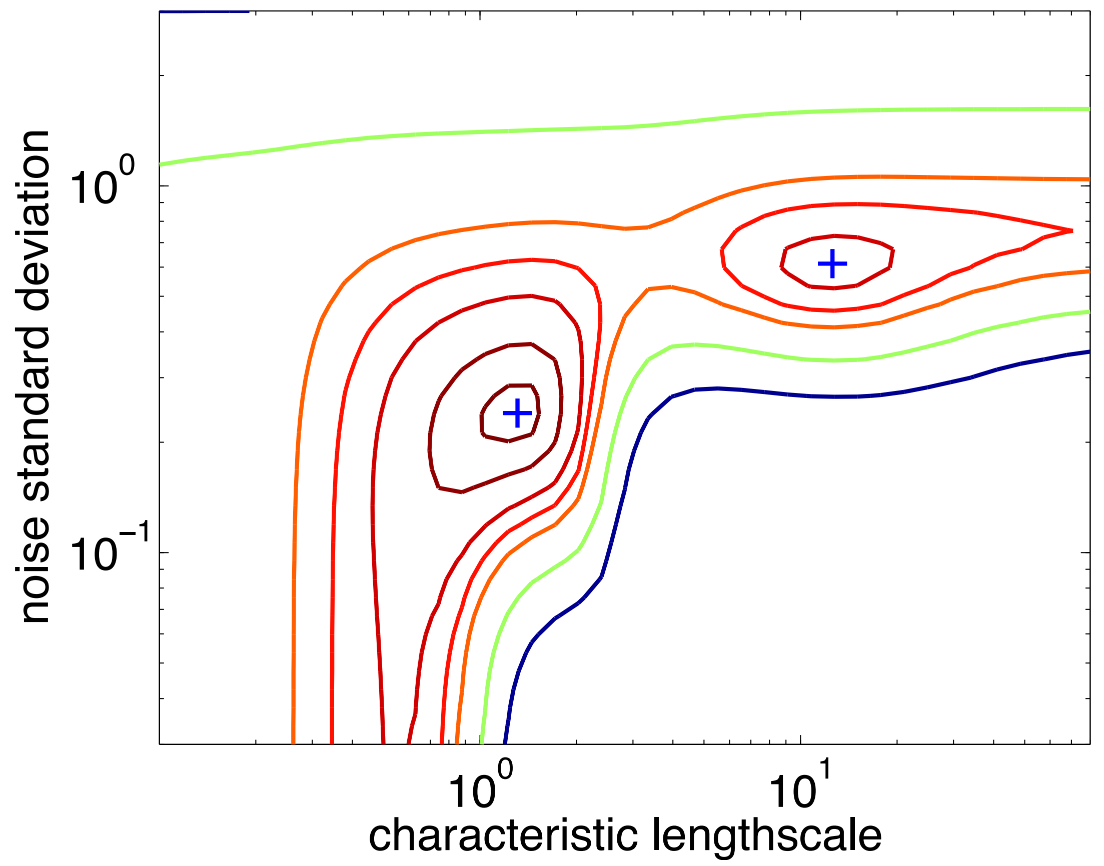
To find optimal hyper parameters we need the **marginal likelihood:**

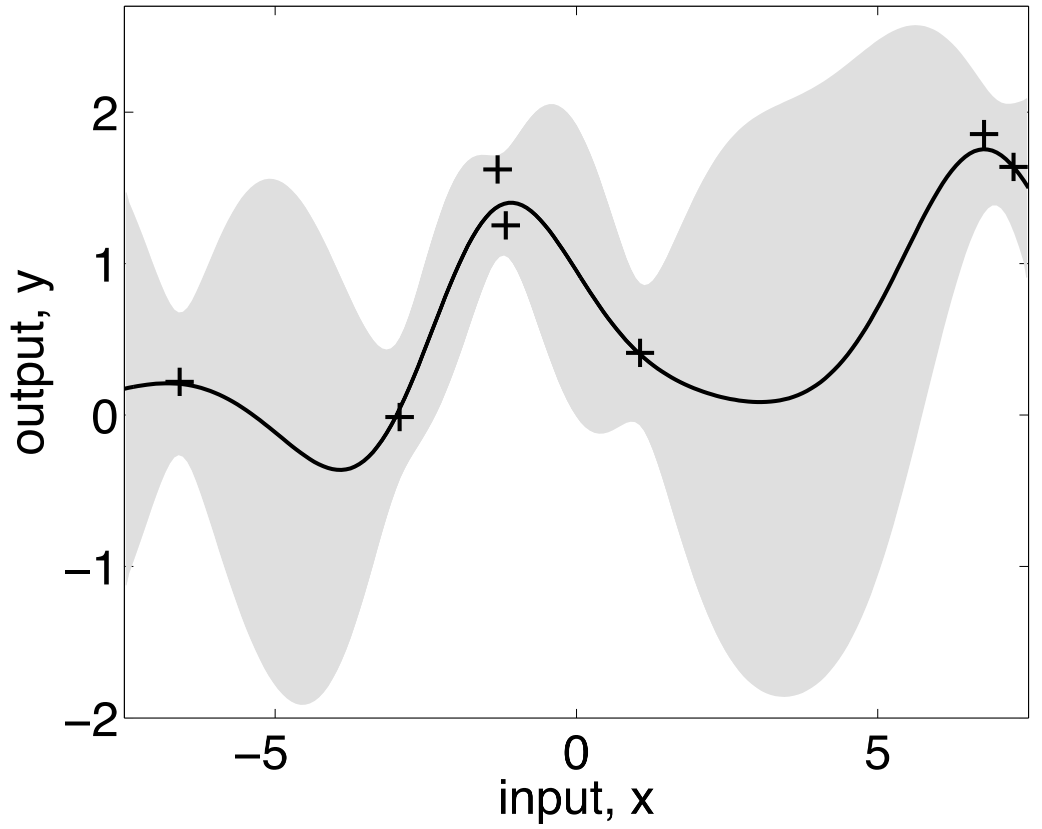
$$p(\mathbf{y} \mid X) = \int p(\mathbf{y} \mid \mathbf{f}, X) p(\mathbf{f} \mid X) d\mathbf{f}$$

This expression implicitly depends on the hyper parameters, but $\mathbf{y}$ and X are given from the training data. It can be computed in closed form, as all terms are Gaussians.

We take the logarithm, compute the derivative and set it to 0. This is the **training** step.

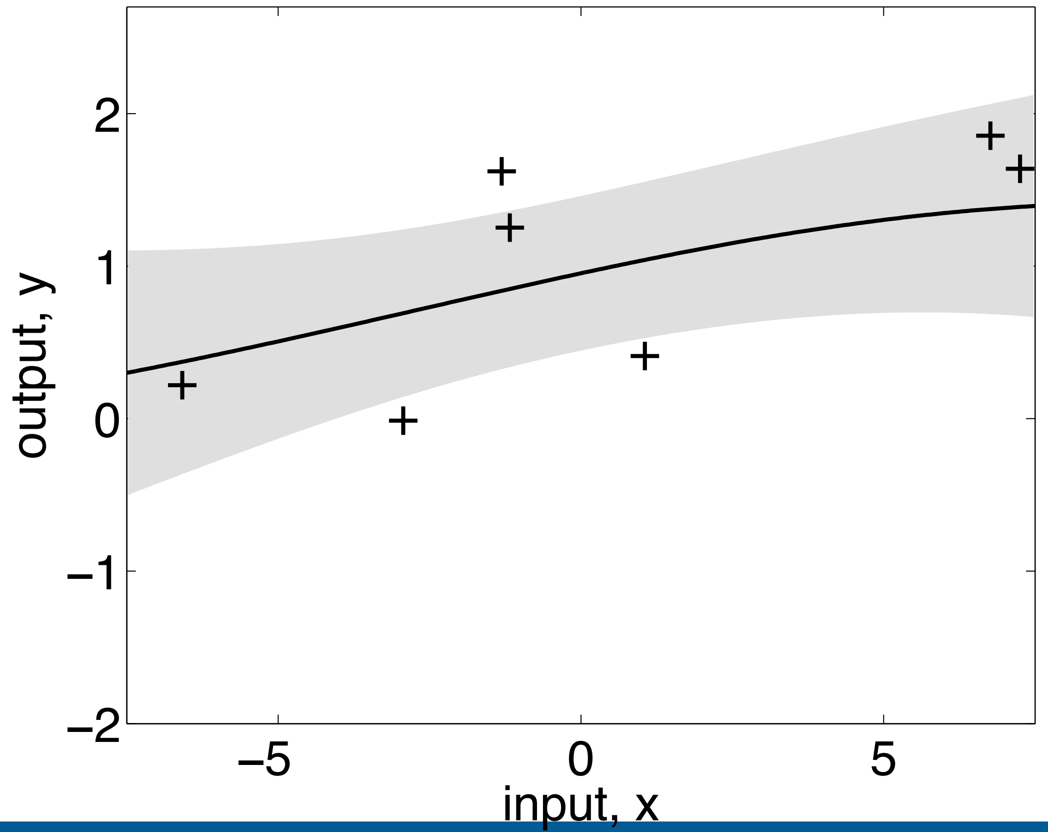
Estimating the Hyperparameters





The log marginal likelihood is not necessarily concave, i.e. it can have local maxima.

The local maxima can correspond to sub-optimal solutions.



Automatic Relevance Determination

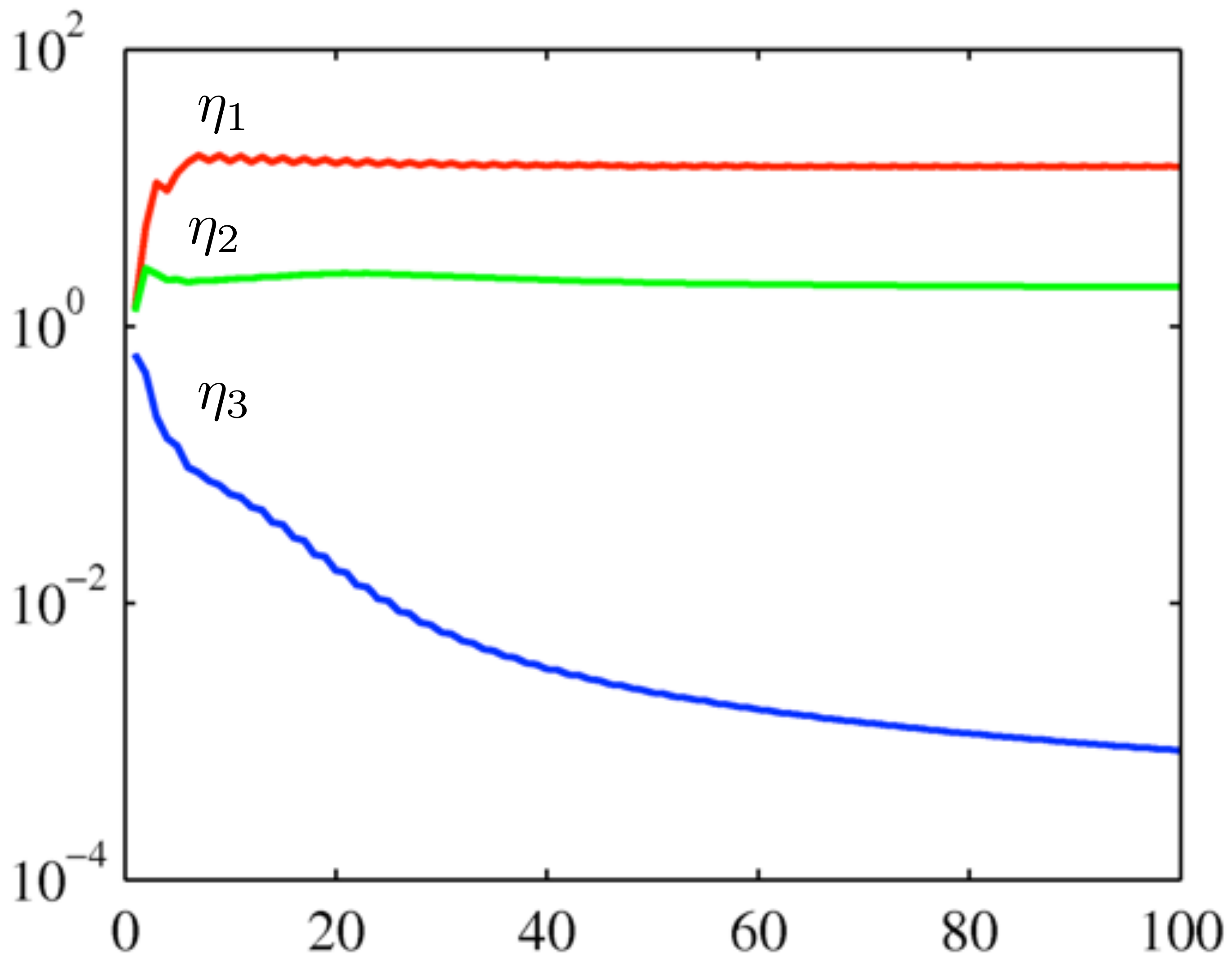
- We have seen how the covariance function can be generalized using a matrix M
- If M is diagonal this results in the kernel function

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f \exp\left(\frac{1}{2}\sum_{i=1}^{D} \eta_i (x_i - x_i')^2\right)$$

- We can interpret the η_i as weights for each feature dimension
- Thus, if the length scale $l_i = 1/\eta_i$ of an input dimension is large, the input is less relevant
- During training this is done automatically

Automatic Relevance Determination

3-dimensional data, parameters η_1 η_2 η_3 as they evolve during training



During the optimization process to learn the hyper-parameters, the reciprocal length scale for one parameter decreases, i.e.:

This hyper parameter is not very relevant!

Gaussian Processes - Classification

Gaussian Processes For Classification

In regression we have $y \in \mathbb{R}$, in binary classification we have $y \in \{-1; 1\}$

To use a GP for classification, we can apply a **sigmoid** function to the posterior obtained from the GP and compute the class probability as:

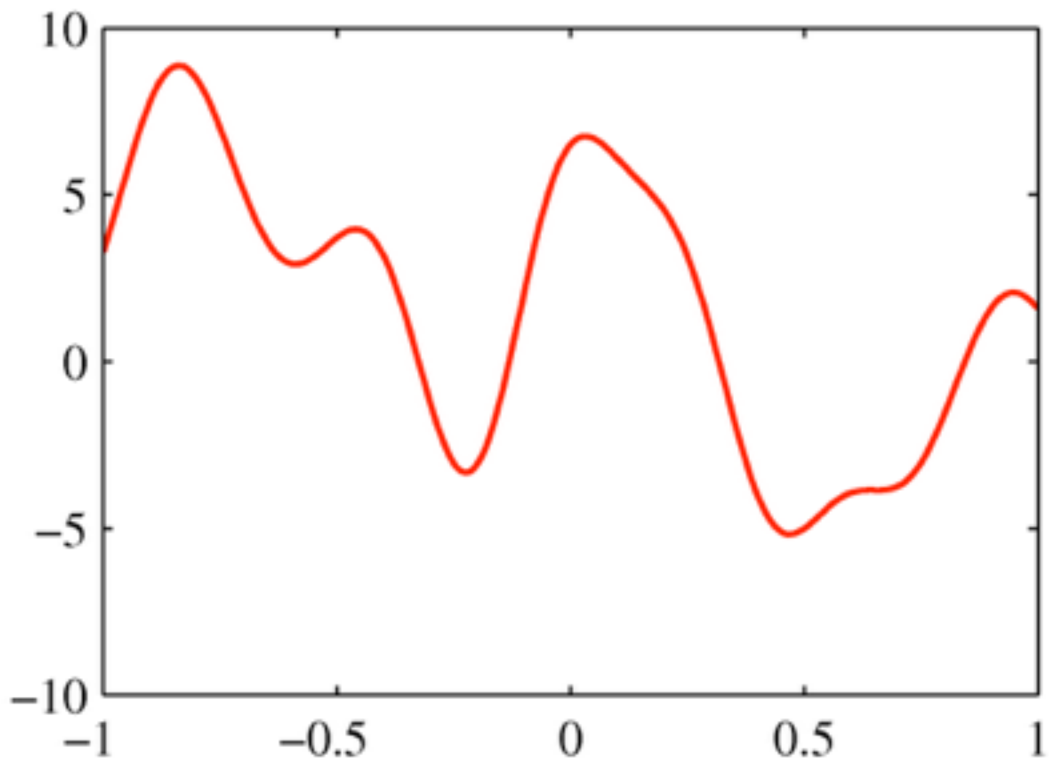
$$p(y = +1 \mid \mathbf{x}) = \sigma(f(\mathbf{x}))$$

If the sigmoid function is symmetric: $\sigma(-z) = 1 - \sigma(z)$ then we have $p(y \mid \mathbf{x}) = \sigma(yf(\mathbf{x}))$.

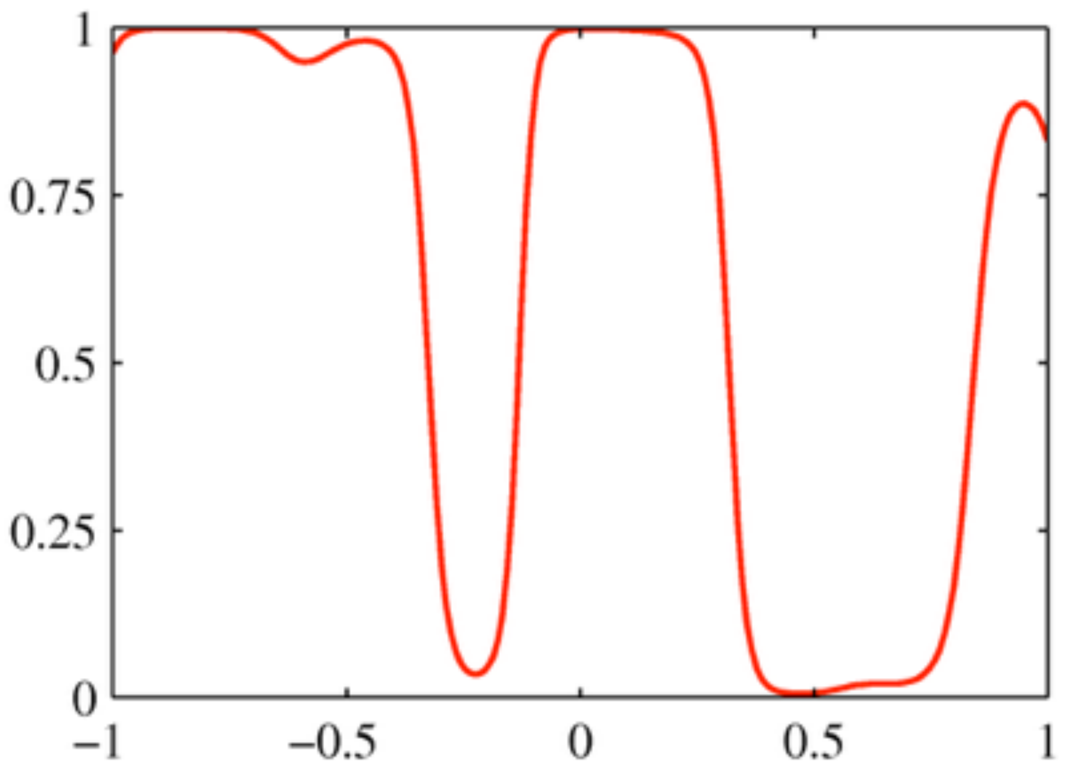
A typical type of sigmoid function is the logistic sigmoid:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

Application of the Sigmoid Function



Function sampled from a Gaussian Process

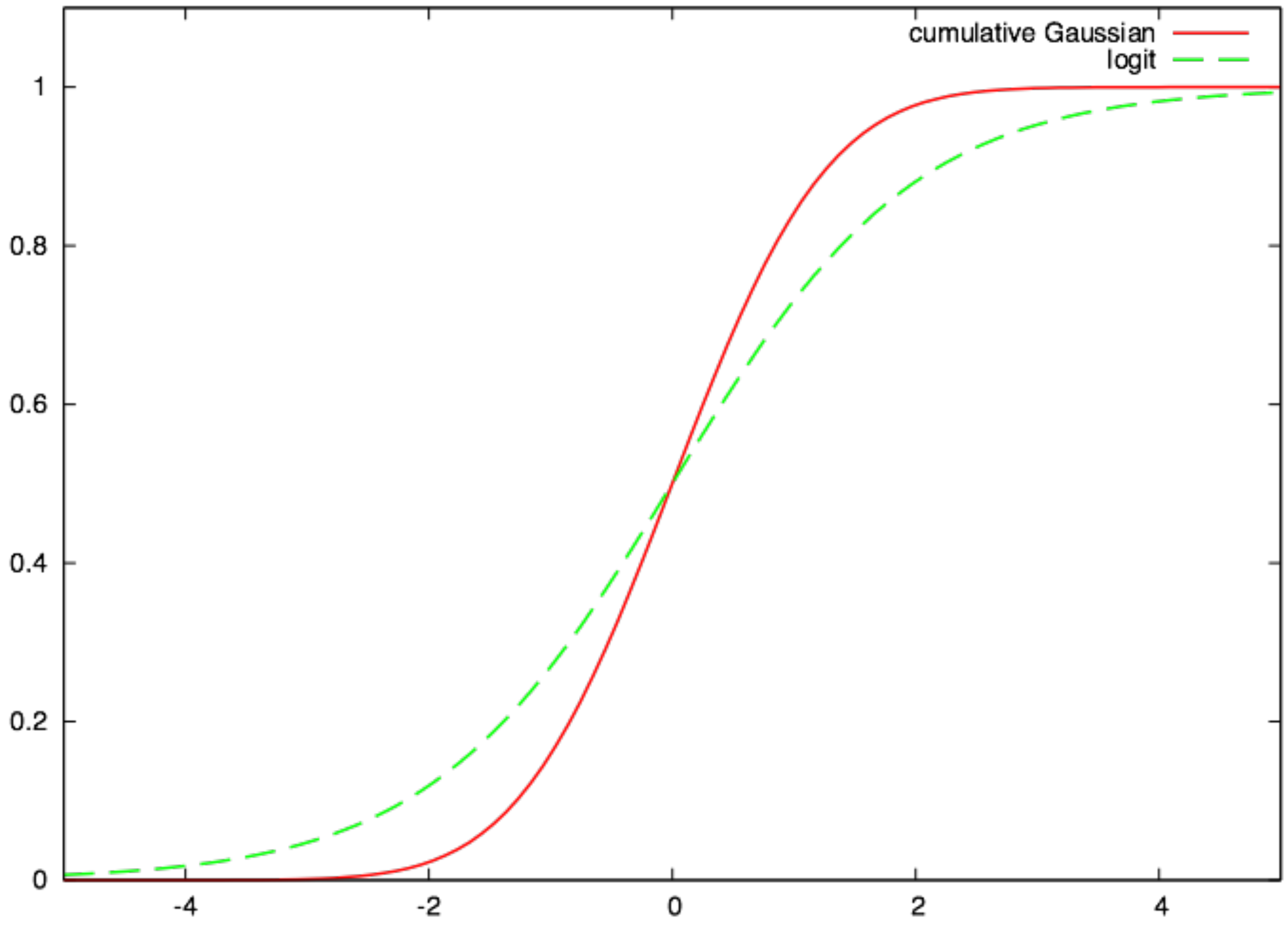


Sigmoid function applied to the GP function

Another symmetric sigmoid function is the **cumulative Gaussian:**

$$\Phi(z) = \int_{-\infty}^{z} \mathcal{N}(x \mid 0, 1) dx$$

Visualization of Sigmoid Functions



The cumulative Gaussian is slightly steeper than the logistic sigmoid

The Latent Variables

In regression, we directly estimated f as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

and values of f where observed in the training data. Now only labels +1 or -1 are observed and f is treated as a set of **latent variables.**

A major advantage of the Gaussian process classifier over other methods is that it **marginalizes** over all latent functions rather than maximizing some model parameters.

Class Prediction with a GP

The aim is to compute the predictive distribution

$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int \underbrace{p(y_* \mid f_*)}_{\sigma(f_*)} p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

Class Prediction with a GP

The aim is to compute the predictive distribution

$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

we marginalize over the latent variables from the training data:

$$p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) = \int \underbrace{p(f_* \mid X, \mathbf{x}_*, \mathbf{f})}_{\text{predictive distribution of the latent variable (from regression)}} p(\mathbf{f} \mid X, \mathbf{y}) d\mathbf{f}$$

Class Prediction with a GP

The aim is to compute the predictive distribution

$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

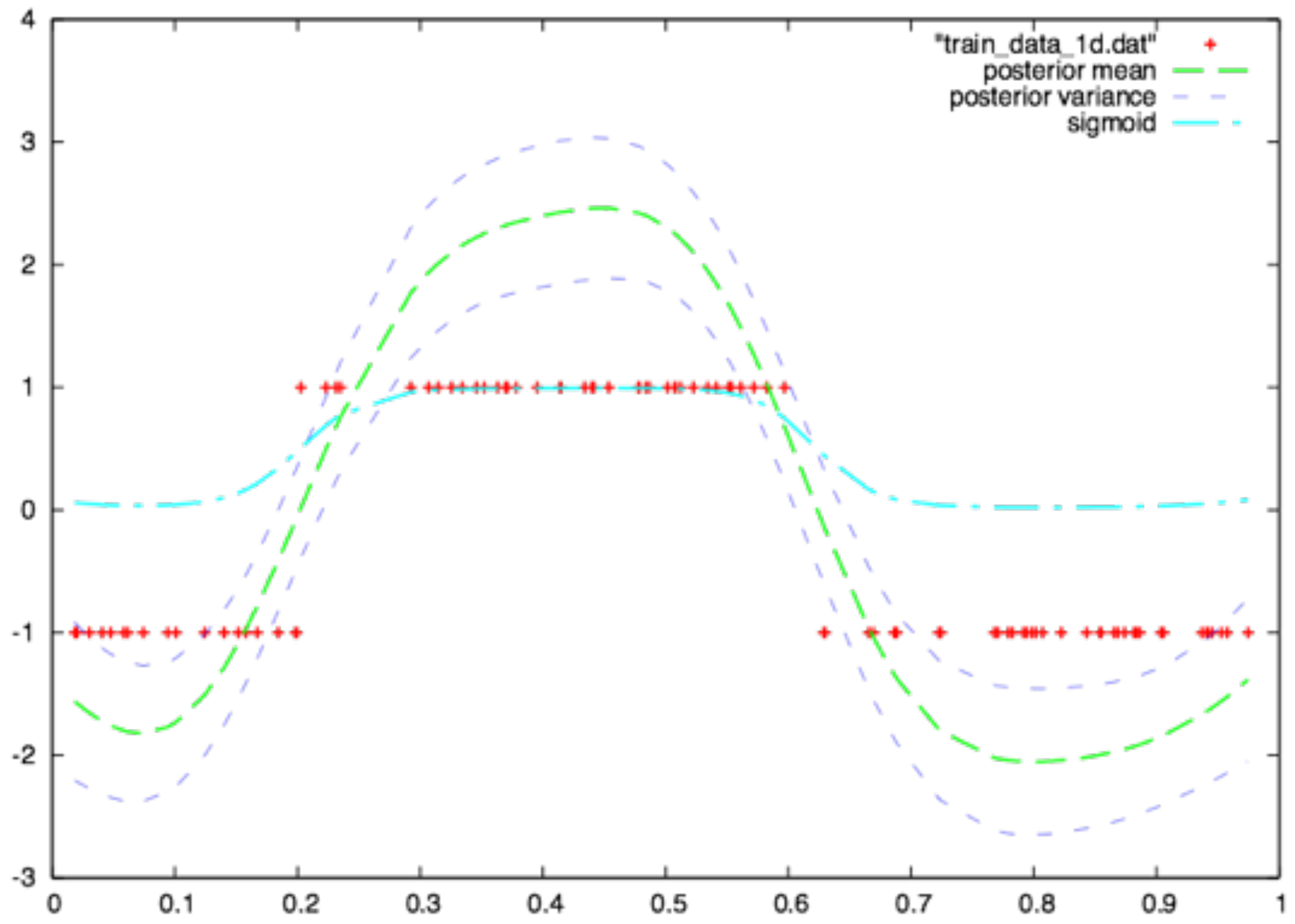
we marginalize over the latent variables from the training data:

$$p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f} \mid X, \mathbf{y}) d\mathbf{f}$$

we need the posterior over the latent variables:

$$p(\mathbf{f} \mid X, \mathbf{y}) = \frac{\overset{\text{likelihood (sigmoid)}}{p(\mathbf{y} \mid \mathbf{f})}\,\overset{\text{prior}}{p(\mathbf{f} \mid X)}}{\underset{\text{normalizer}}{p(\mathbf{y} \mid X)}}$$

A Simple Example



- Red: Two-class training data
- Green: mean function of $p(\mathbf{f} \mid X, \mathbf{y})$
- Light blue: sigmoid of the mean function

But There Is A Problem...

$$p(\mathbf{f} \mid X, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f})p(\mathbf{f} \mid X)}{p(\mathbf{y} \mid X)}$$

- The likelihood term is not a Gaussian!
- This means, we can not compute the posterior in closed form.
- There are several different solutions in the literature, e.g.:
  - Laplace approximation
  - Expectation Propagation
  - Variational methods

Laplace Approximation

$$p(\mathbf{f} \mid X, \mathbf{y}) \approx q(\mathbf{f} \mid X, \mathbf{y}) = \mathcal{N}(\mathbf{f} \mid \hat{\mathbf{f}}, A^{-1})$$

where $\hat{\mathbf{f}} = \arg\max_{\mathbf{f}} p(\mathbf{f} \mid X, \mathbf{y})$

and $A = -\nabla\nabla \log p(\mathbf{f} \mid X, \mathbf{y})|_{\mathbf{f}=\hat{\mathbf{f}}}$

second-order Taylor expansion

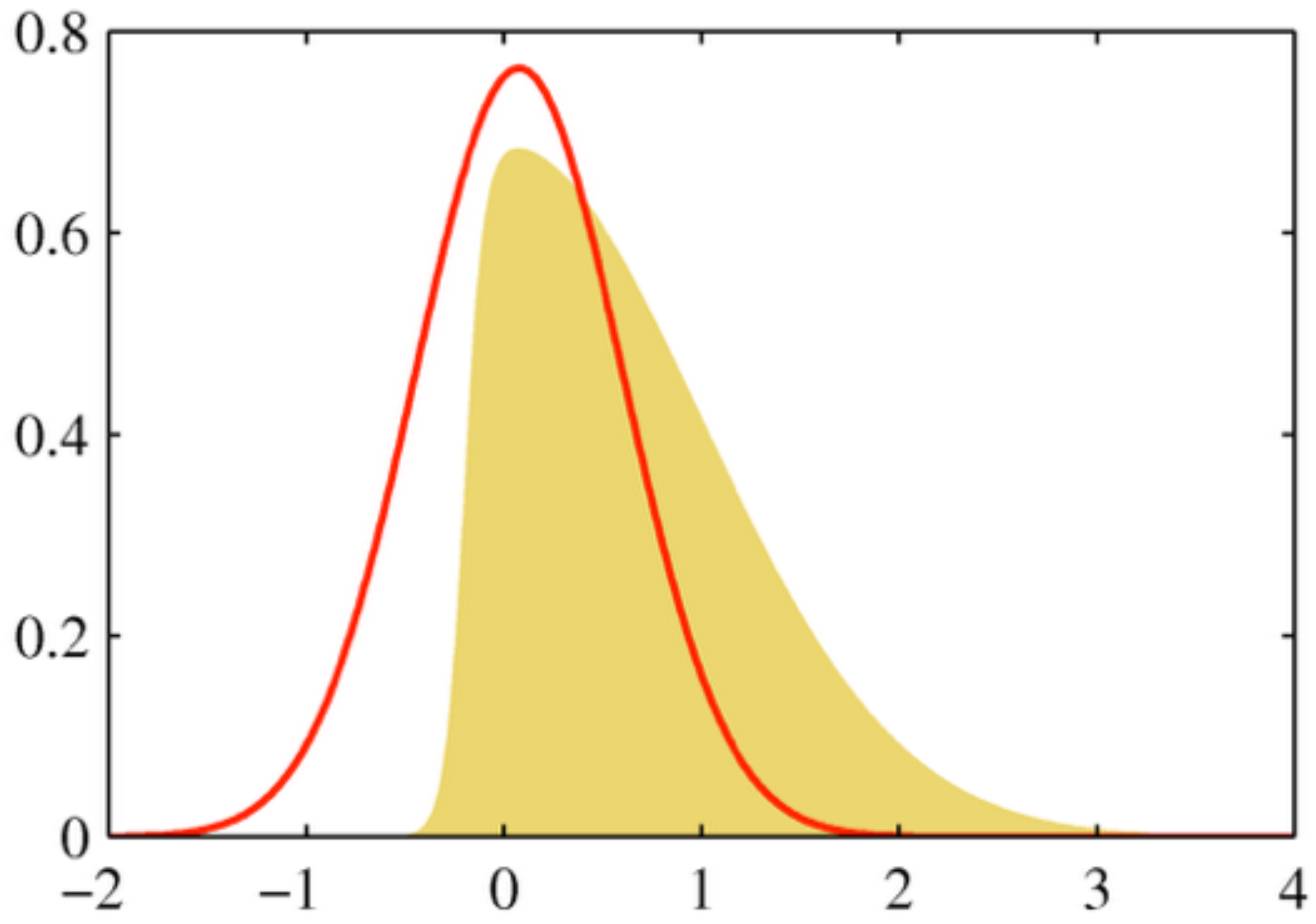
To compute $\hat{\mathbf{f}}$ an iterative approach using Newton's method has to be used.

The Hessian matrix A can be computed as

$$A = K^{-1} + W$$

where $W = -\nabla\nabla \log p(\mathbf{y} \mid \mathbf{f})$ is a diagonal matrix which depends on the sigmoid function.

Laplace Approximation



- Yellow: a non-Gaussian posterior
- Red: a Gaussian approximation, the mean is the mode of the posterior, the variance is the negative second derivative at the mode

Predictions

Now that we have $p(\mathbf{f} \mid X, \mathbf{y})$ we can compute:

$$p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f} \mid X, \mathbf{y}) d\mathbf{f}$$

From the regression case we have:

$$p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) = \mathcal{N}(f_* \mid \mu_*, \Sigma_*)$$

where $\mu_* = \mathbf{k}_*^T K^{-1} \mathbf{f}$ $\qquad \Sigma_* = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T K^{-1} \mathbf{k}_*$

Linear in $\mathbf{f}$

This reminds us of a property of Gaussians that we saw earlier!

Gaussian Properties (Rep.)

If we are given this:

I. $p(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \mu, \Sigma_1)$

II. $p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}(\mathbf{y} \mid A\mathbf{x} + \mathbf{b}, \Sigma_2)$

Then it follows (properties of Gaussians):

III. $p(\mathbf{y}) = \mathcal{N}(\mathbf{y} \mid A\mu + \mathbf{b}, \Sigma_2 + A\Sigma_1 A^T)$

IV. $p(\mathbf{x} \mid \mathbf{y}) = \mathcal{N}(\mathbf{x} \mid \Sigma(A^T\Sigma_2^{-1}(\mathbf{y} - \mathbf{b}) + \Sigma_1^{-1}\mathbf{y}), \Sigma)$

where

$$\Sigma = (\Sigma_1^{-1} + A^T \Sigma_s^{-1} A)^{-1}$$

Applying this to Laplace

$$\mathbb{E}[f_* \mid X, \mathbf{y}, \mathbf{x}_*] = \mathbf{k}(\mathbf{x}_*)^T K^{-1} \hat{\mathbf{f}}$$

$$\mathbb{V}[f_* \mid X, \mathbf{y}, \mathbf{x}_*] = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T (K + W^{-1})^{-1} \mathbf{k}_*$$

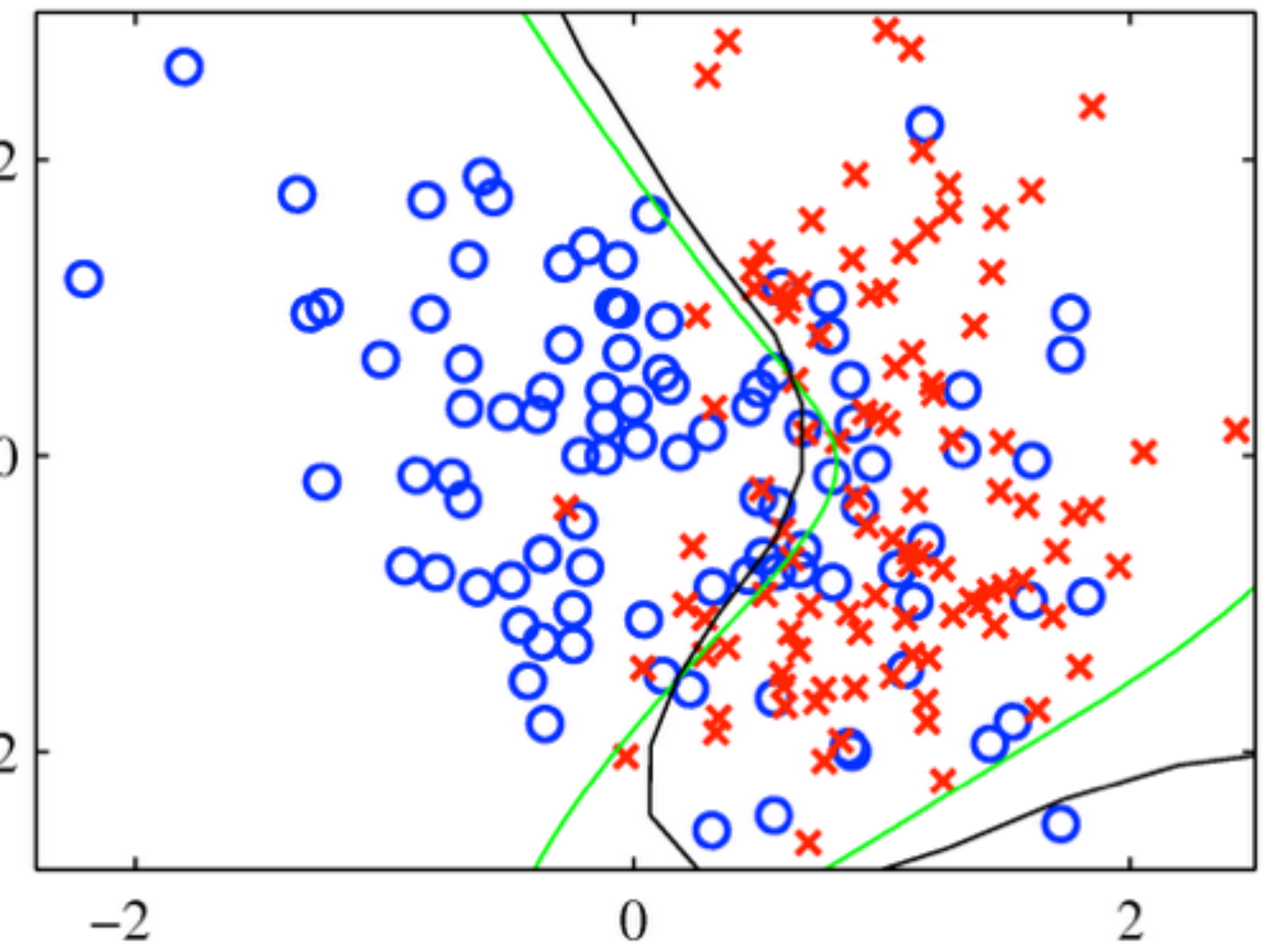
It remains to compute

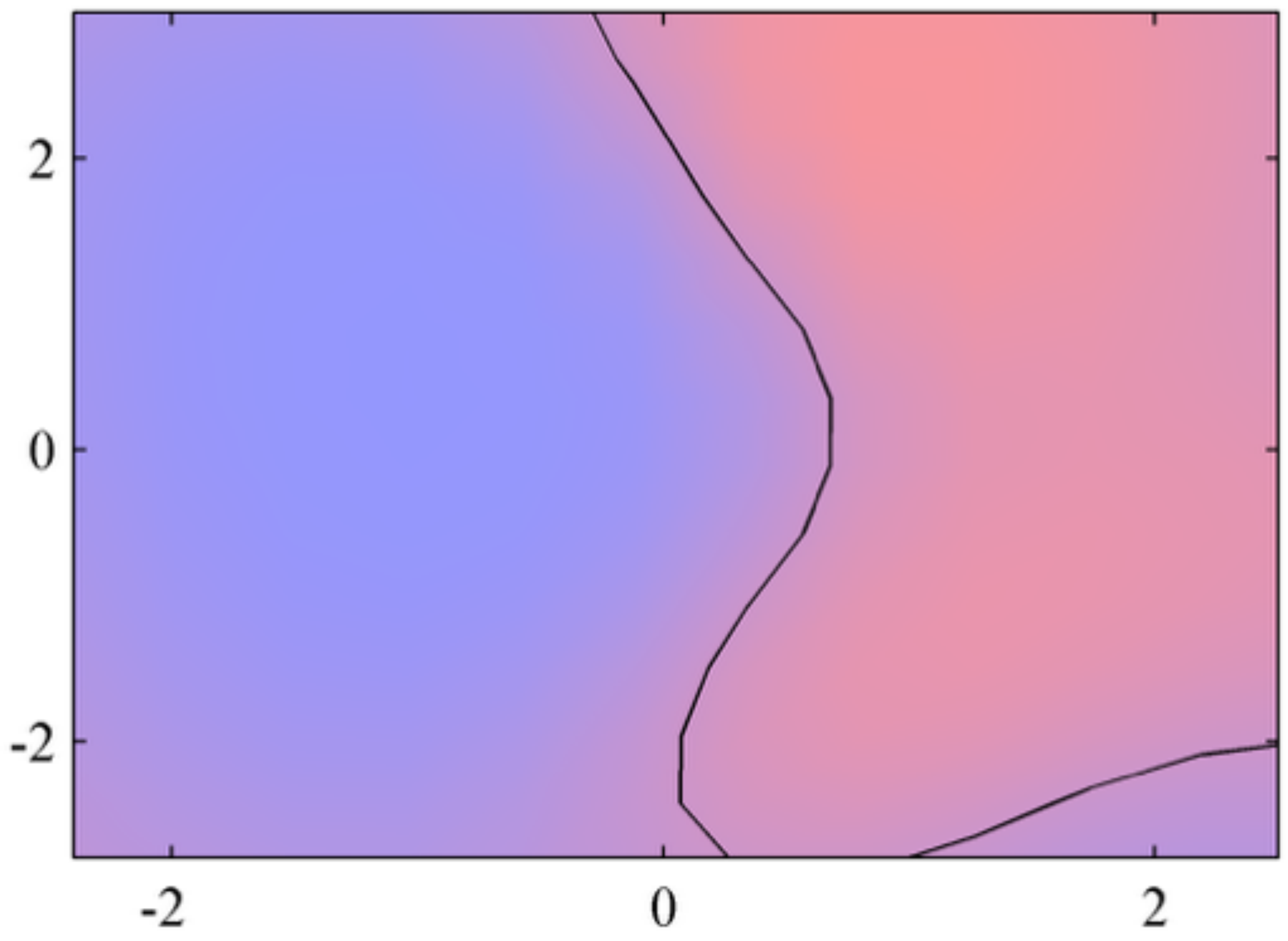
$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

Depending on the kind of sigmoid function we

- can compute this in closed form (cumulative Gaussian sigmoid)
- have to use sampling methods or analytical approximations (logistic sigmoid)

A Simple Example





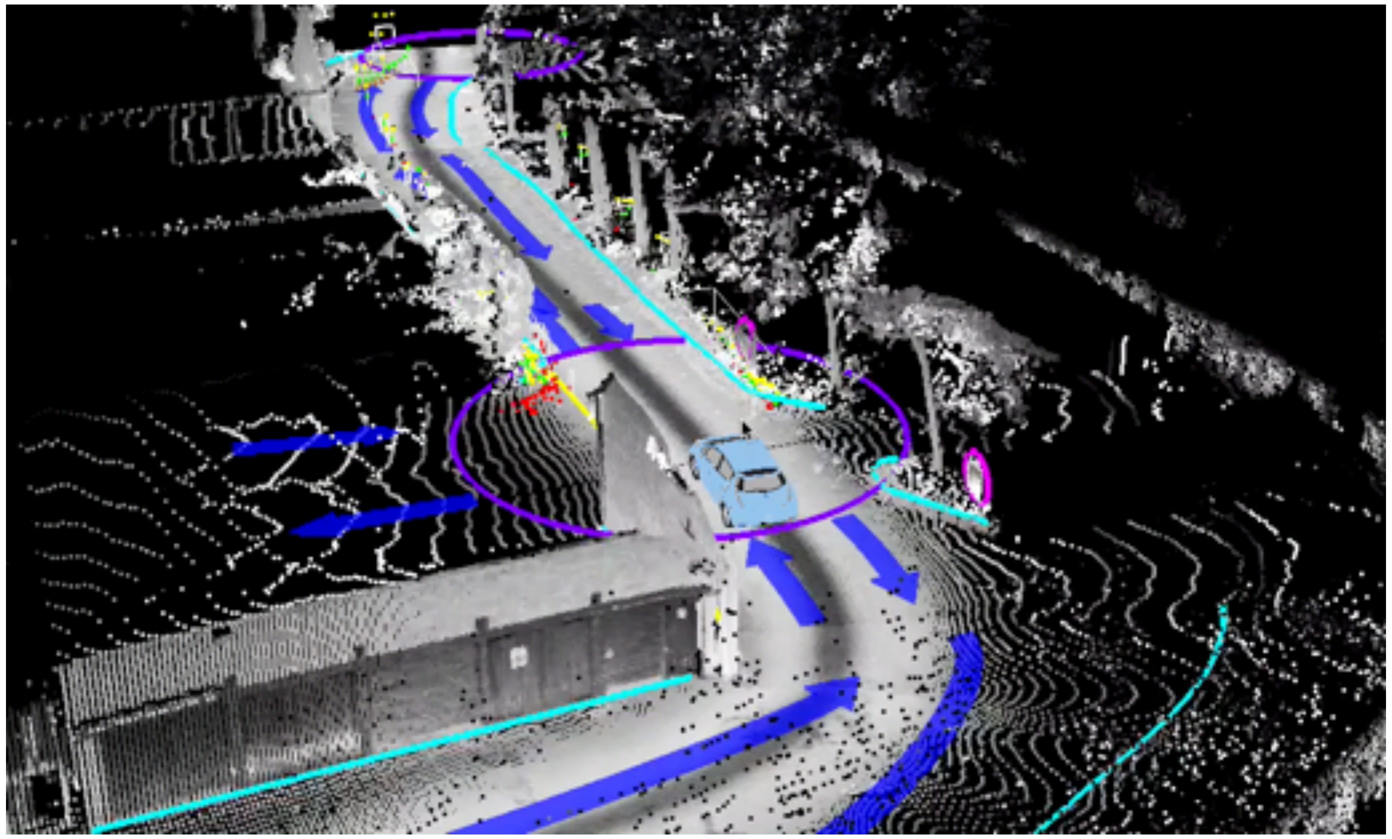
- Two-class problem (training data in red and blue)
- Green line: optimal decision boundary
- Black line: GP classifier decision boundary
- Right: posterior probability

Summary

- Kernel methods solve problems by implicitly mapping the data into a (high-dimensional) feature space
- The feature function itself is not used, instead the algorithm is expressed in terms of the kernel
- Gaussian Processes are Normal distributions over functions
- To specify a GP we need a covariance function (kernel) and a mean function
- More on Gaussian Processes: http://videolectures.net/epsrcws08_rasmussen_lgp/

Application Example: Semantic Mapping

Semantic Mapping


Hand-crafted 3D semantic map

Benchmark data for semantic mapping

Active Learning is well suited for semantic mapping because:

- it can deal with **large amounts of data.**
- data is not **independent.**
- the task is essentially an **online** learning problem.

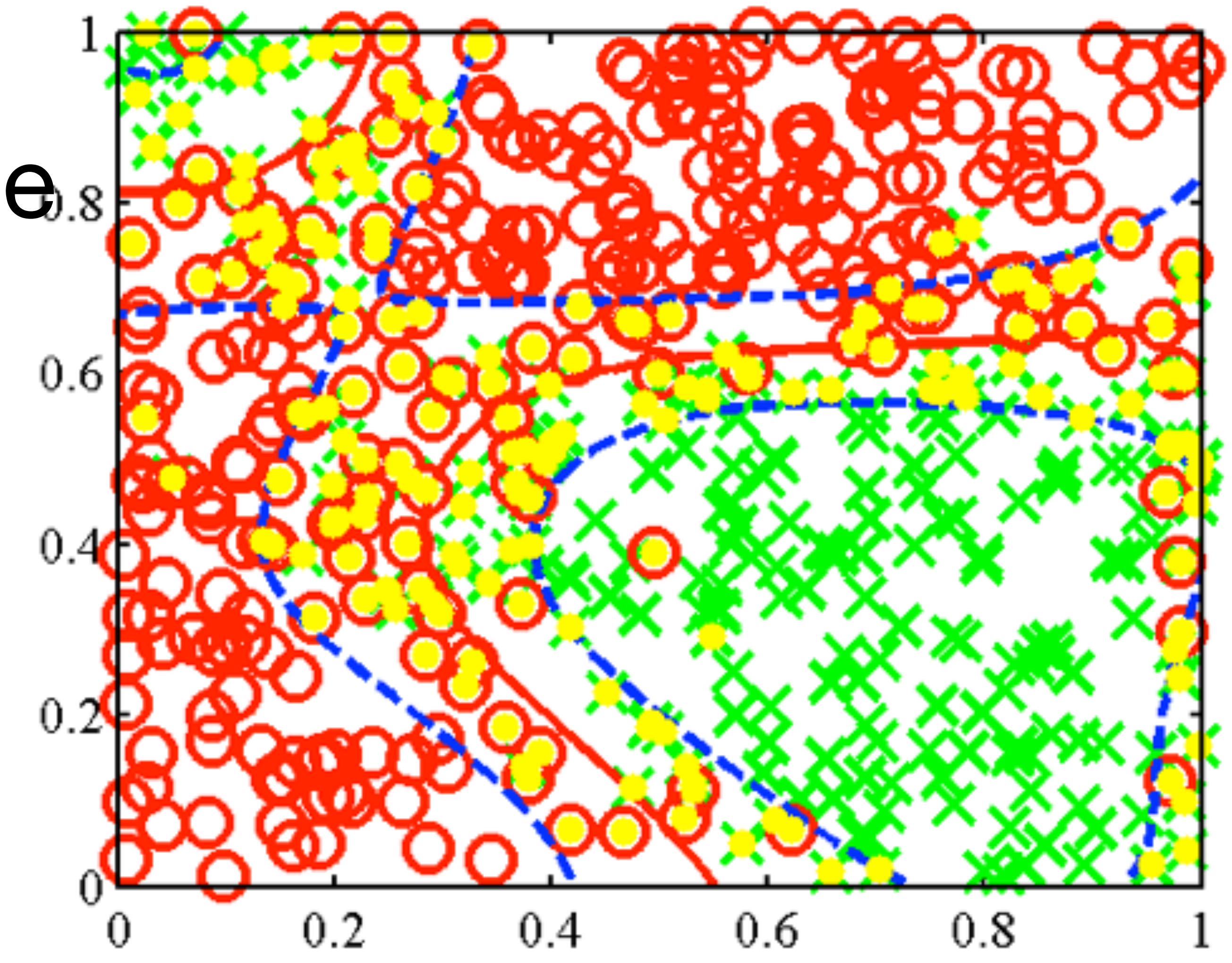
The Informative Vector Machine

Main differences to standard GP classifier:

- it only uses a **subset** (“active set”) of training points
- the (inverse) posterior covariance matrix is computed **incrementally**

Decision of inclusion in the active set based on information-theoretic criterion

Slight caveat: Training of hyper-parameters needs to be done iteratively

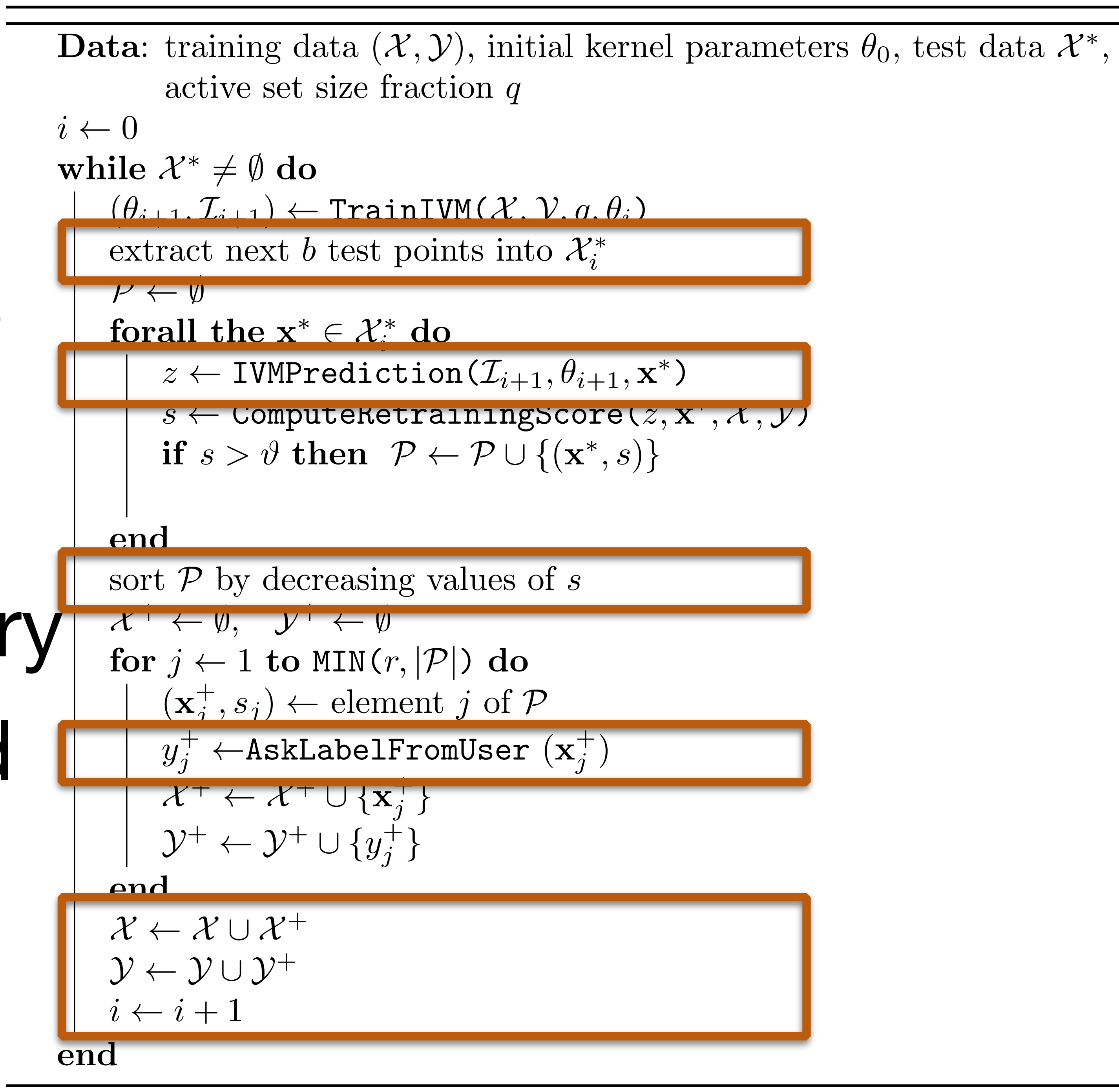


From: http://staffwww.dcs.shef.ac.uk/people/N.Lawrence/ivm/

Active Learning with an IVM

Ongoing Learning algorithm

- New test data arrives
- Classifier predicts a class label and decides if it is uncertain
- The most uncertain points are used for query
- Training set is extended and next training round starts

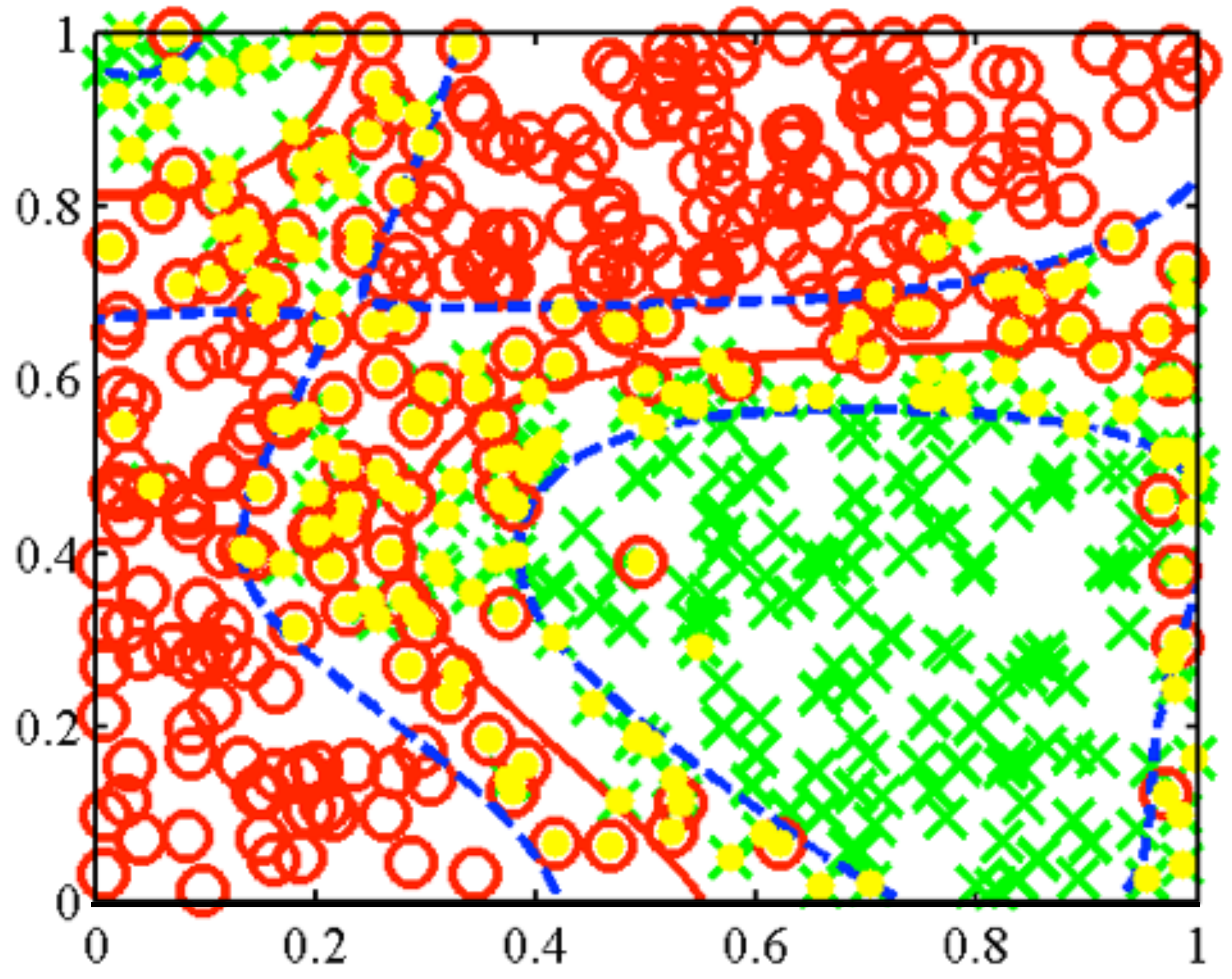


```
Data: training data $(\mathcal{X}, \mathcal{Y})$, initial kernel parameters $\theta_0$, test data $\mathcal{X}^*$,
      active set size fraction $q$
$i \leftarrow 0$
while $\mathcal{X}^* \neq \emptyset$ do
    $(\theta_{i+1}, \mathcal{I}_{i+1}) \leftarrow$ TrainIVM$(\mathcal{X}, \mathcal{Y}, q, \theta_i)$
    extract next $b$ test points into $\mathcal{X}_i^*$
    $\mathcal{P} \leftarrow \emptyset$
    forall the $\mathbf{x}^* \in \mathcal{X}_i^*$ do
        $z \leftarrow$ IVMPrediction$(\mathcal{I}_{i+1}, \theta_{i+1}, \mathbf{x}^*)$
        $s \leftarrow$ ComputeRetrainingScore$(z, \mathbf{x}^*, \mathcal{X}, \mathcal{Y})$
        if $s > \vartheta$ then $\mathcal{P} \leftarrow \mathcal{P} \cup \{(\mathbf{x}^*, s)\}$
    end
    sort $\mathcal{P}$ by decreasing values of $s$
    $\mathcal{X}^+ \leftarrow \emptyset, \quad \mathcal{Y}^+ \leftarrow \emptyset$
    for $j \leftarrow 1$ to MIN$(r, |\mathcal{P}|)$ do
        $(\mathbf{x}_j^+, s_j) \leftarrow$ element $j$ of $\mathcal{P}$
        $y_j^+ \leftarrow$AskLabelFromUser $(\mathbf{x}_j^+)$
        $\mathcal{X}^+ \leftarrow \mathcal{X}^+ \cup \{\mathbf{x}_j^+\}$
        $\mathcal{Y}^+ \leftarrow \mathcal{Y}^+ \cup \{y_j^+\}$
    end
    $\mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{X}^+$
    $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{Y}^+$
    $i \leftarrow i + 1$
end
```

Memory Efficiency

Problem: training data grows continually in every learning round

Idea: constrain the number of training samples



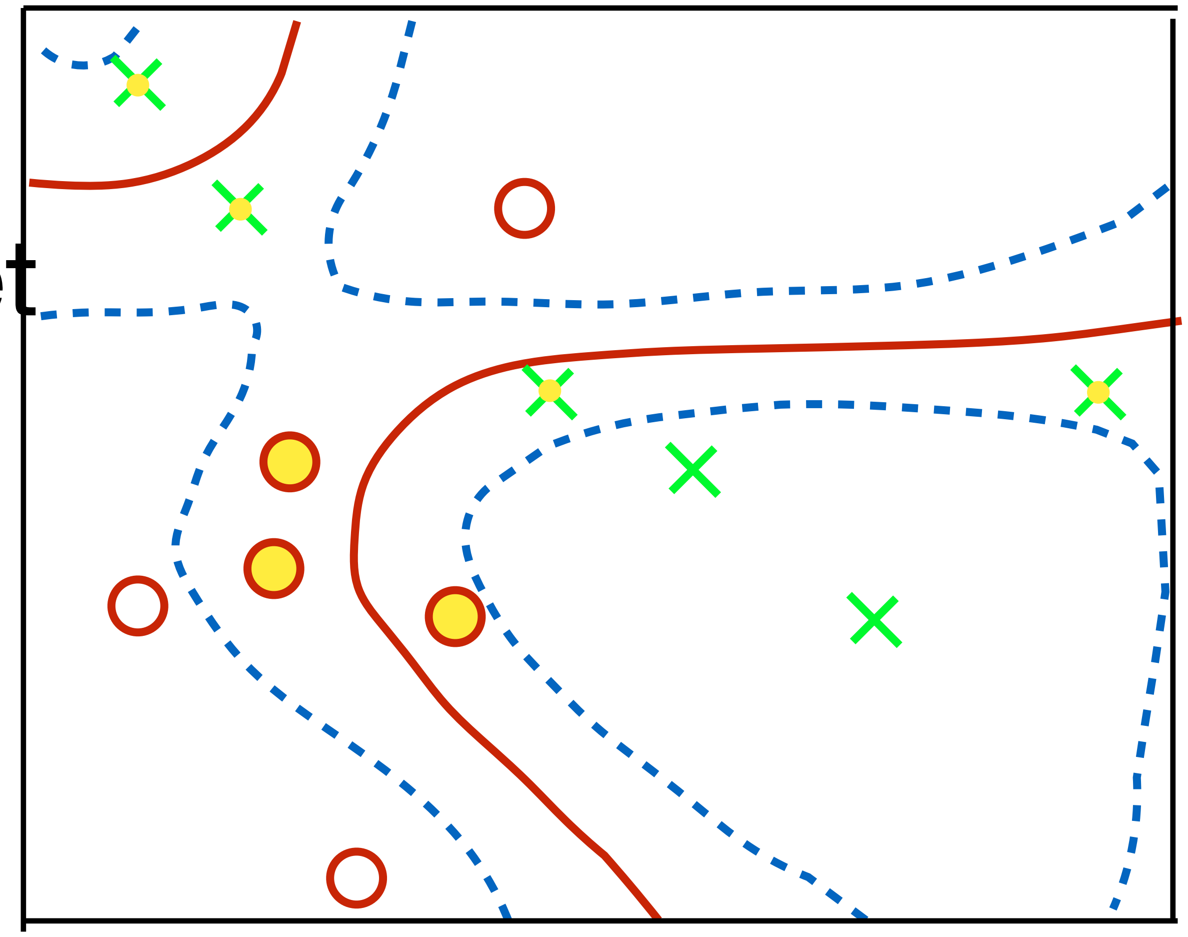
Memory Efficiency

Problem: training data grows continually in every learning round

Idea: constrain the number of training samples

When new point arrives:

- check whether it should be added to the Active Set



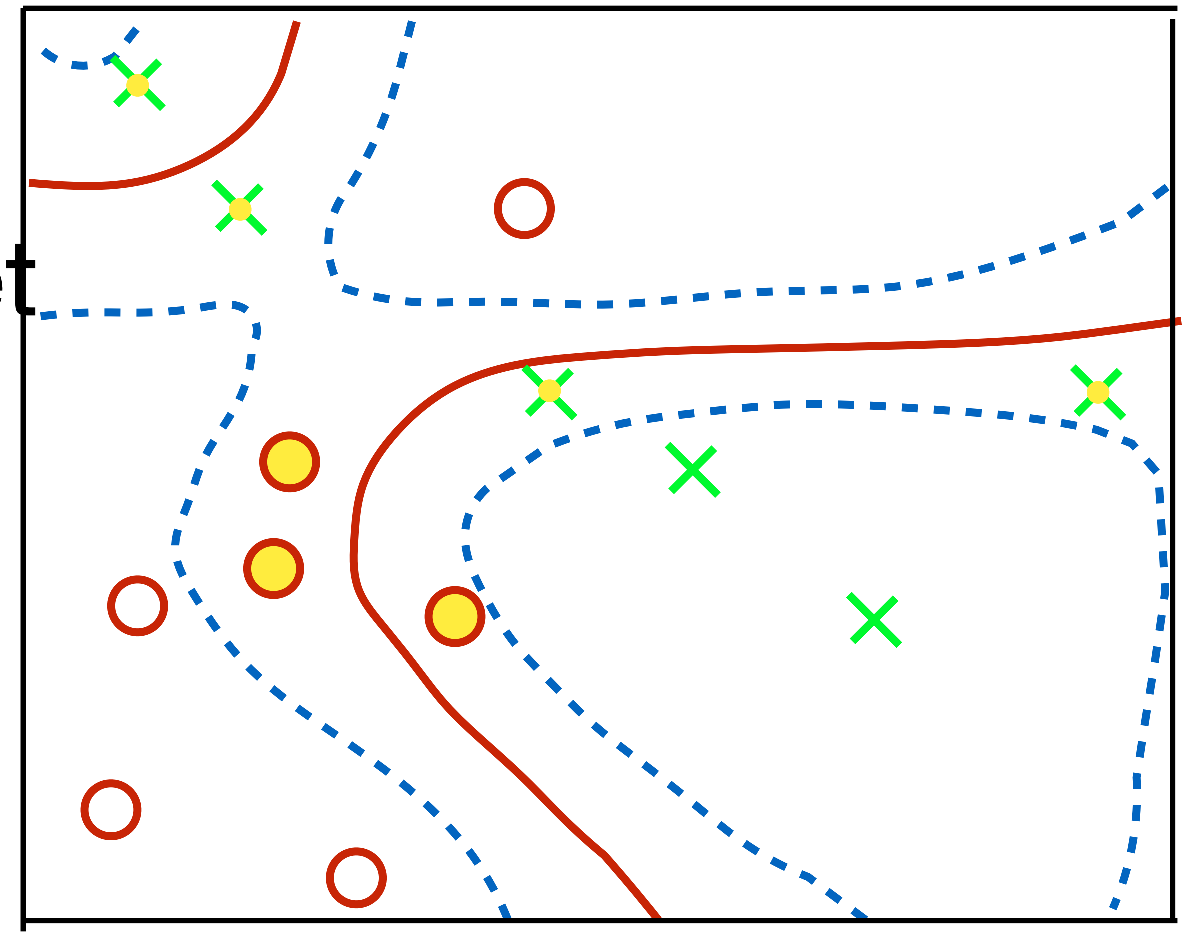
Memory Efficiency

Problem: training data grows continually in every learning round

Idea: constrain the number of training samples

When new point arrives:

- check whether it should be added to the Active Set



NO!
(low entropy change)

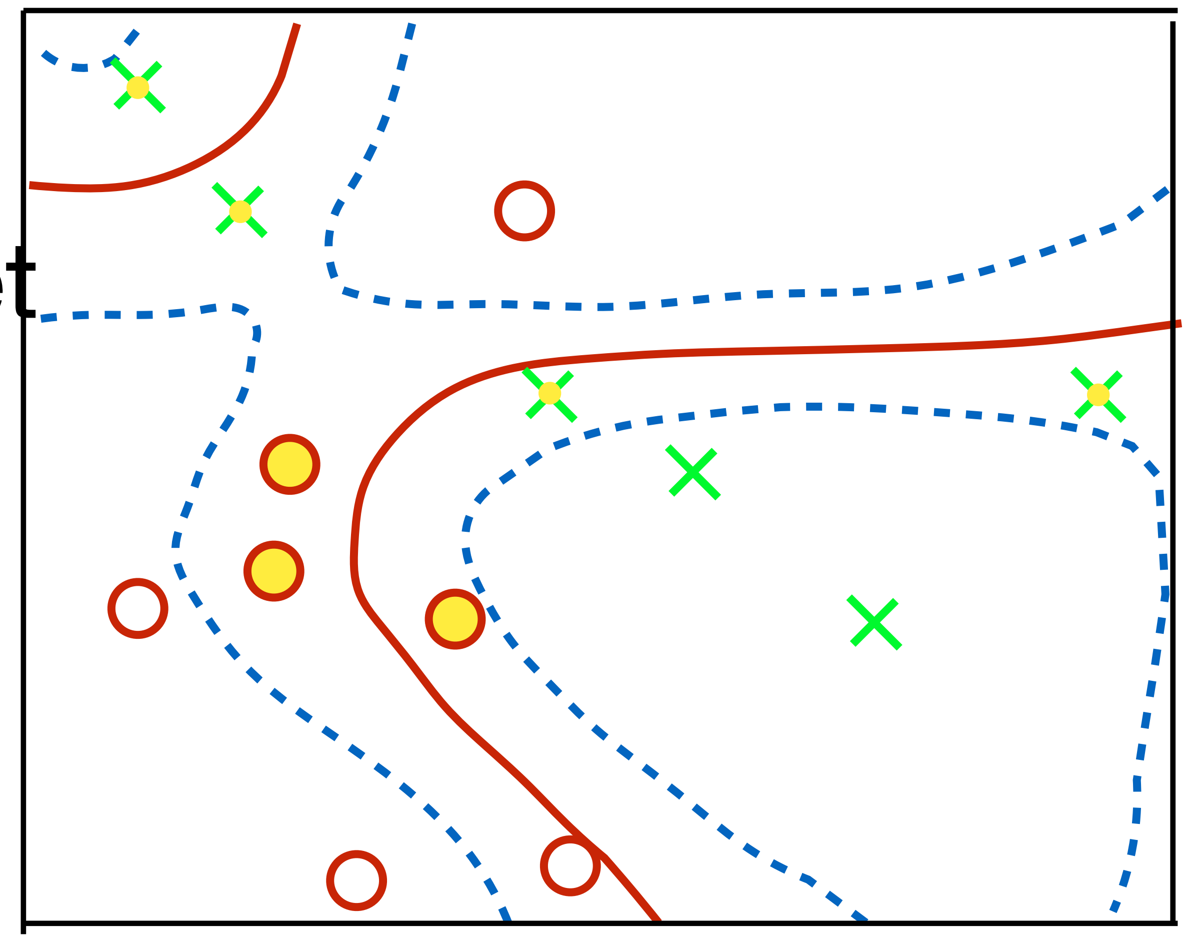
Memory Efficiency

Problem: training data grows continually in every learning round

Idea: constrain the number of training samples

When new point arrives:

- check whether it should be added to the Active Set



YES!
(high entropy change)

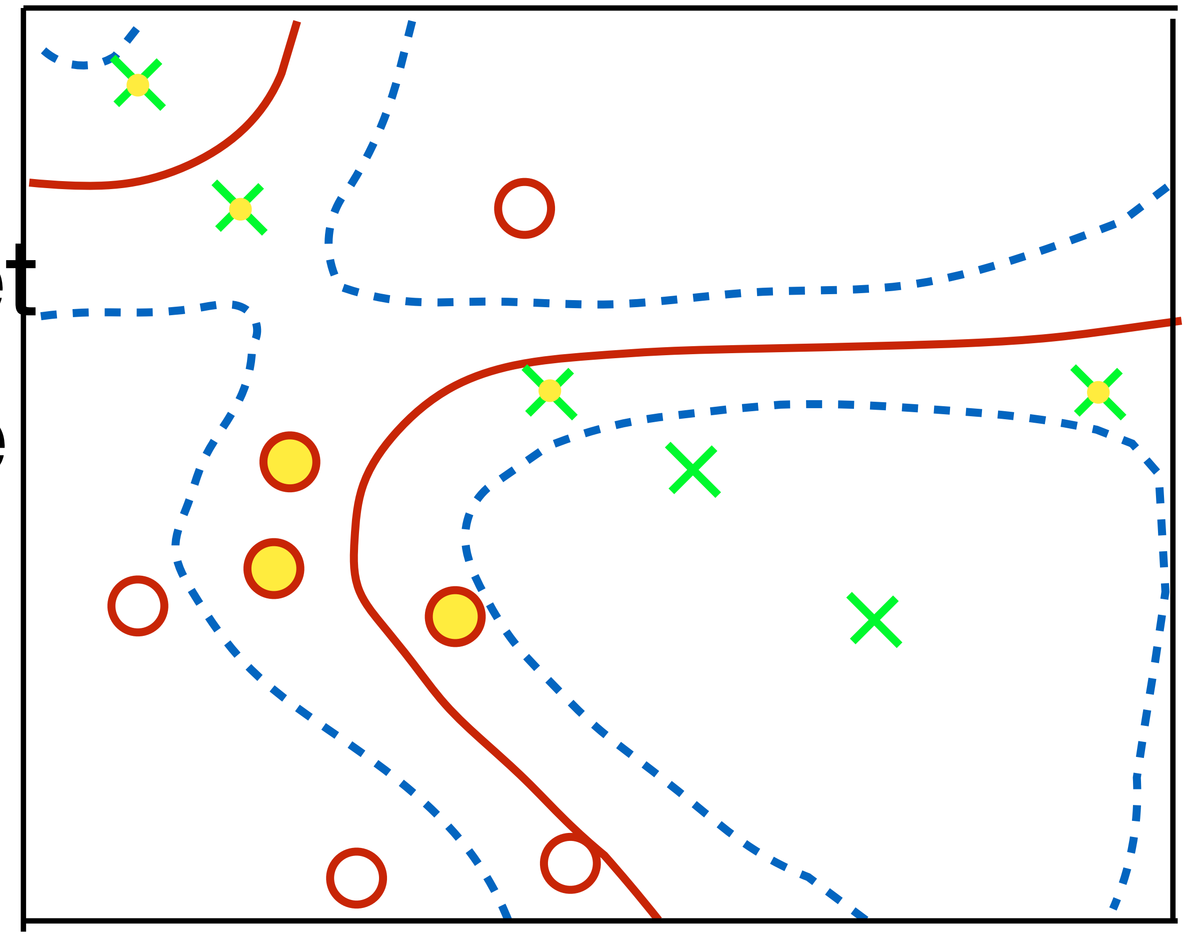
Memory Efficiency

Problem: training data grows continually in every learning round

Idea: constrain the number of training samples

When new point arrives:

- check whether it should be added to the Active Set
- use the entropy difference to rate the new point



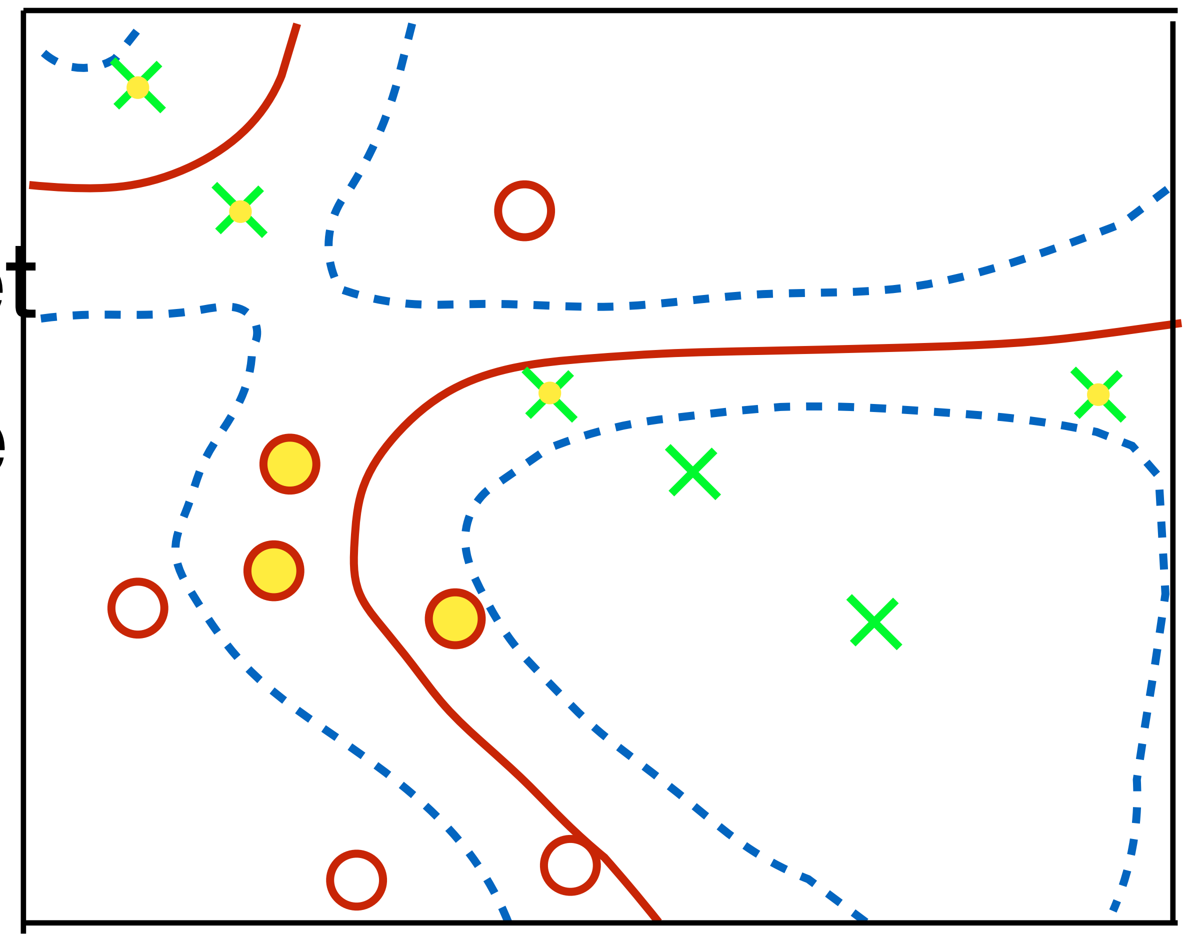
Memory Efficiency

Problem: training data grows continually in every learning round

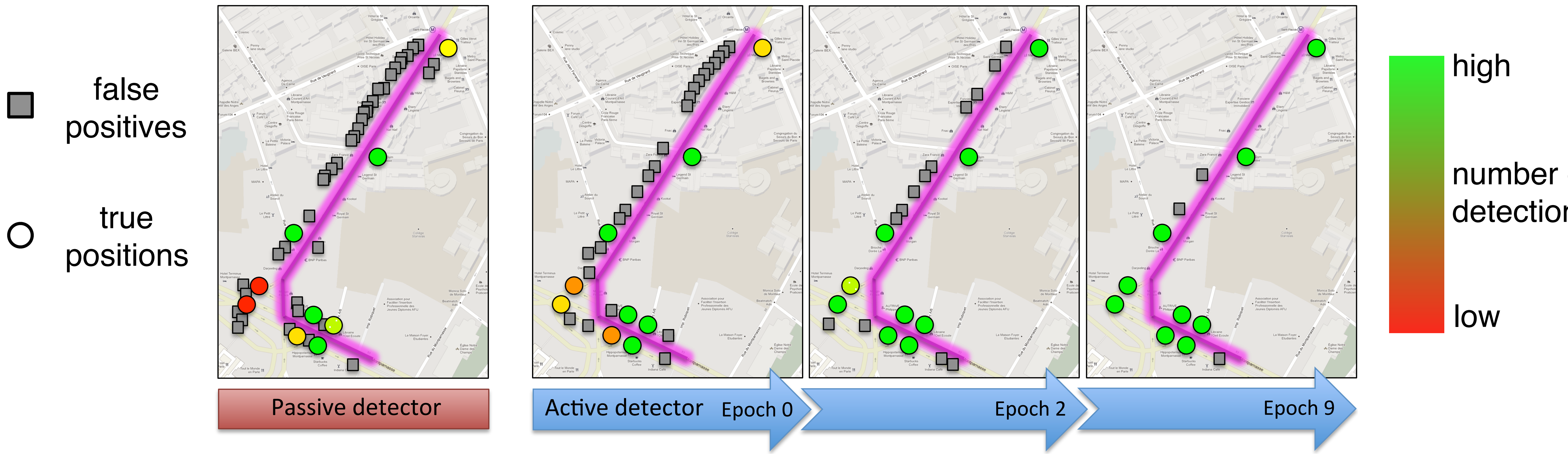
Idea: constrain the number of training samples

When new point arrives:

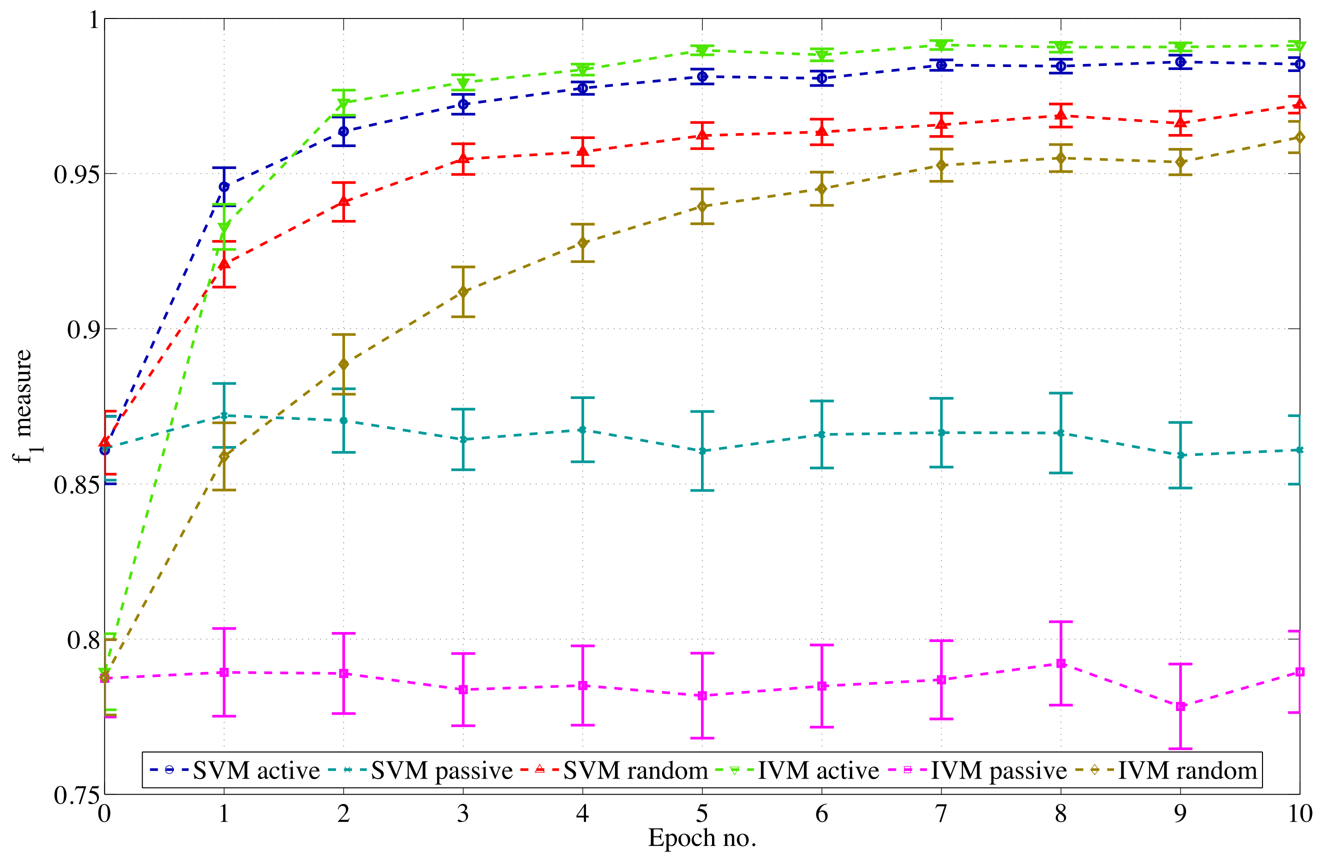
- check whether it should be added to the Active Set
- use the entropy difference to rate the new point
- throw out the point with the lowest rating



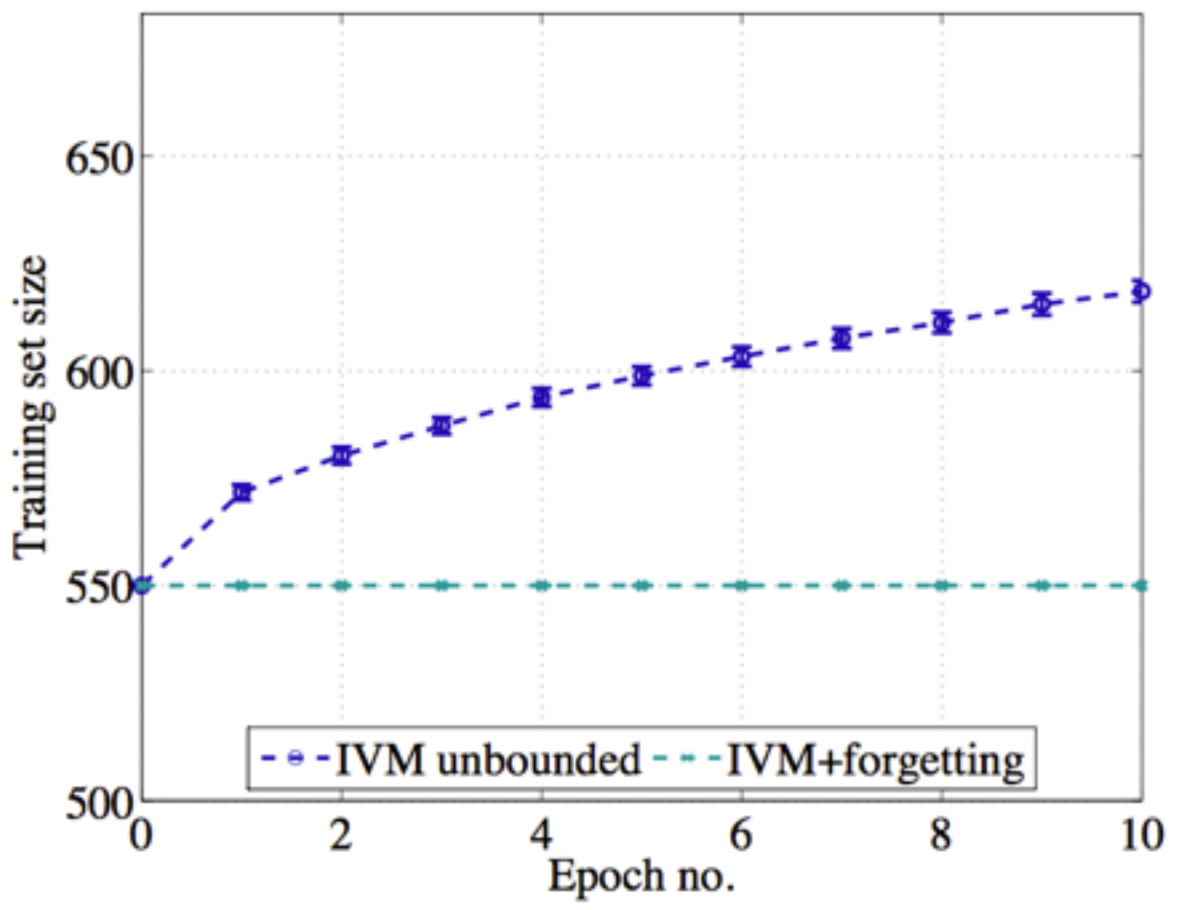
Semantic Mapping: Results (Learning)

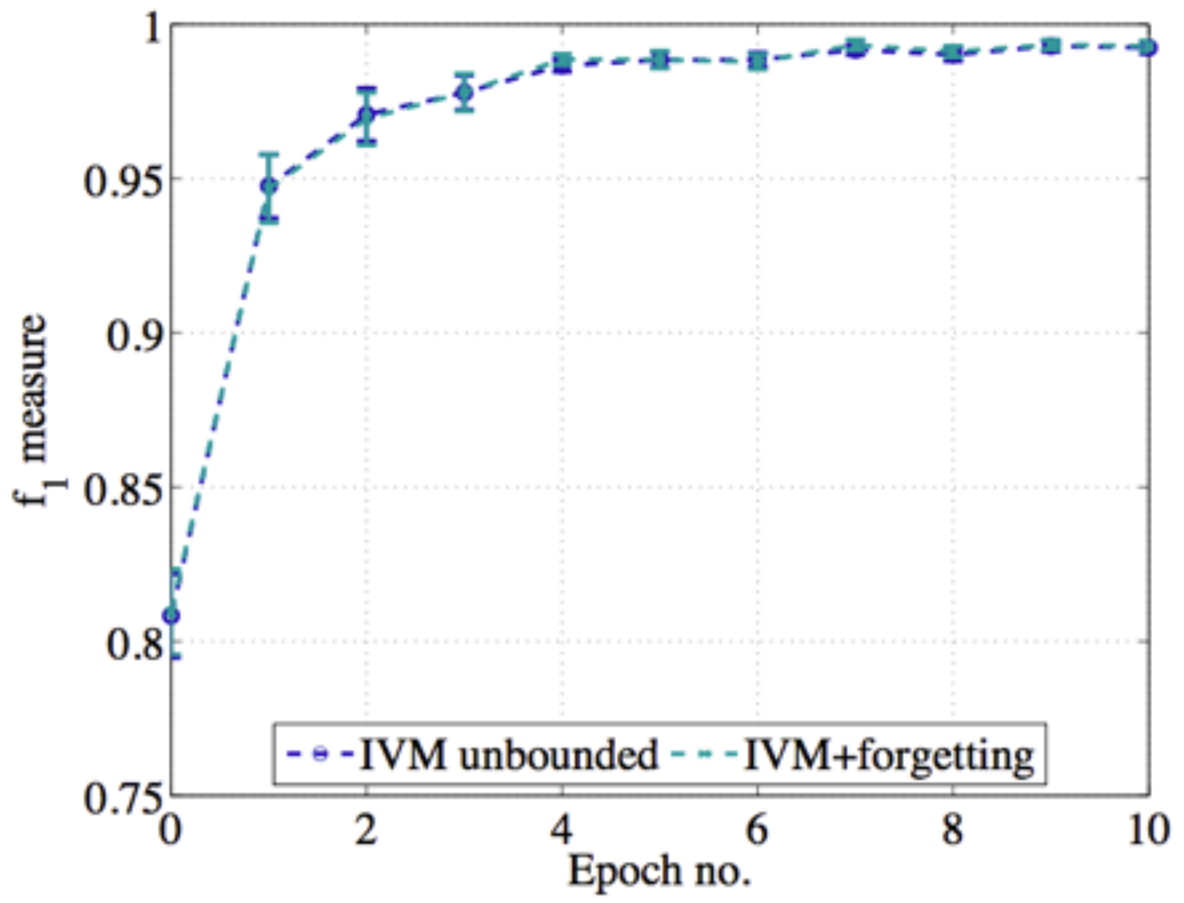


- IVM “overtakes” and stays better than SVM
- Active learning better than passive learning
- Random selection is not better



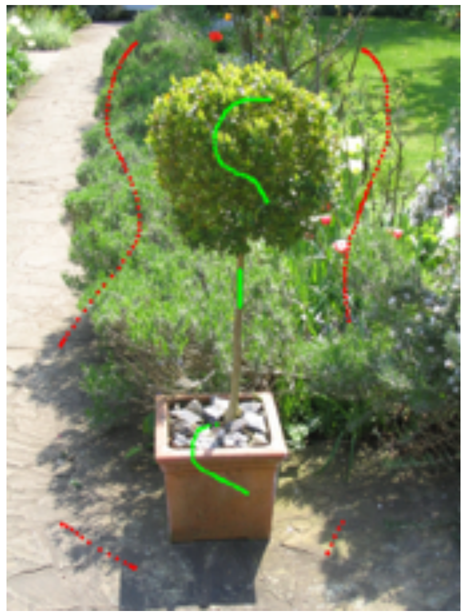
Semantic Mapping: Results (Forgetting)

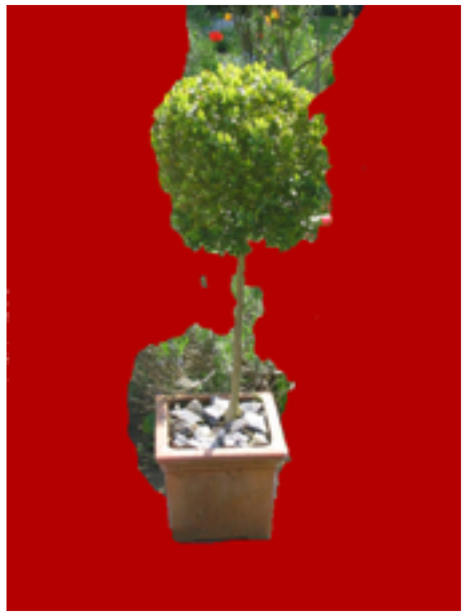


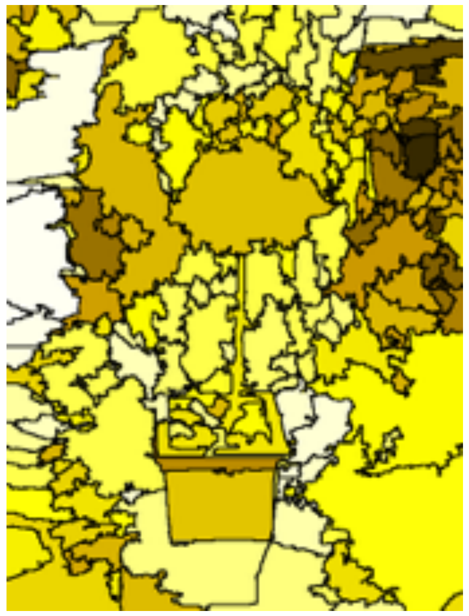


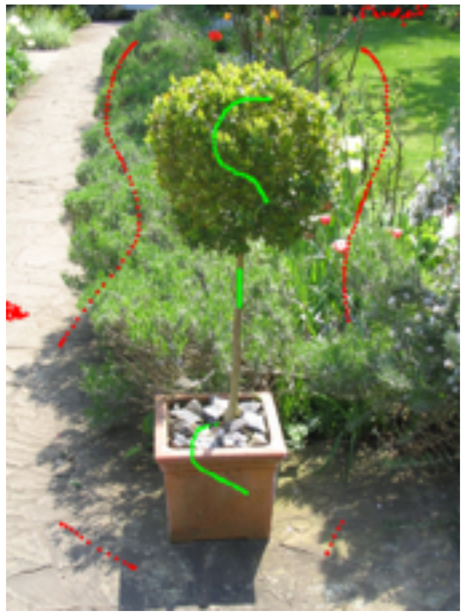
Forgetting has almost no influence on the classification result!

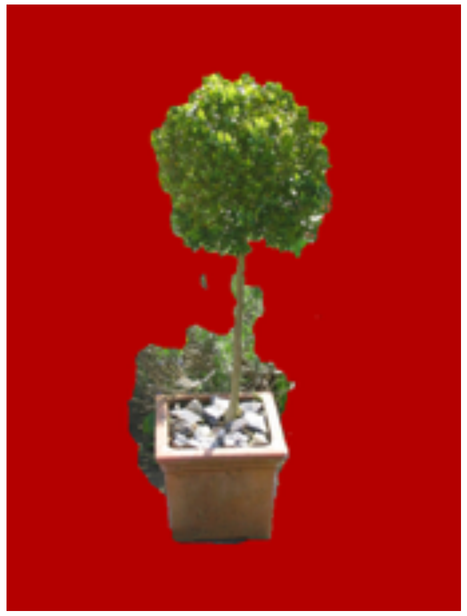
Application: Interactive Image Segmentation

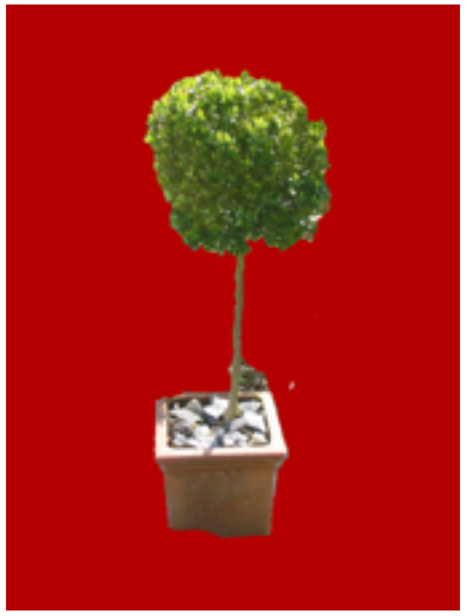

Initial scribbles

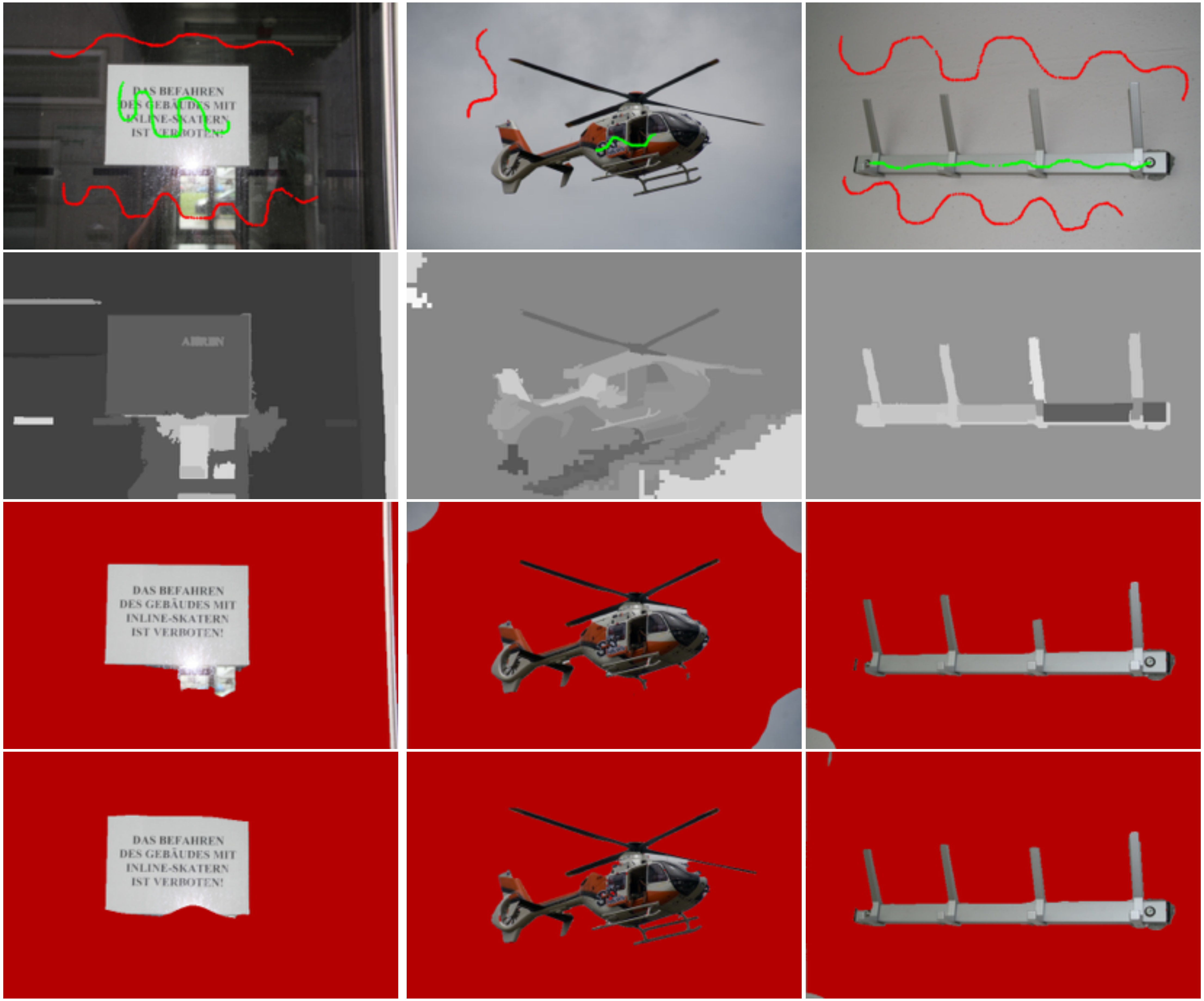

First segmentation


Uncertainties


New scribbles


Next segmentation


Final segmentation


DAS BEFAHREN
DES GEBÄUDES MIT
INLINE-SKATERN
IST VERBOTEN!