

# Feature Matching and Robust Fit

Giacomo Boracchi

CVPR USI, April 24 2020

# Object Recognition by Computer Vision Features

## Estimating Image Correspondences

- Extract features from each image (last time...)
  - Keypoint detection
  - Descriptor Computation
- Match features between images
- Prune matches and then perform triangulation  
detect objects / stitching ...

# Object Recognition by Computer Vision Features

## Estimating Image Correspondences

- Extract features from each image
  - Keypoint detection
  - Descriptor Computation
- Match features between images (today)
- Prune matches and then perform triangulation  
detect objects / stitching ...

# The General Picture



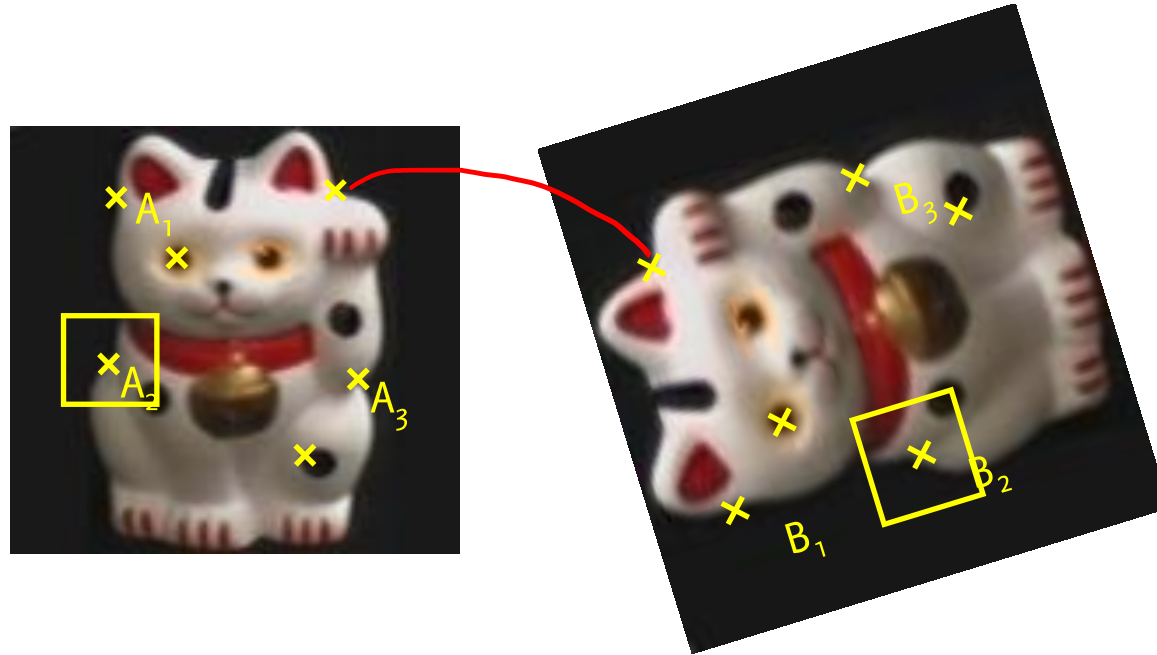


# The General Picture



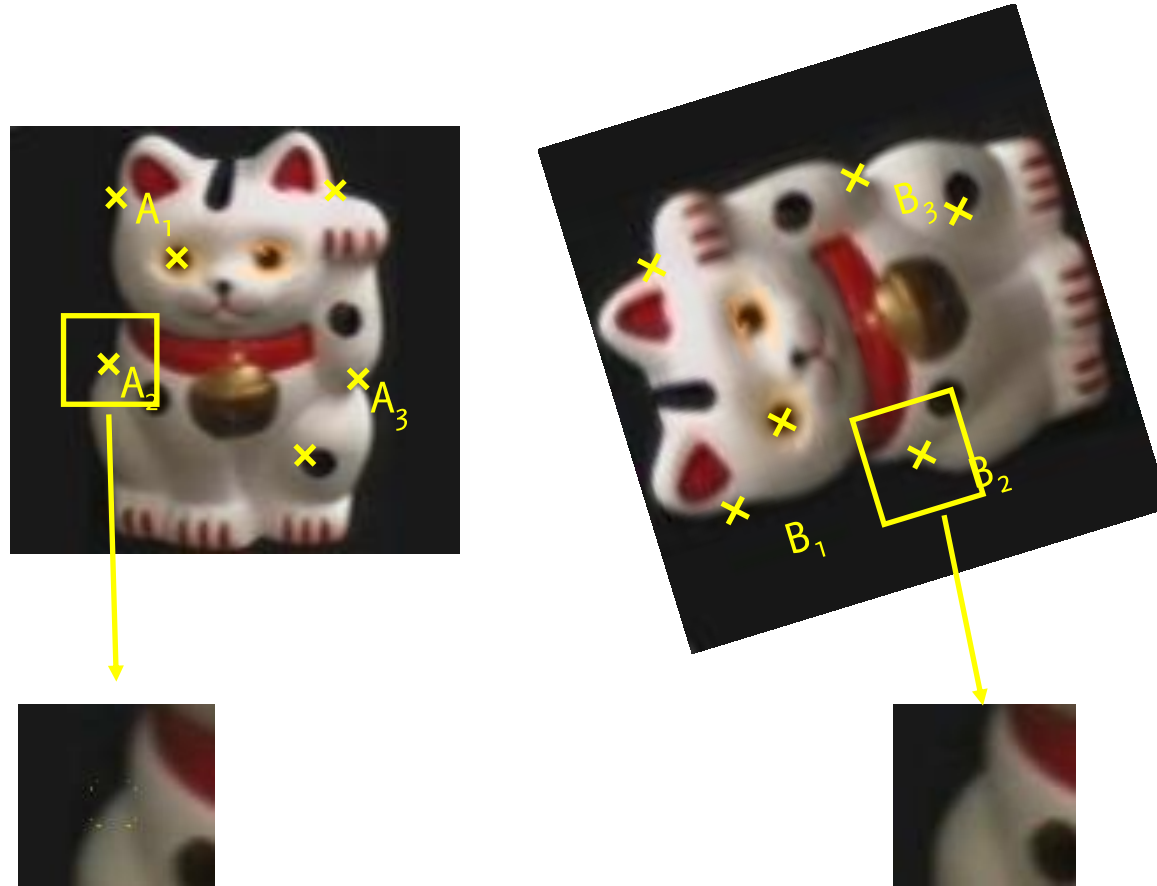
1. Find a set of distinctive key-points

# The General Picture



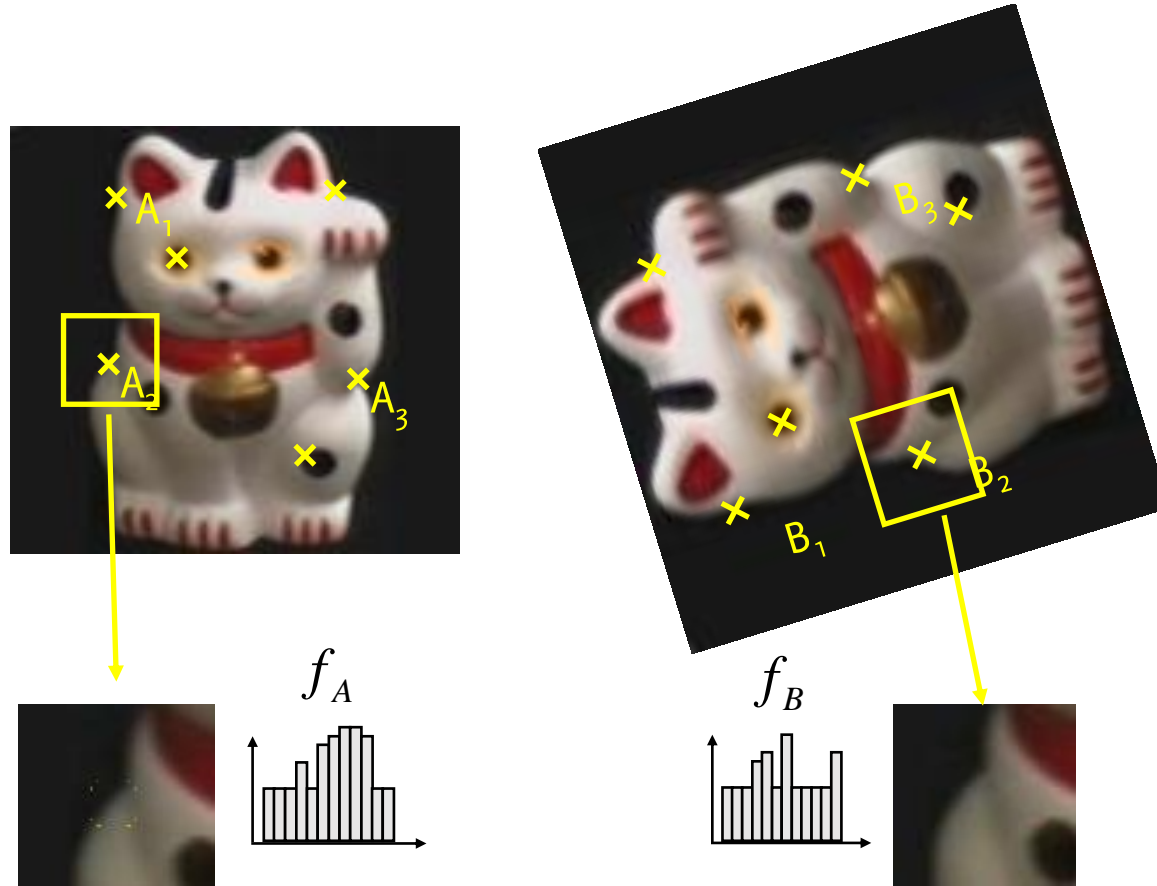
1. Find a set of distinctive keypoints
2. Define a region around each keypoint

# The General Picture



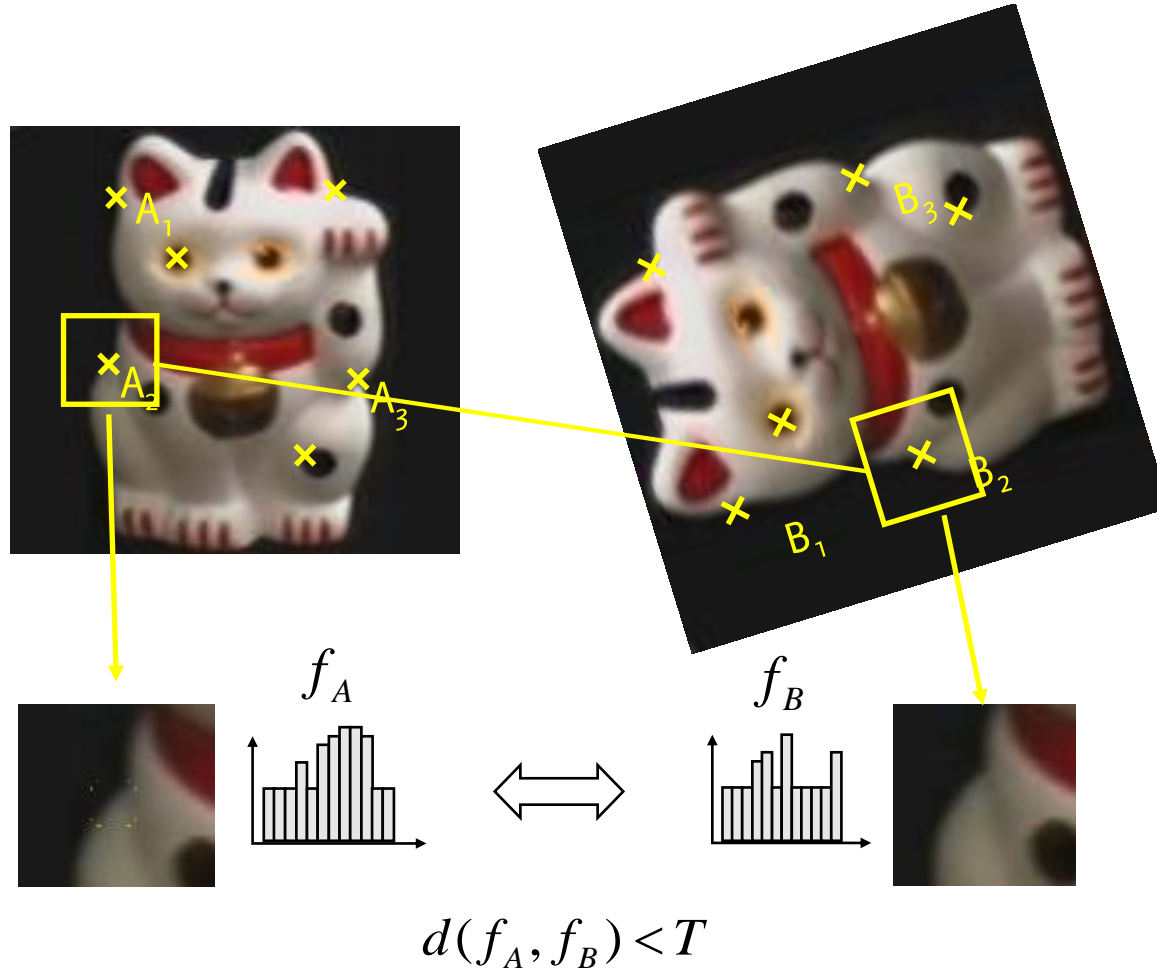
1. Find a set of distinctive keypoints
2. Define a region around each keypoint
3. Extract and normalize the region content

# The General Picture



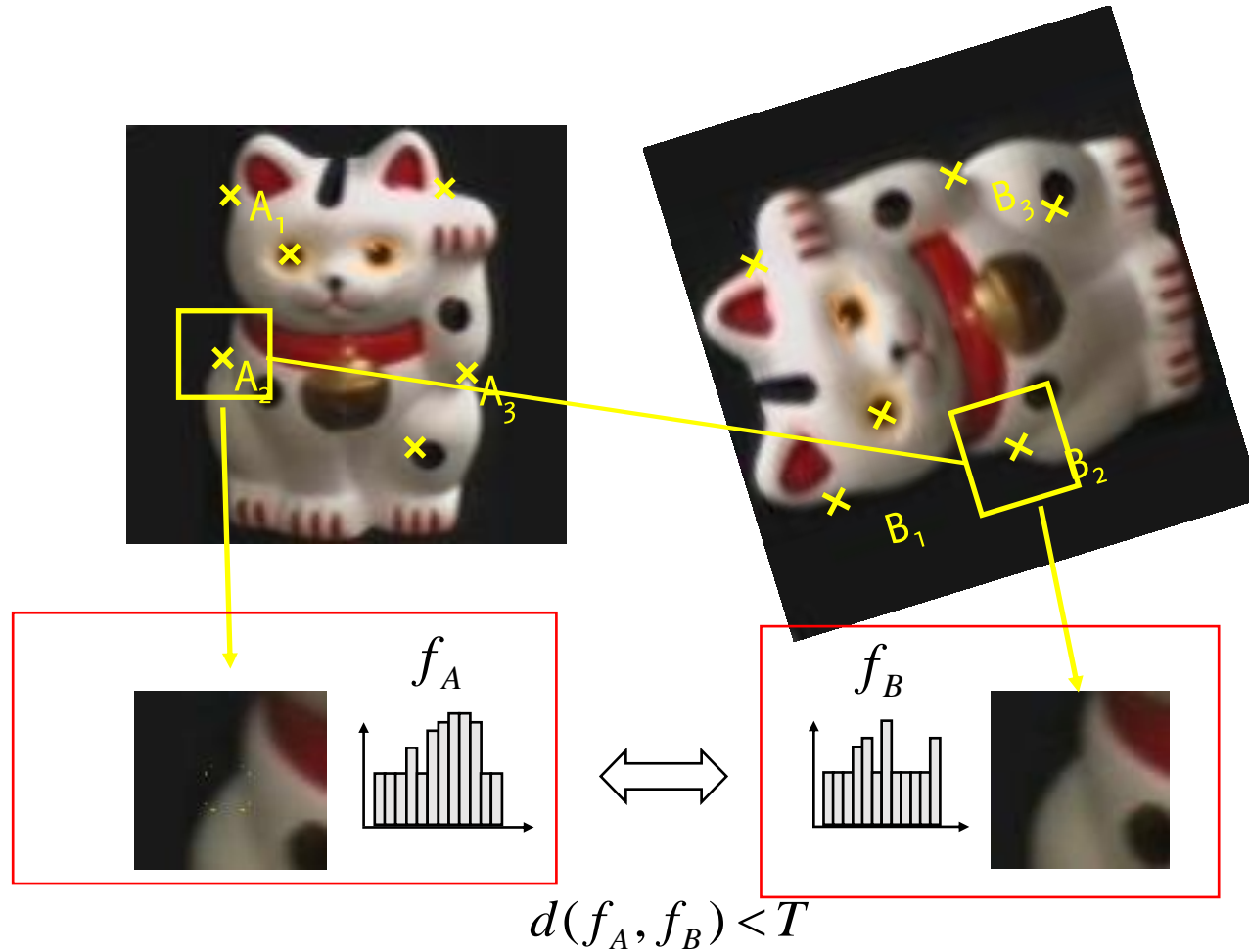
1. Find a set of distinctive keypoints
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region

# The General Picture



1. Find a set of distinctive keypoints
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

# The General Picture



1. Find a set of distinctive keypoints

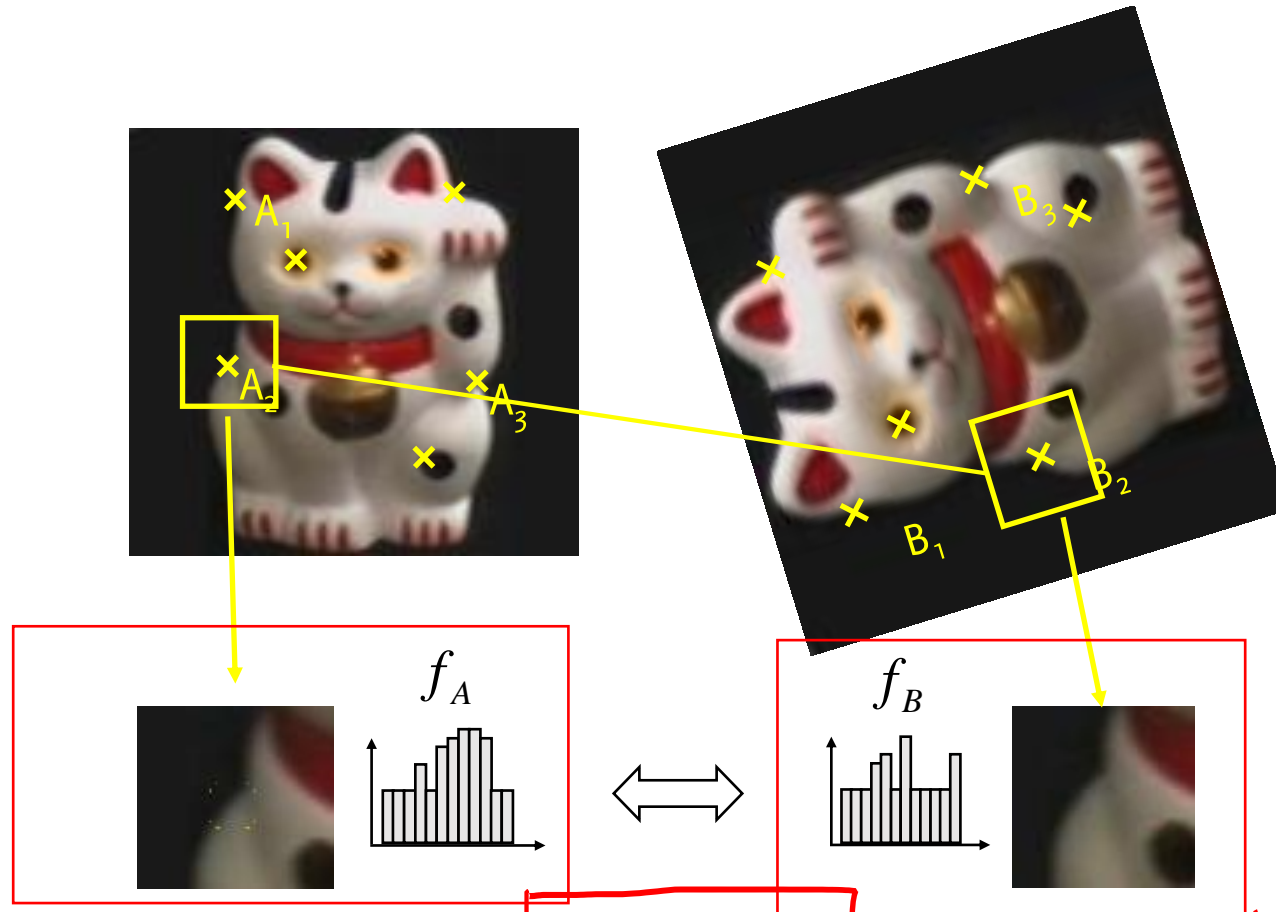
2. Define a region around each keypoint

3. Extract and normalize the region content

4. Compute a local descriptor from the normalized region

5. Match local descriptors

# The General Picture

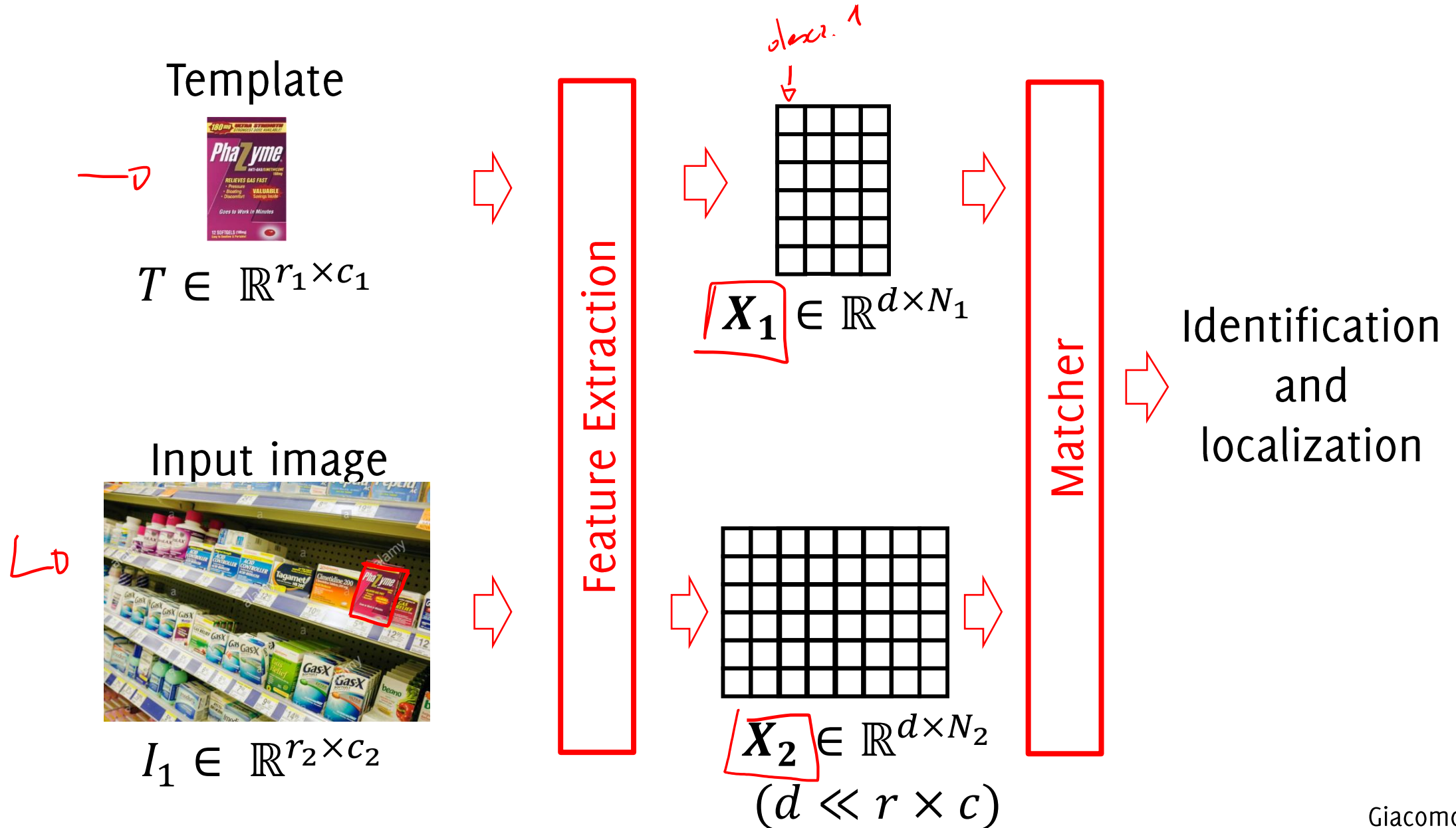


1. Find a set of distinctive keypoints
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

$$d(f_A, f_B) < T$$

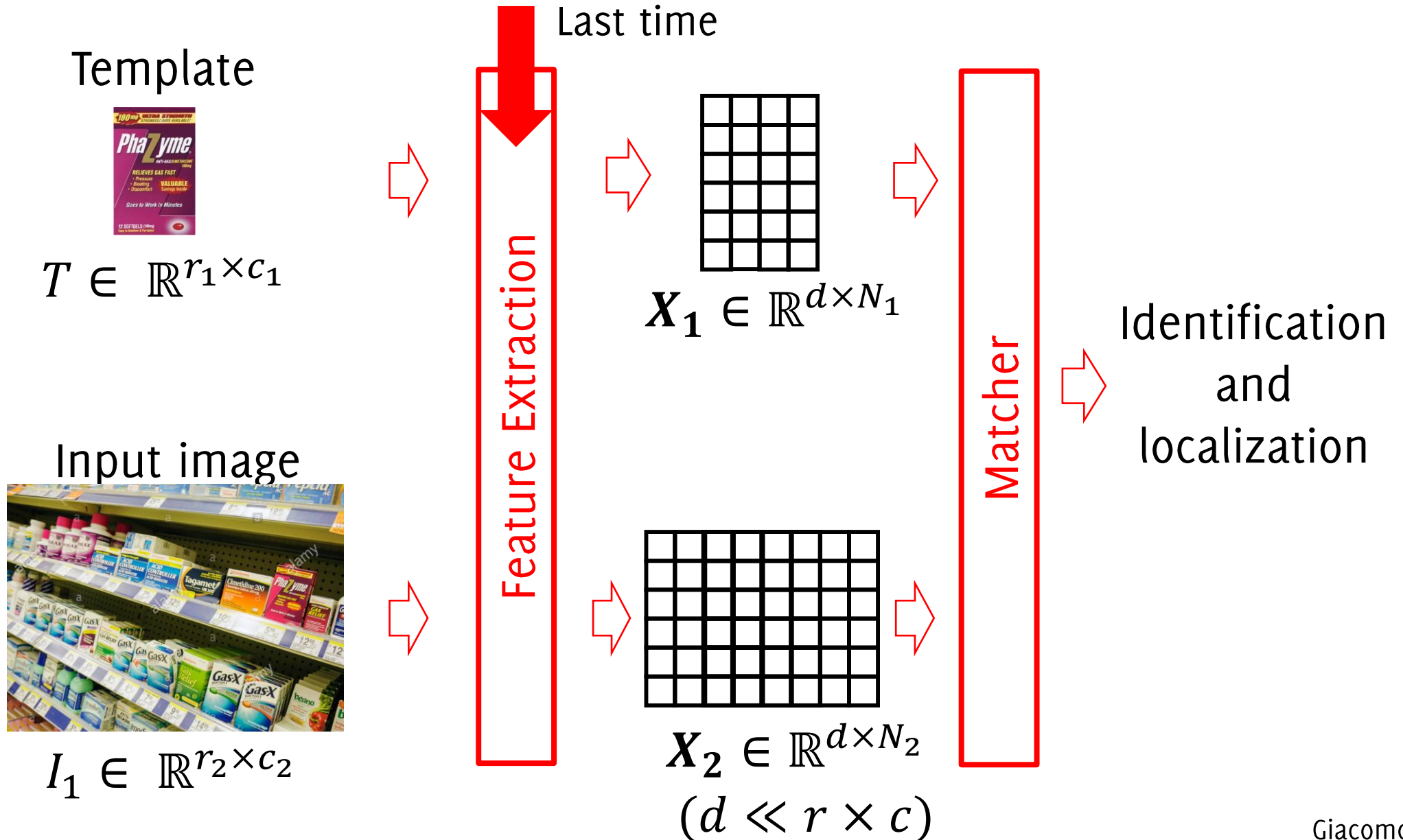
*( $f_A, f_B$ ) over a network?*

# Object Recognition by Feature Extraction

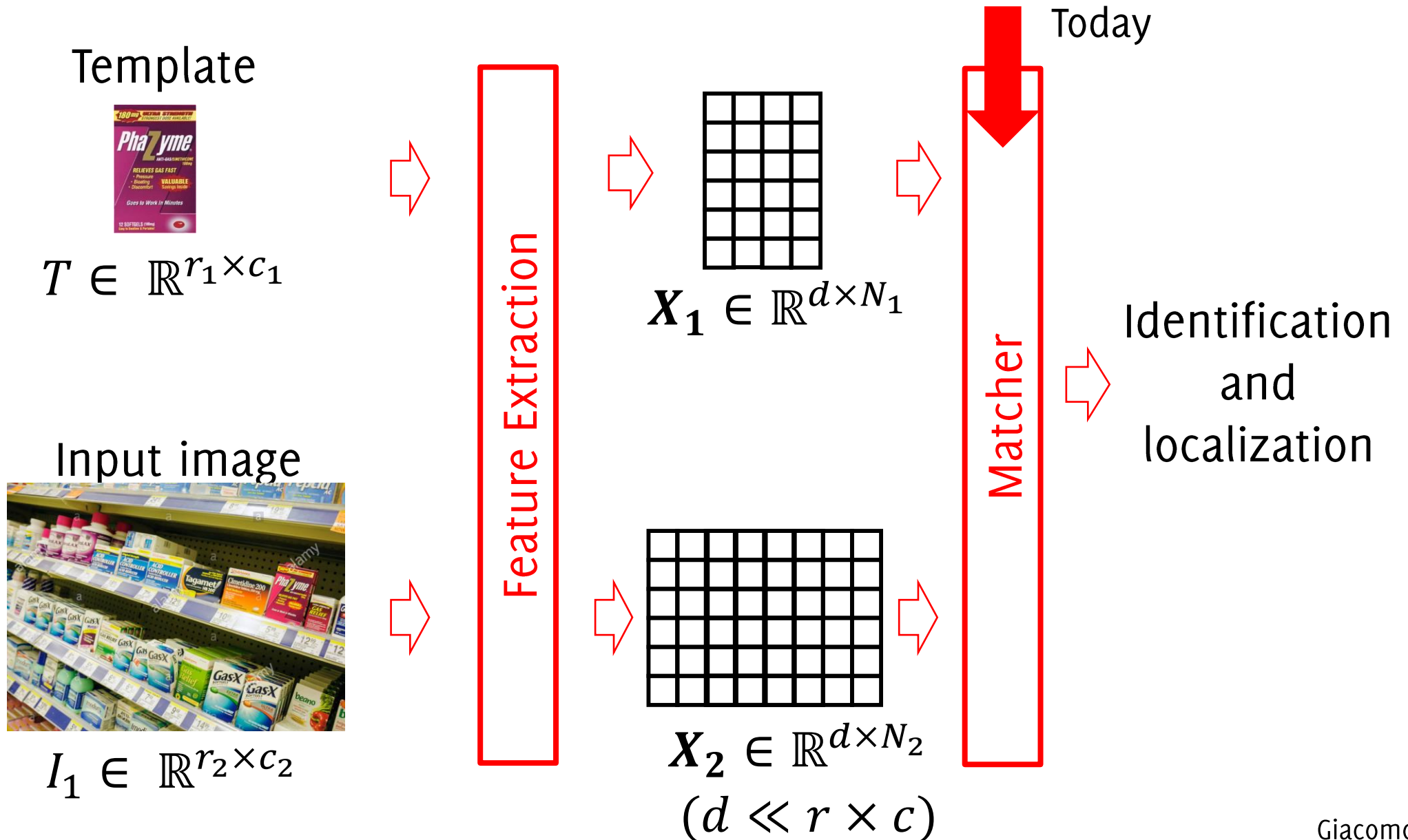




# Object Recognition by Feature Extraction



# Object Recognition by Feature Extraction



# Feature Matching

# Object Recognition by Computer Vision Features

## Estimating Image Correspondences

- Extract features from each image
  - Keypoint detection
  - Descriptor Computation
- Match features between images
- Prune matches and then perform triangulation  
detect objects / stitching ...

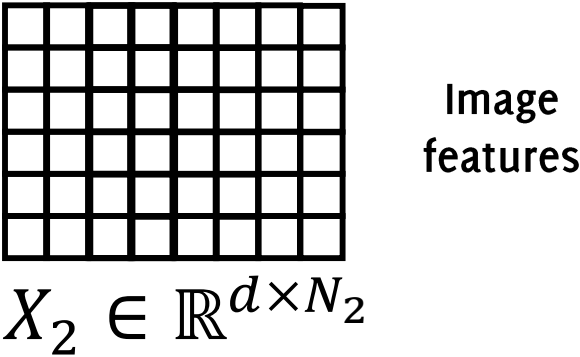
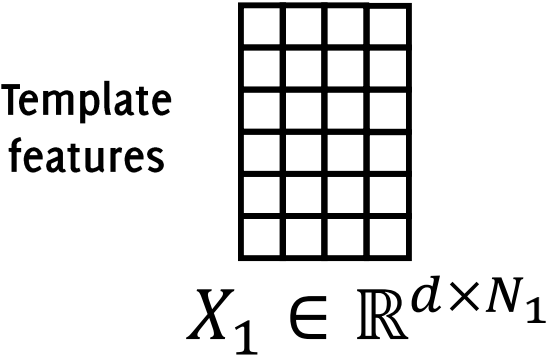
# The General Approach to Feature Matching

When comparing two images  $I_1$  and  $I_2$  one typically:

- **Independently extract SIFT features** (keypoint + descriptor) from **each image**
- **For each descriptor** in  $I_1$  we look for the most similar, descriptors in  $I_2$ , i.e., we compute its **nearest neighborhood** in  $I_2$  in terms of the **Euclidean distance**
- **Matches are confirmed** when the **distance between the two descriptors is below a threshold**. A matched feature connects both keypoint location and scale.
- After having iterated the procedure for all the points, **global criteria to discard wrong matches** can be implemented
- **Objects are detected by clustering** (grouping) **matches**, to this purpose, some global criteria is enforced

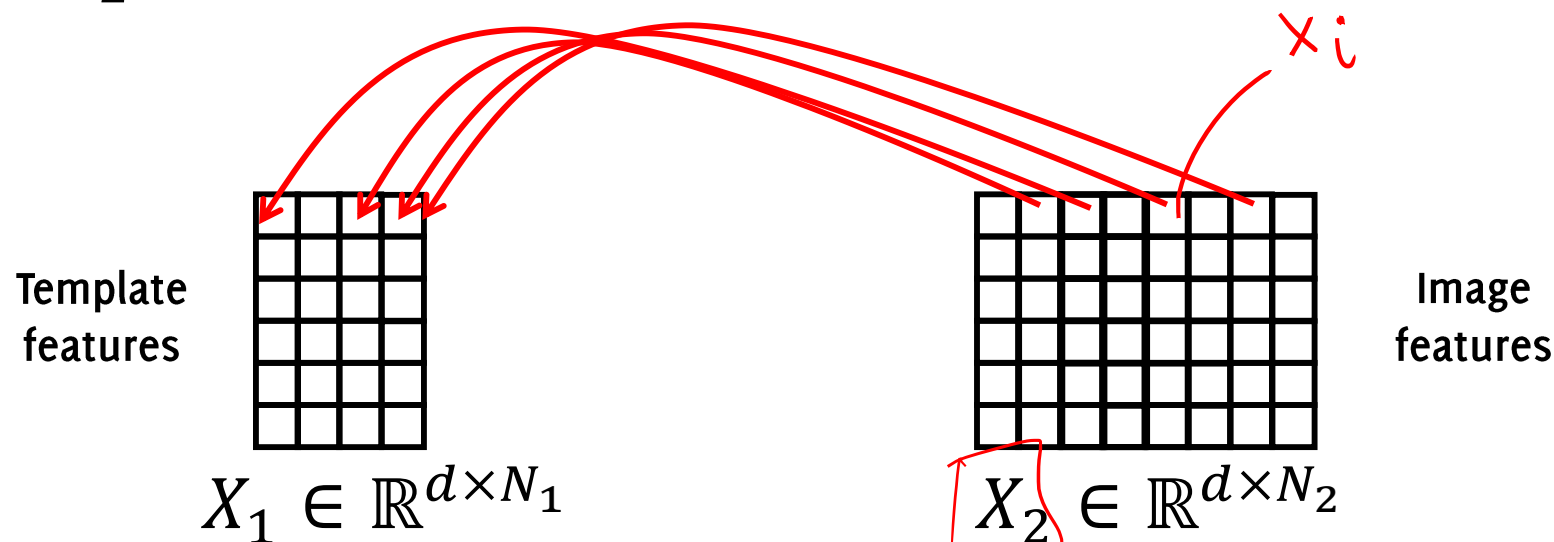
# Feature Matching

**Input:**  $X_1$  and  $X_2$ , features extracted from  $T$  and  $I$



# Feature Matching

**Input:**  $X_1$  and  $X_2$ , features extracted from  $T$  and  $I$



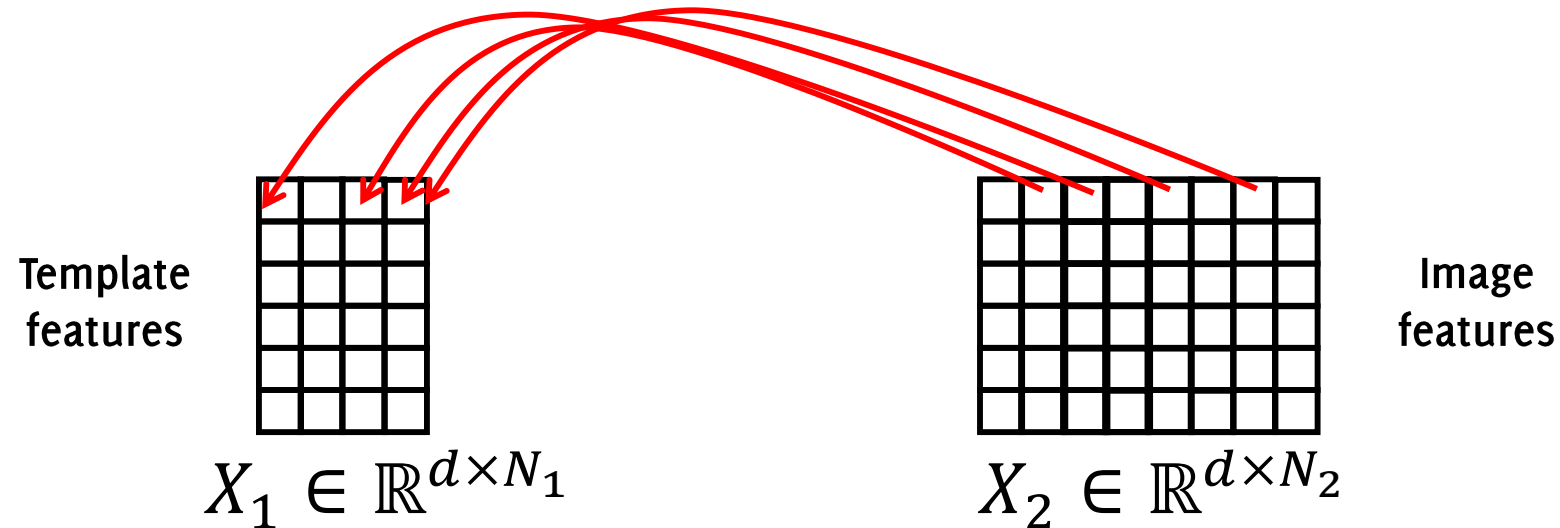
**Goal:** match image features on the template features.

Estimate each image feature  $\mathbf{x}_i \in X_2$ , the template feature  $\mathbf{x}_j^i \in X_1$  minimizing distance the Euclidean distance

$$\mathbf{x}_j^i = \operatorname{argmin}_{\mathbf{x}_j \in X_1} \left( \underbrace{\|\mathbf{x}_i - \mathbf{x}_j\|_2}_{\ni X_2} \right) \quad *$$

# Feature Matching

**Input:**  $X_1$  and  $X_2$ , features extracted from  $T$  and  $I$



**Goal:** match image features on the template features.

Estimate  $e$   
minimizing

This is the **Nearest-Neighbor Matching Problem**  
(on high-dimensional data)

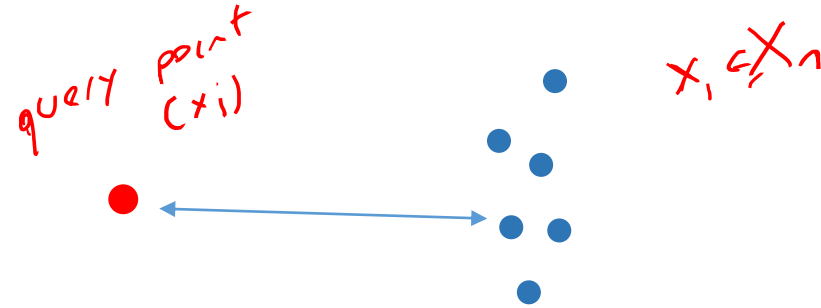
A central problem in CV, ML, document retrieval,  
data analysis, bioinformatics, compression

$X_1$



# Feature Matching

One option is linear search: exhaustively looking for the closest point in a loop



Finding nearest neighbor matches to high dimensional vectors of the training set is **one of the most computationally expensive part of CL and ML algorithms**, it is  $O(n)$  but training set are typically very large

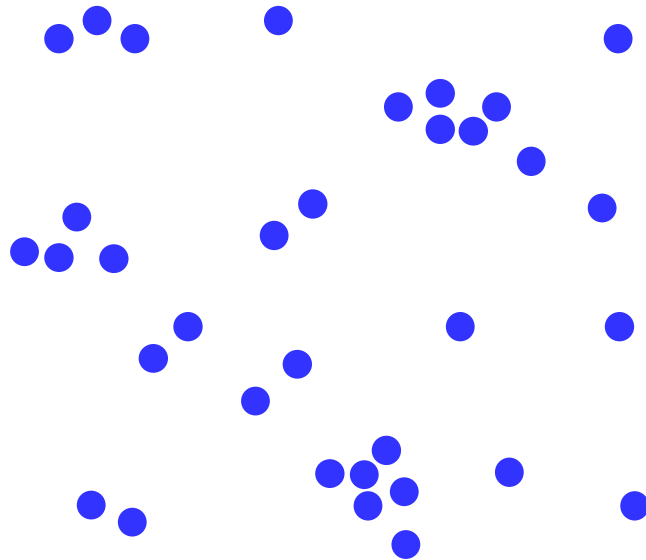
This is computationally infeasible in cases of practical interest, and **more efficient solutions are needed**

# K-d trees: the rationale

Data-structured approach: **organize data for efficiently searching nearest neighbor.**

**K-d trees** are constructed by **recursive binary splits** on each marginal component (typically based on the median of the covariate population)

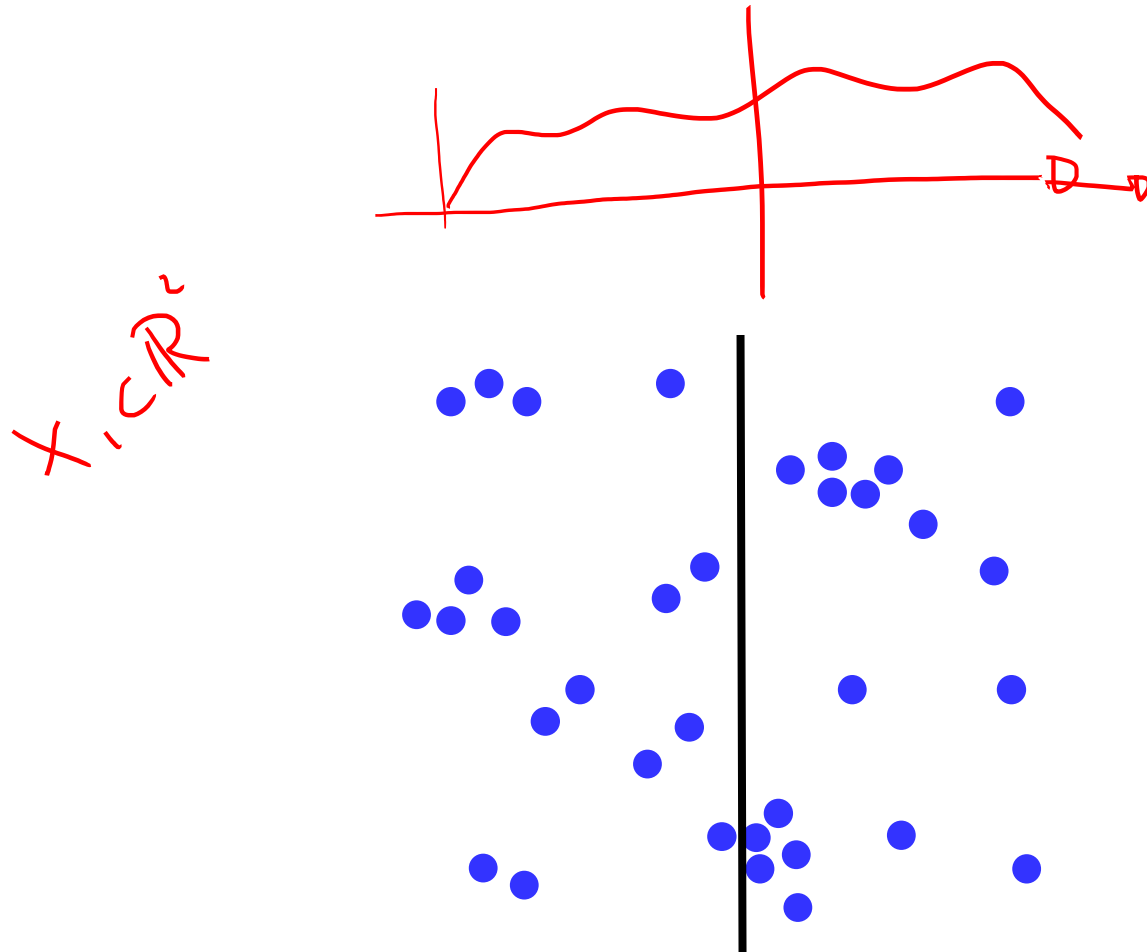
$X_1$



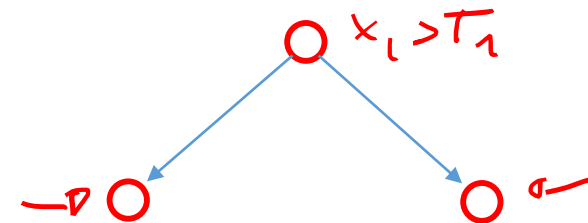
Data where to  
build a k-d tree

# K-d trees: the rationale

**K-d trees** are constructed by **recursive binary splits** on each marginal component (typically based on the median of the covariate population)

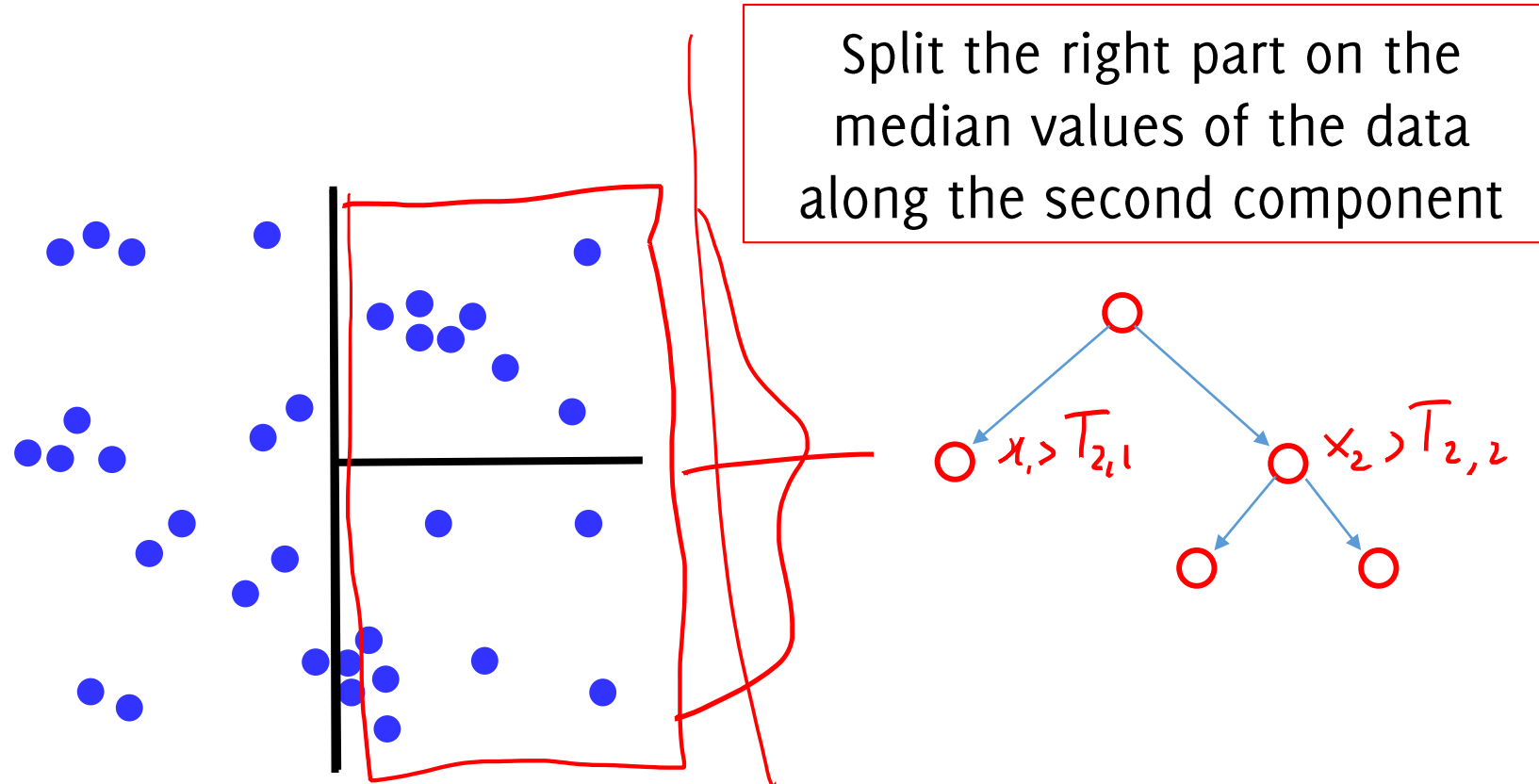


Split the space on the median values of the data along the first component



# K-d trees: the rationale

**K-d trees** are constructed by **recursive binary splits** on each marginal component (typically based on the median of the covariate population)

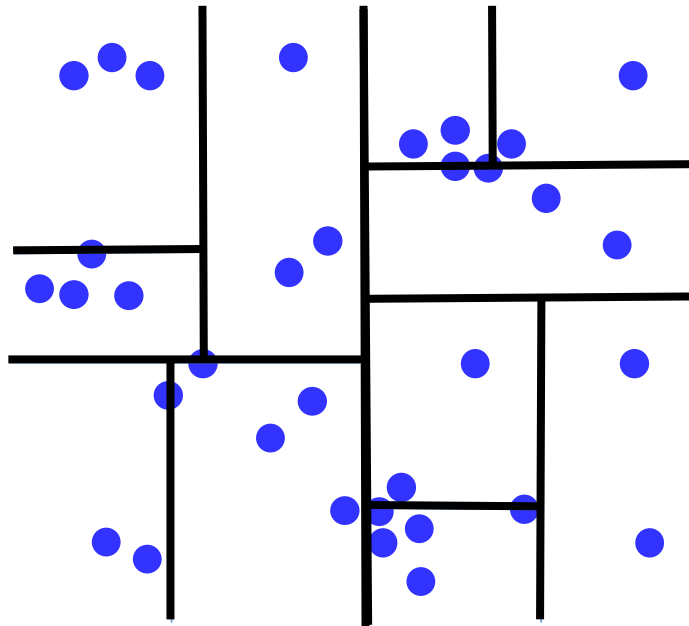


# K-d trees: the rationale

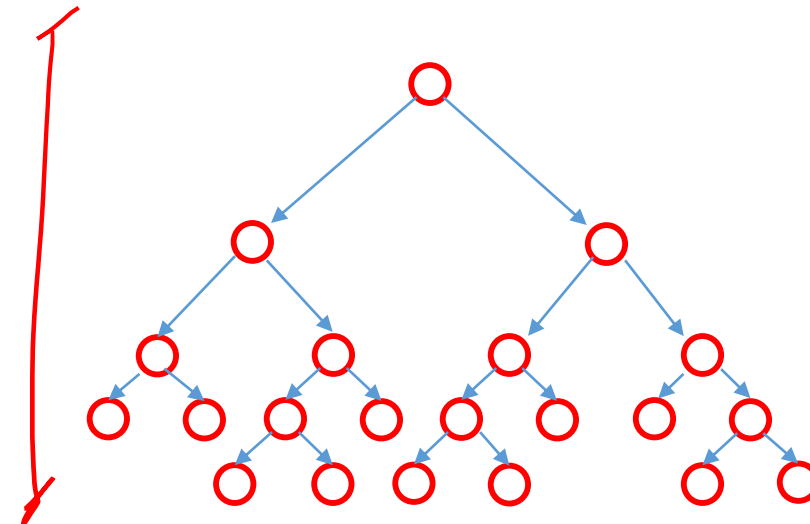
**K-d trees** are constructed by **recursive binary splits** on each marginal component (typically based on the median of the covariate population)

This scheme yields a tree with node splits based on  $[\mathbf{x}]_i$

$n$



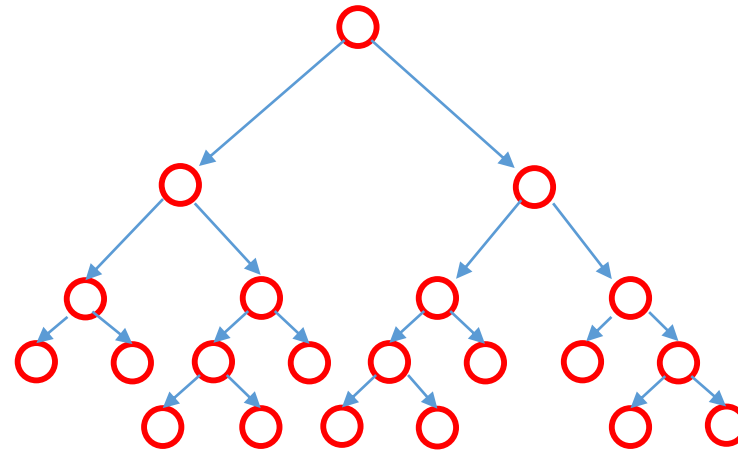
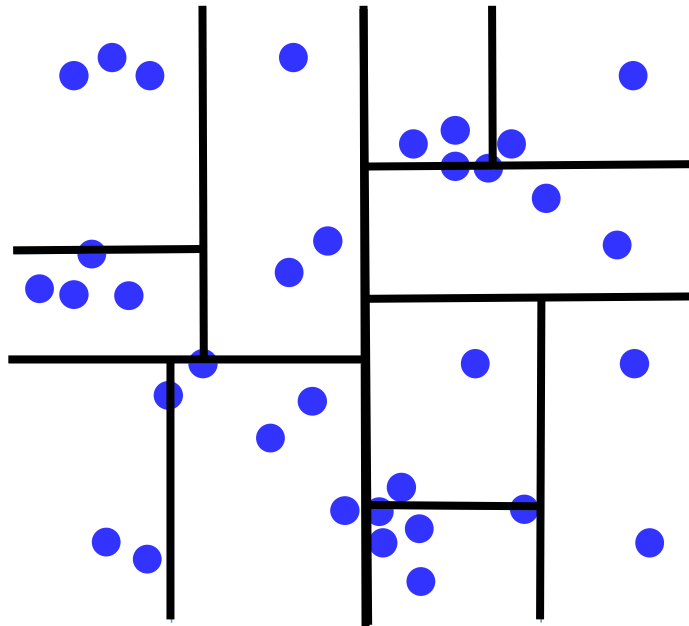
$\log_2(n)$



# Nearest Neighbor (NN) search over K-d trees

NN search in K-d trees is very efficient since it consists in:

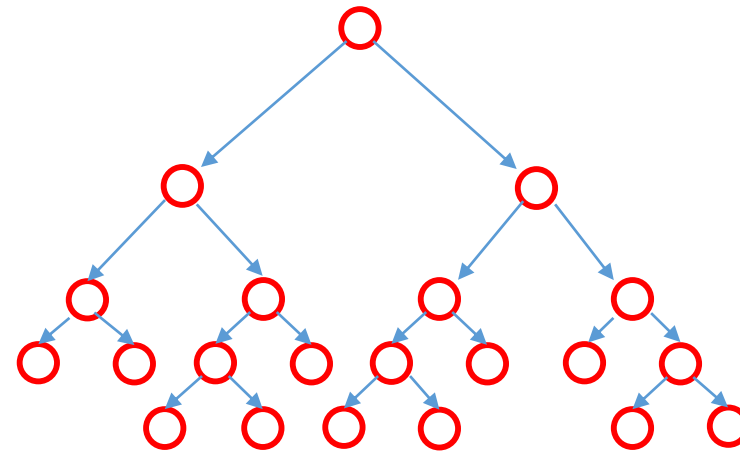
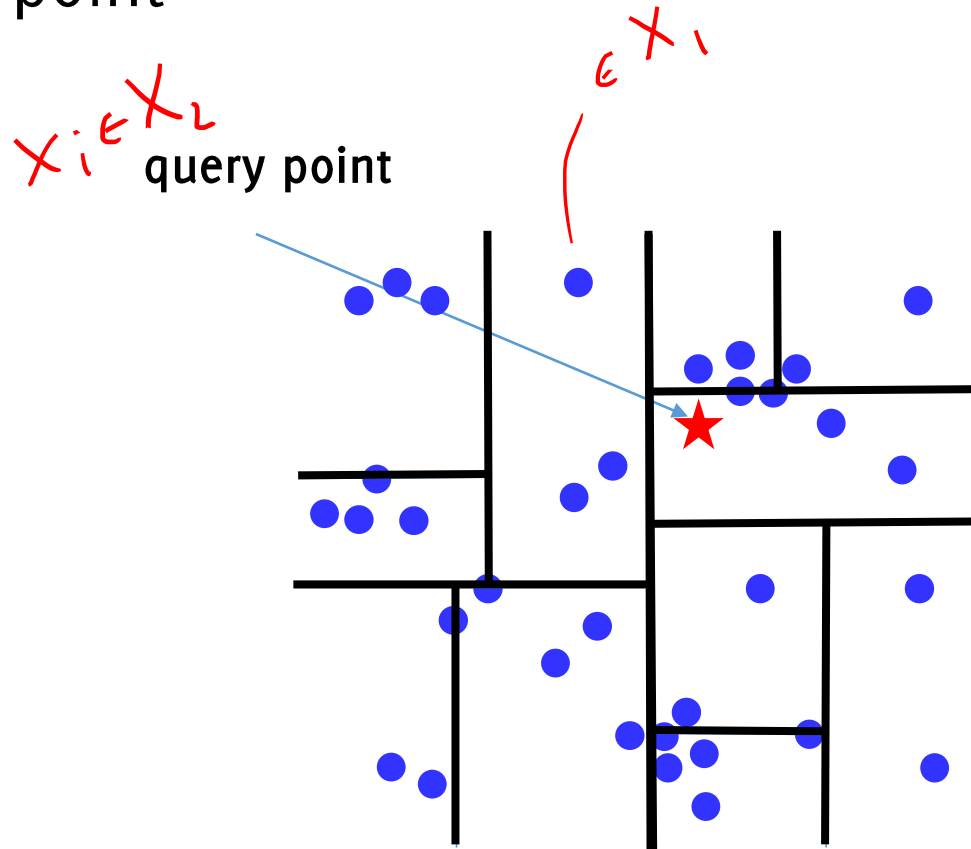
1. walking through the tree and finding the leaf containing the query point



# Nearest Neighbor (NN) search over K-d trees

NN search in K-d trees is very efficient since it consists in:

1. walking through the tree and finding the leaf containing the query point

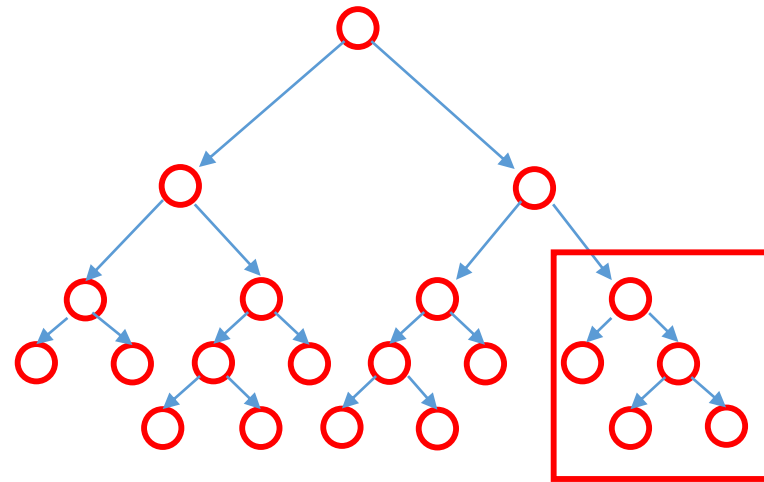
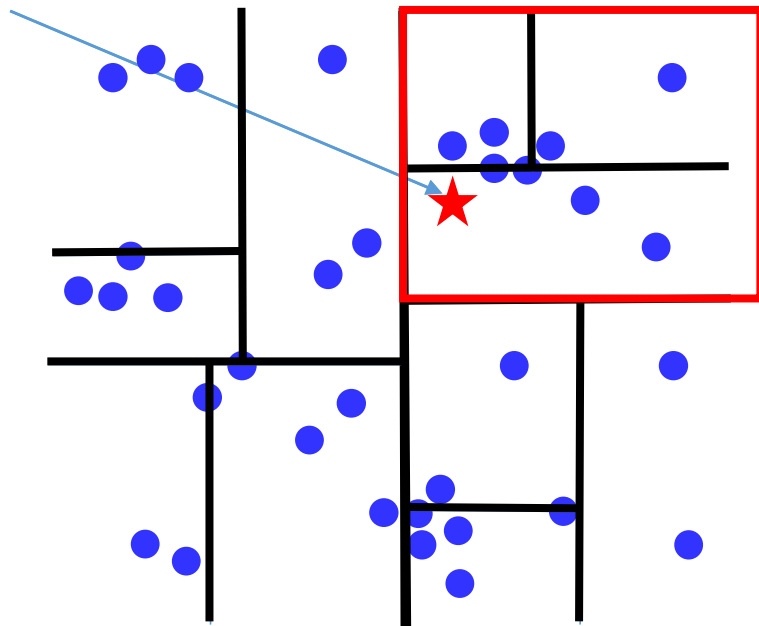


# Nearest Neighbor (NN) search over K-d trees

NN search in K-d trees is very efficient since it consists in:

1. walking through the tree and finding the leaf containing the query point

query point

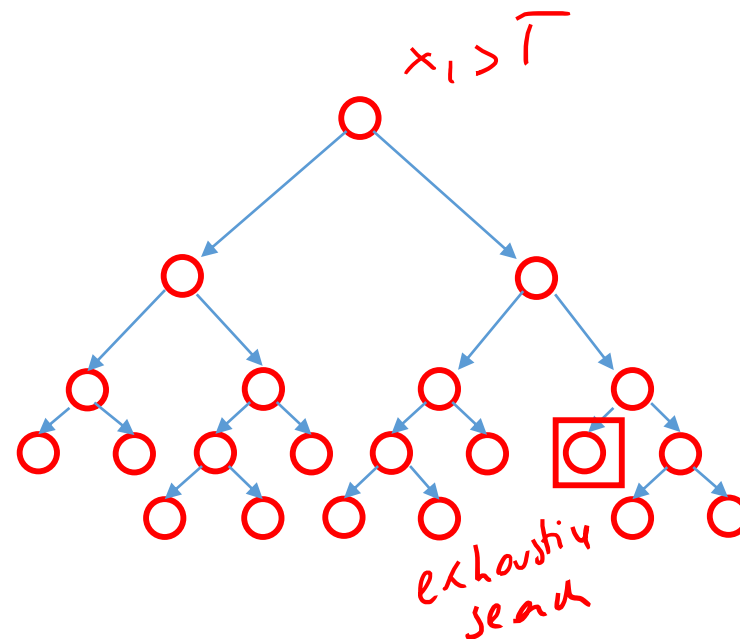
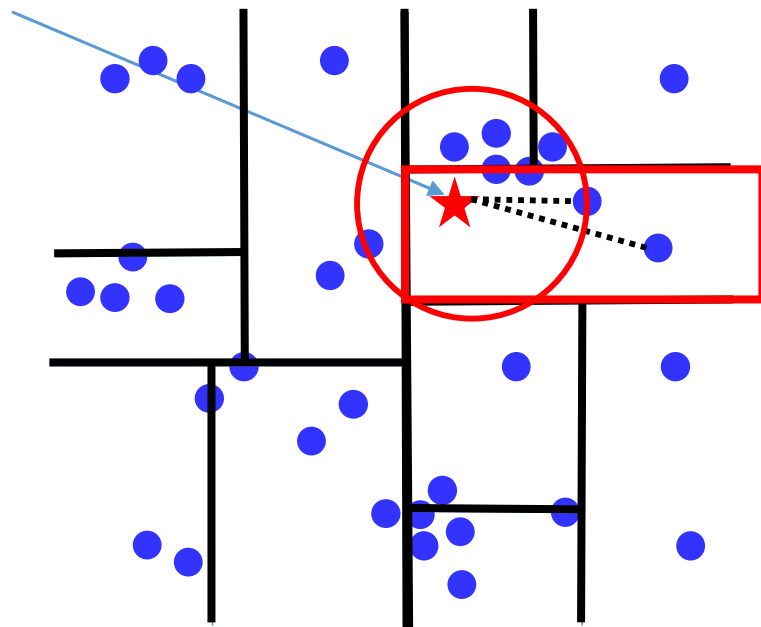




# Nearest Neighbor (NN) search over K-d trees

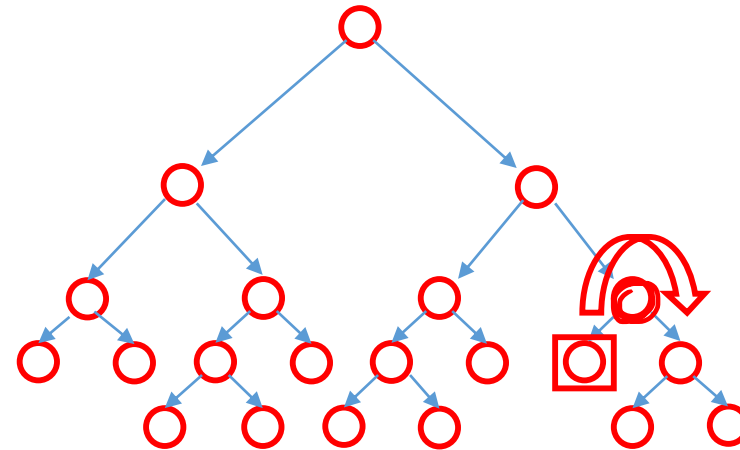
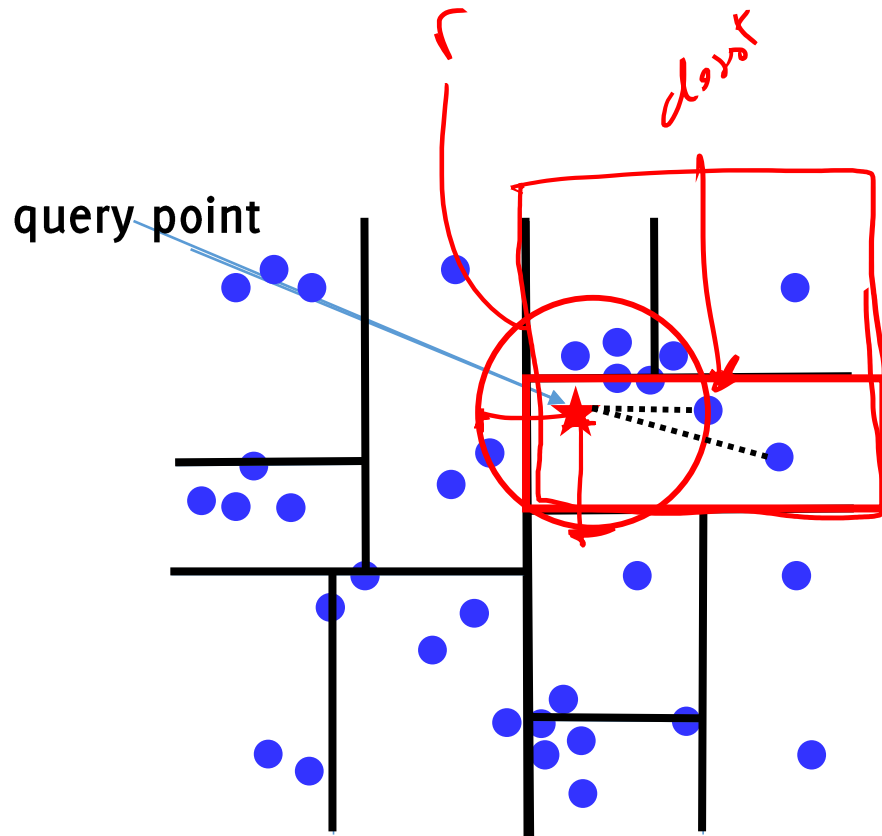
NN search in K-d trees is very efficient since it consists in:

1. walking through the tree and finding the leaf containing the query point
2. Computing the distance from the closest point in the leaf



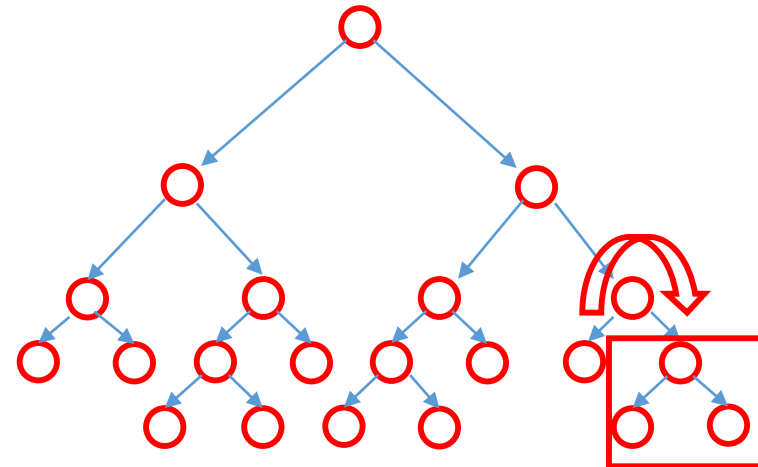
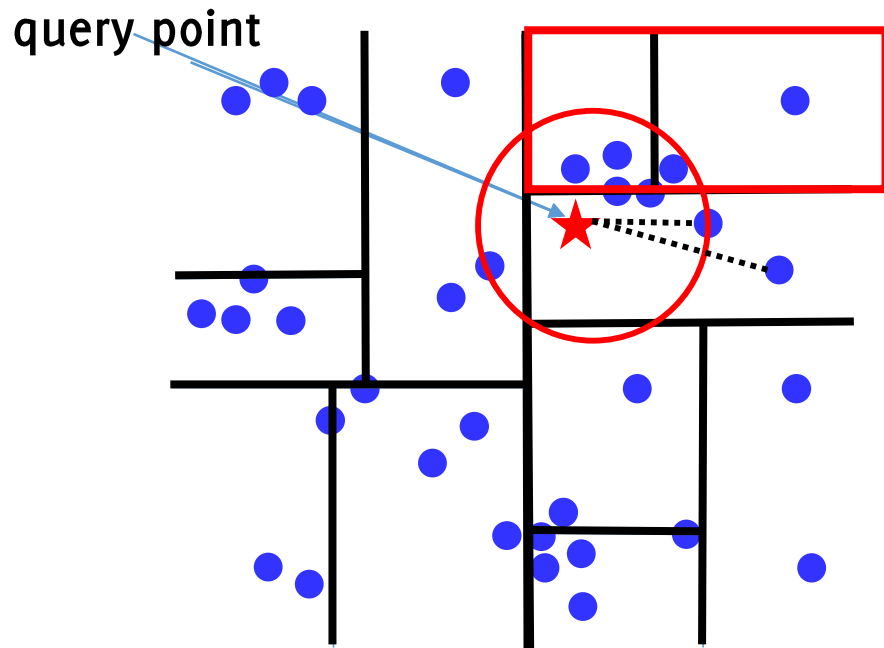
# NN search over K-d trees: the rationale

3. Backtrack the recursion on the leaves, to check whether the NN is in another leaf



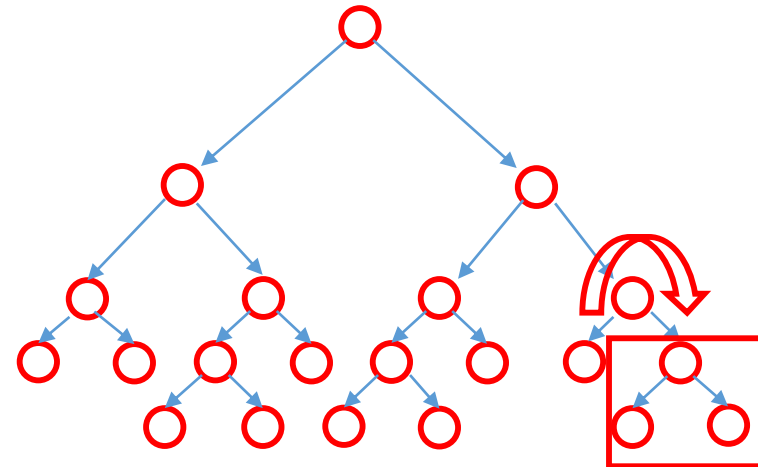
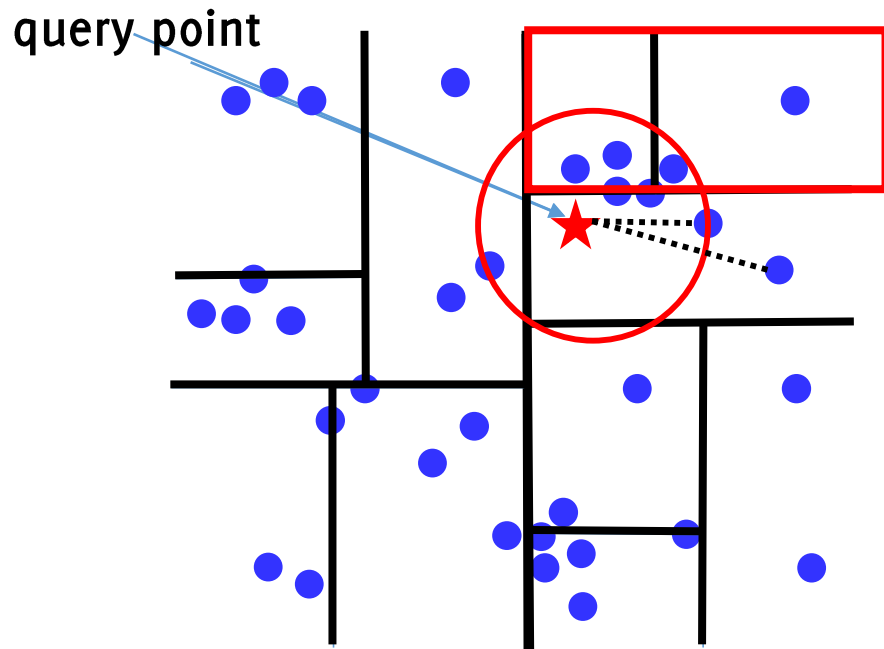
# NN search over K-d trees: the rationale

3. Backtrack the recursion on the leaves. For each leaf:
  - Intersect the hypersphere having radius of the closest point with the hyperplanes defining the leaf



# NN search over K-d trees: the rationale

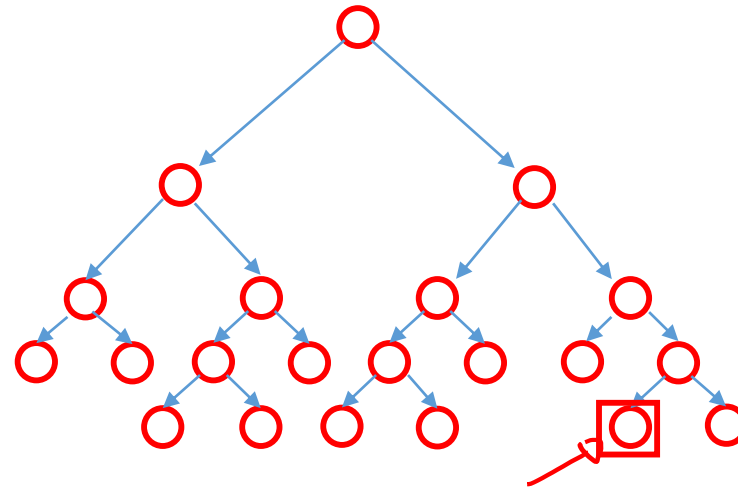
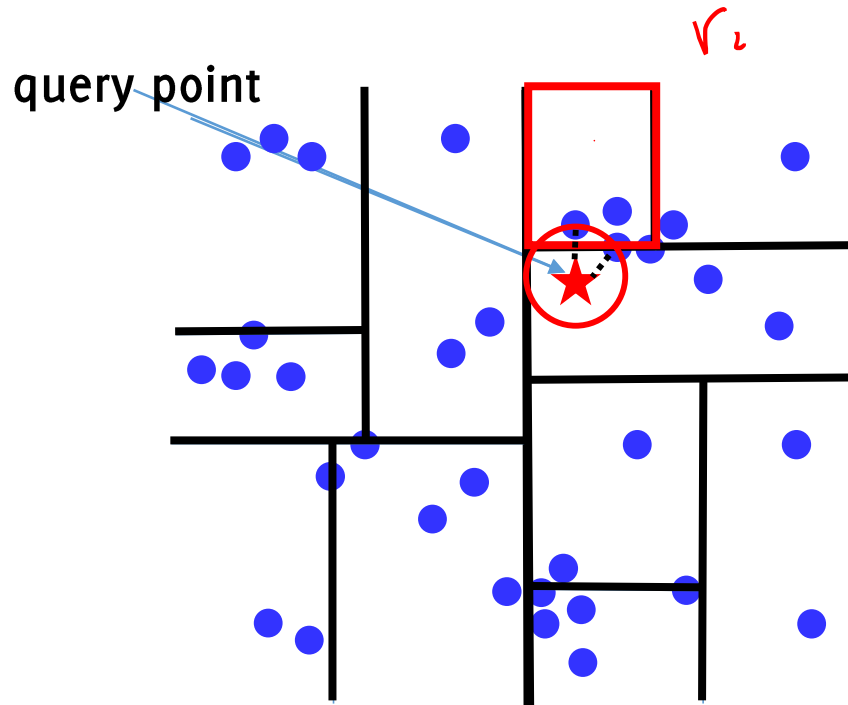
3. Backtrack the recursion on the leaves. For each leaf:
- Intersect the hypersphere with leaf hyperplanes
  - Recursively follow the tree down to nonempty intersections



# NN search over K-d trees: the rationale

3. Backtrack the recursion on the leaves. For each leaf:

- Intersect the hypersphere with leave hyperplanes
- Recursively follow the tree down
- Possibly define a new hyperplane radius (i.e. NN)

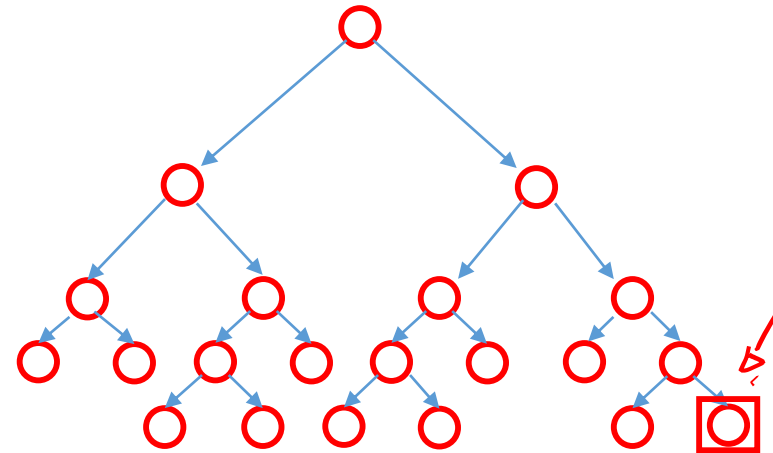
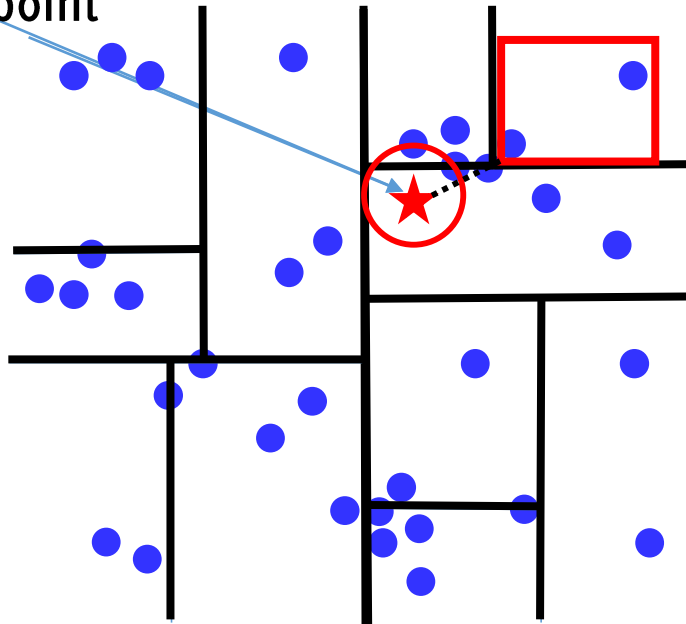


# NN search over K-d trees: the rationale

3. Backtrack the recursion on the leaves. For each leaf:

- Intersect the hypersphere with leave hyperplanes
- Recursively follow the tree down
- Possibly define a new hyperplane radius (i.e. NN)
- Empty intersection  $\rightarrow$  skip the whole branch

query point

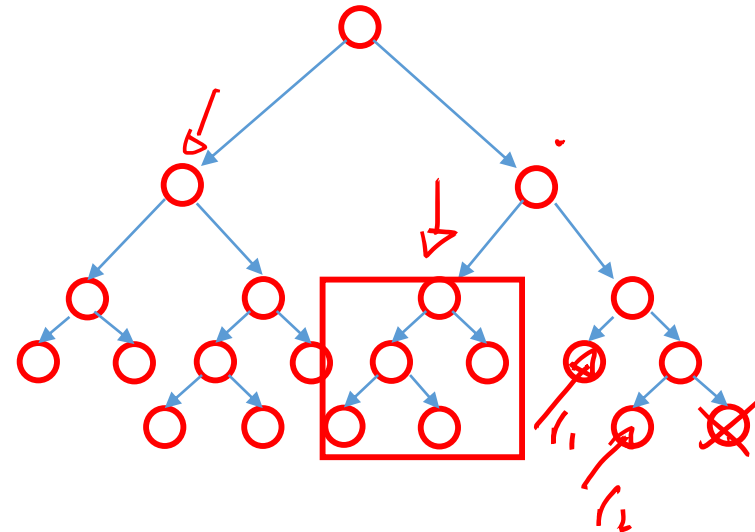
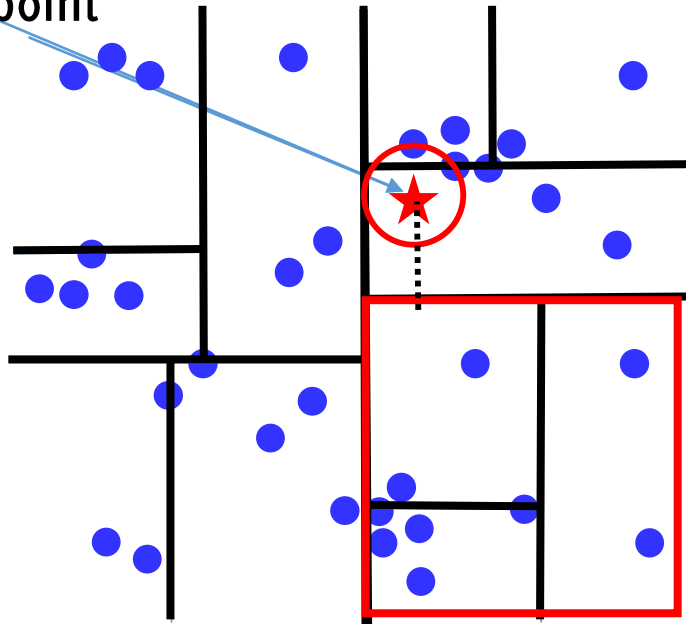


# NN search over K-d trees: the rationale

3. Backtrack the recursion on the leaves. For each leaf:

- Intersect the hypersphere with leave hyperplanes
- Recursively follow the tree down
- Possibly define a new hyperplane radius (i.e. NN)
- Empty intersection  $\rightarrow$  skip the whole branch

query point

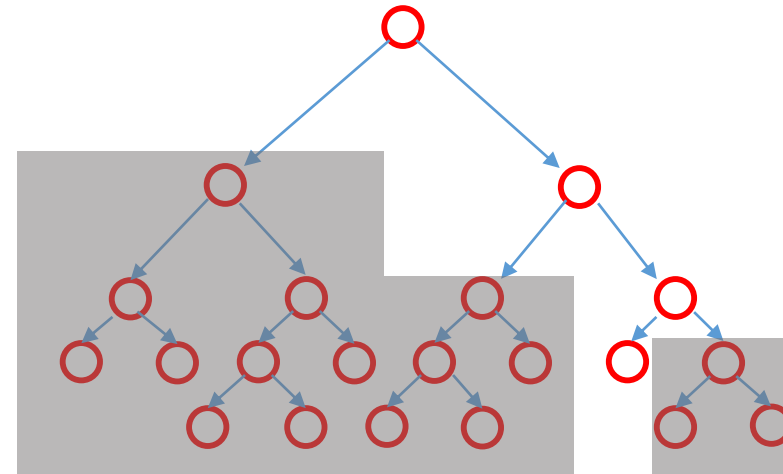
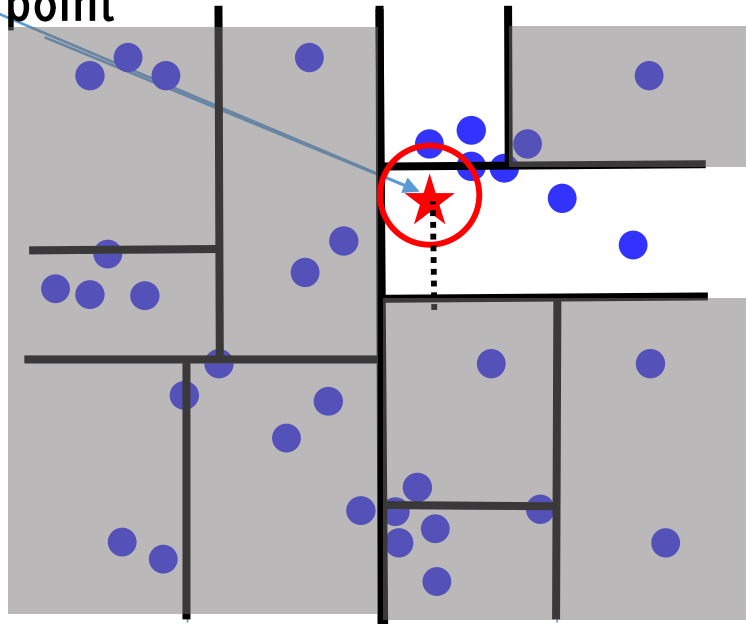


# NN search over K-d trees: the rationale

3. Backtrack the recursion on the leaves. For each leaf:

- Intersect the hypersphere with leave hyperplanes
- Recursively follow the tree down
- Possibly define a new hyperplane radius (i.e. NN)
- Empty intersection  $\rightarrow$  skip the whole branch

query point





# NN search over K-d trees

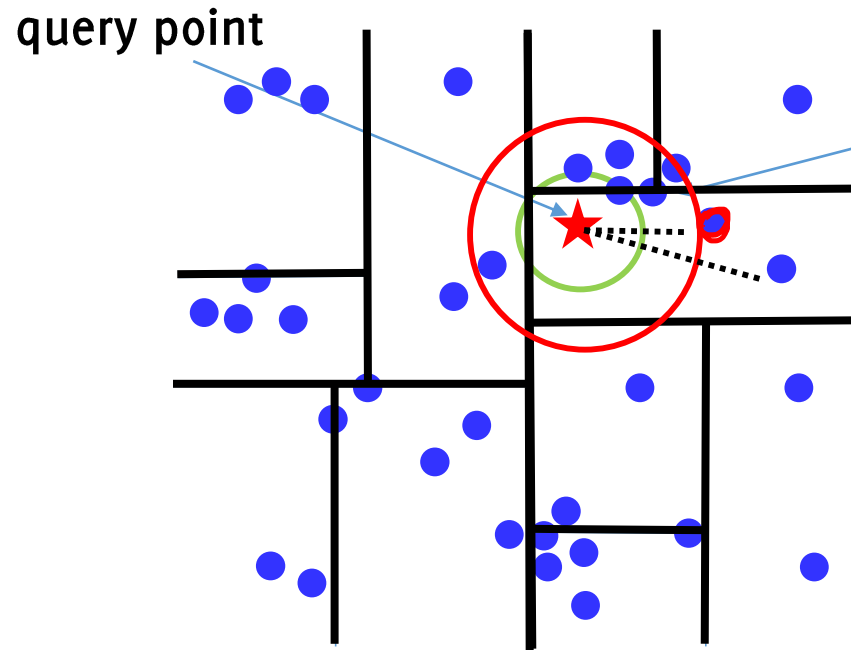
Nearest Neighbor search using k-d trees can be far more efficient than linear search, complexity becomes  $O(\log(n))$

- Efficient distance calculation since leaf splits are parallel to the axis  
→ k-d trees can be crawled by checking a single component of the vector at a time
- However, it is not very effective in high-dimensional spaces, when the number of points approaches data-dimension it becomes close to linear search.

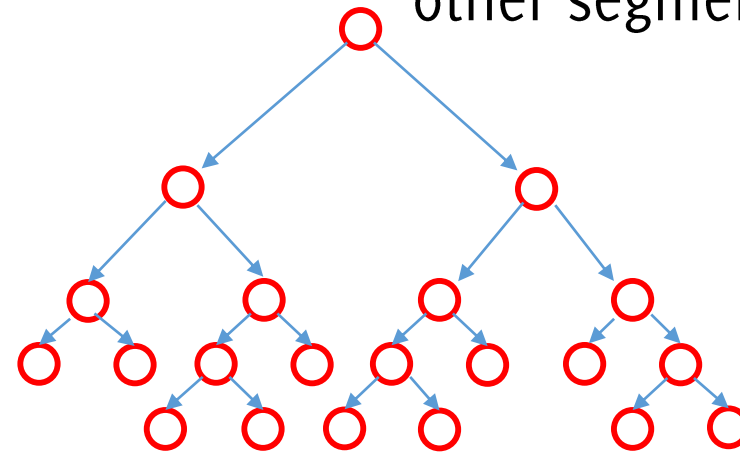
# Approximated searches over K-d trees

Approximate versions can be implemented:

- Upper-bounding the number of points to check
- Reducing the hypersphere radius by  $1/\alpha$  when controlling the intersection with other leaves

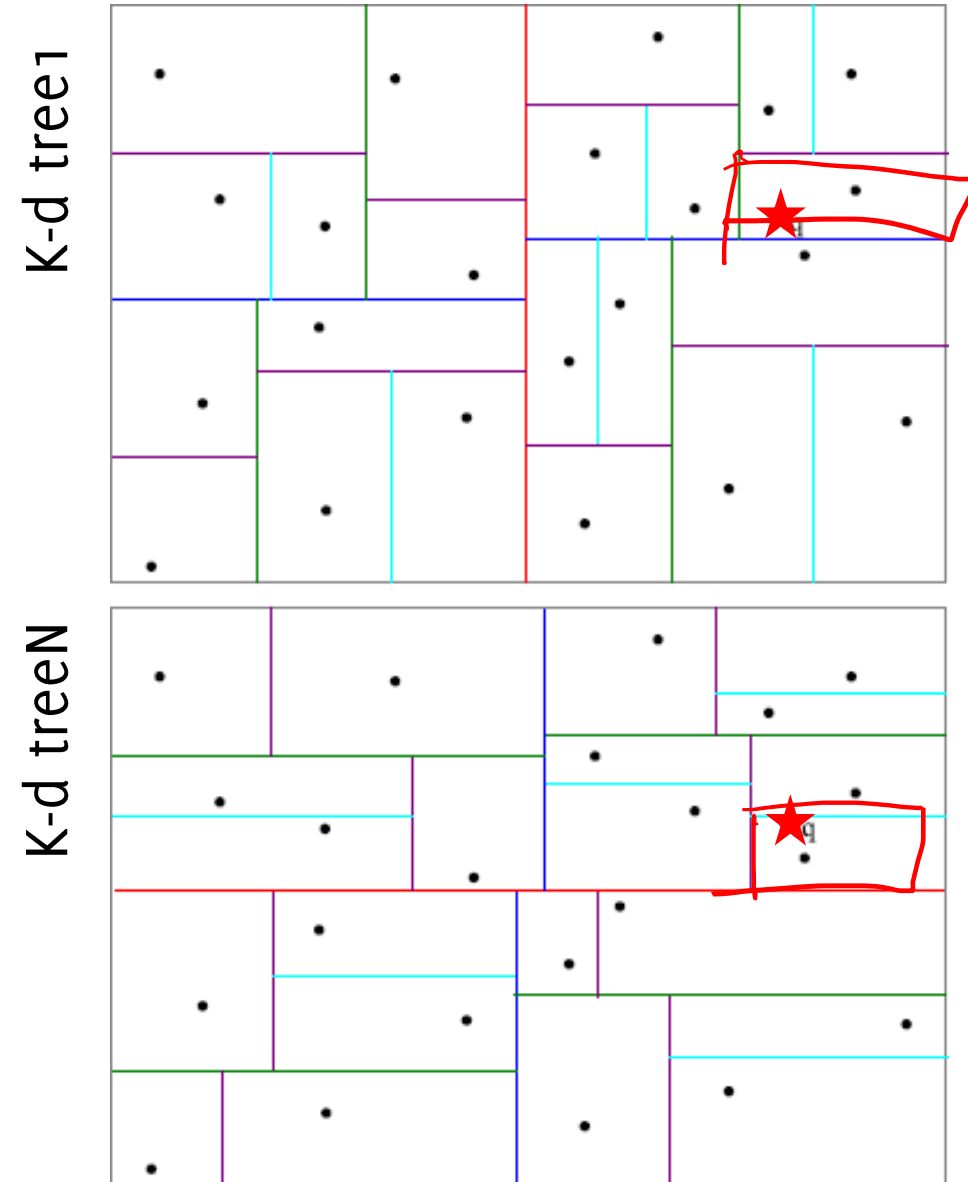


Use this radius to compare against boundaries of other segments



# Randomized Ensembles of K-d trees

Randomized ensembles of trees can speed up approximate calculations since it is more likely that the query point and the nearest neighbor fall in the same cell at least once



FLANN

# FLANN: Fast Library for Approximated Nearest Neighborhood

A library which implements advanced approximated searching methods based on trees, including new algorithms:

- Priority search k-means tree algorithms (which are not constructed as splits along the axis)
- Hierarchical Clustering Tree (meant for binary features)

# Pruning Matches

# Object Recognition by Computer Vision Features

## Estimating Image Correspondences

- Extract features from each image
  - Keypoint detection
  - Descriptor Computation
- Match features between images
- Prune matches and then perform triangulation  
detect objects / stitching ...

# Remove matches that are not good enough

The major issue:

- There does not exist a reference value for a descriptor distance for good/wrong matches
- Descriptor distance can vary a lot from scene to scene, and feature to feature

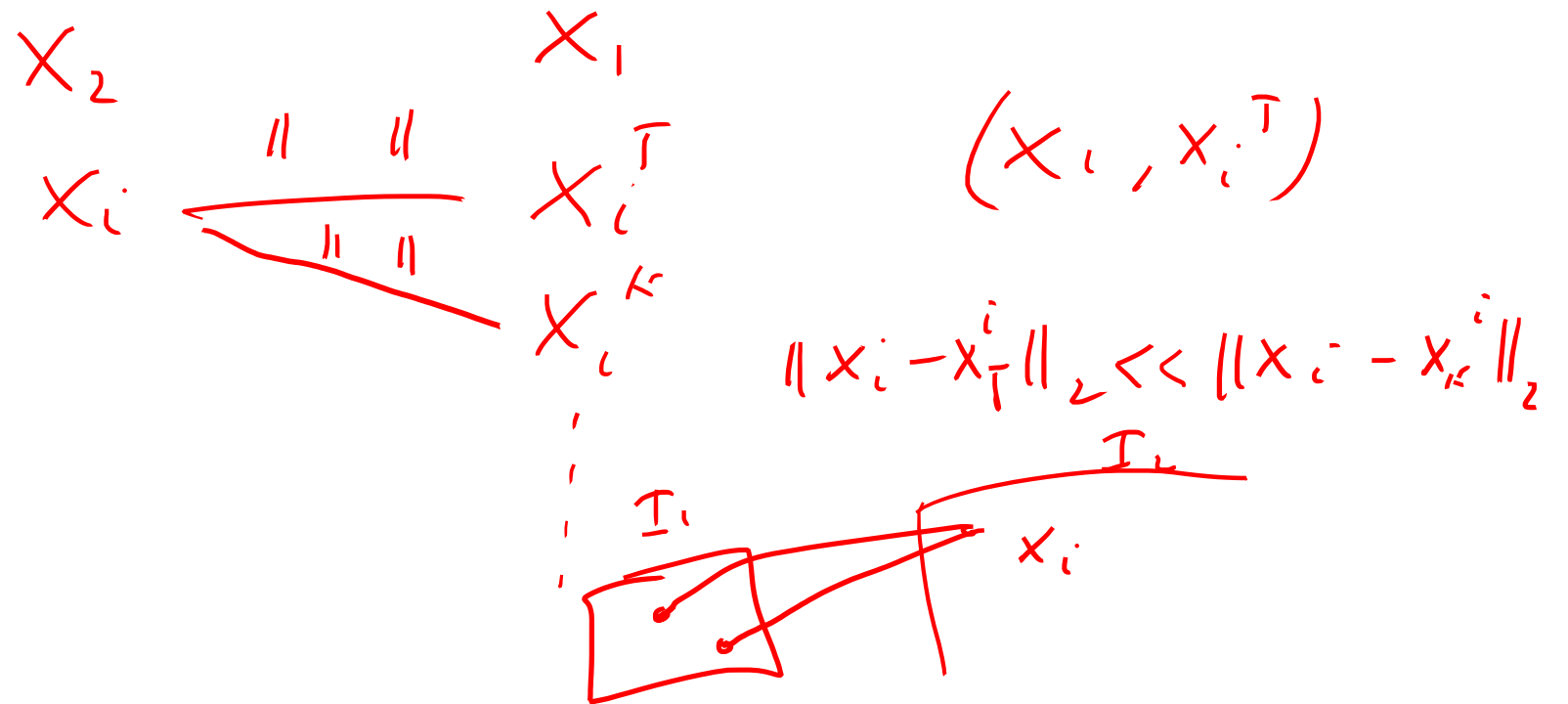
Matches have to be pruned by looking at their relative distance

$$\forall x_i \in X_2 \quad x_j^i \in X_2 \quad \text{The NEAREST.}$$
$$\|x_i - x_j^i\|_2 > \tau \quad (x_i, x_j^i) \text{ are a match}$$

# Ratio Test

By nearest neighbor search we get for each image feature  $x_i \in X_2$ , the closest template feature  $x_j^i \in X_1$

Wrong matches  $(x_i, x_j^i)$  need still to be rejected.





# Ratio Test

During matches, retrieve the 2-NN neighbor of each  $\mathbf{x}_i$ , i.e.

$$(\mathbf{x}_i, \mathbf{x}_j^i) \text{ and } (\mathbf{x}_i, \mathbf{x}_k^i)$$

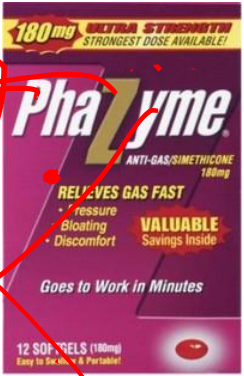
Discard keypoints where

$$\frac{\|\mathbf{x}_i - \mathbf{x}_k^i\|_2}{\|\mathbf{x}_i - \mathbf{x}_j^i\|_2} > 0.8$$

This analysis discards matches where the second nearest neighbor is very close to the first. These are:

- Ambiguous matches
- False matches arising from background clutter

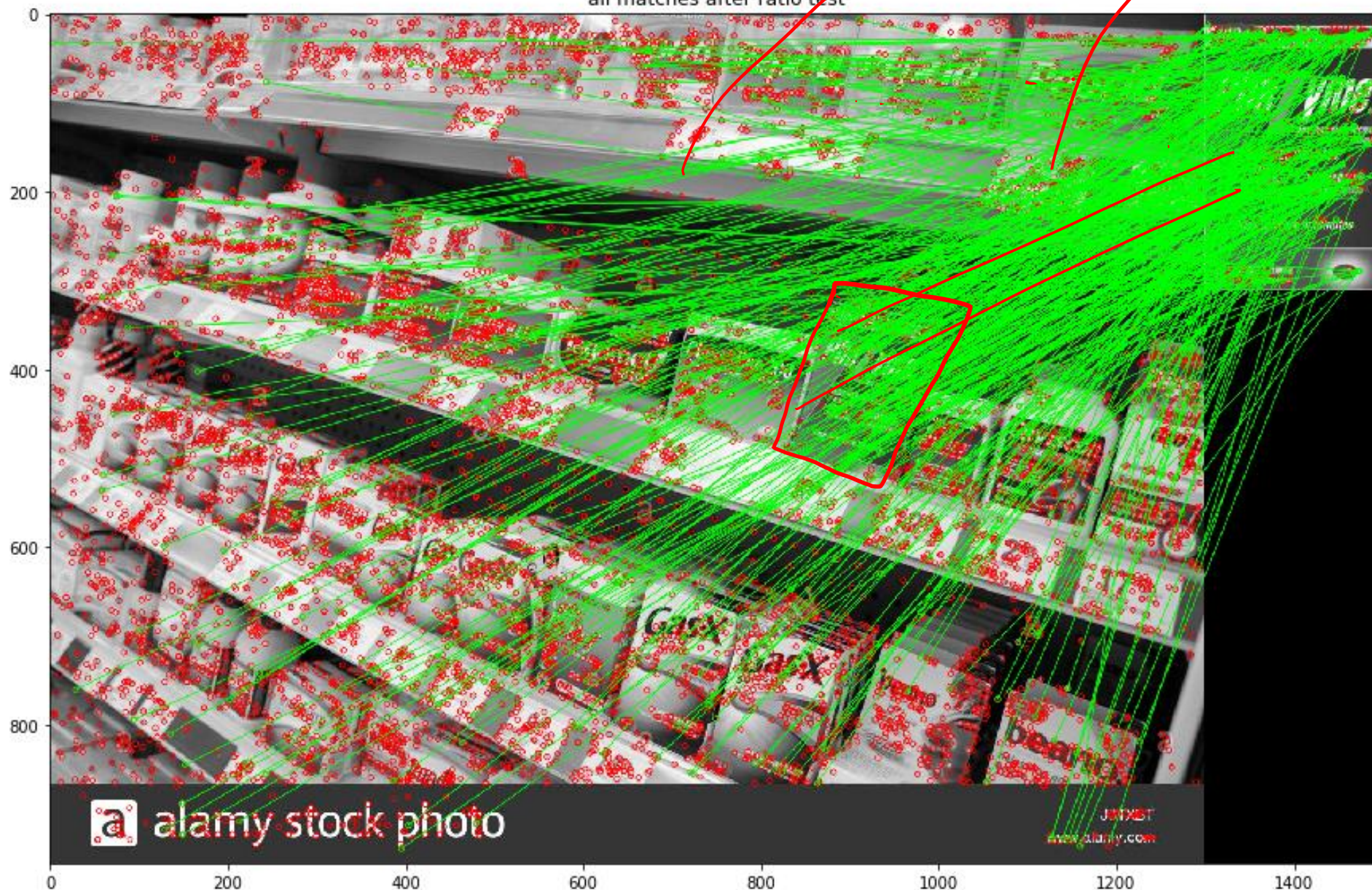
# Example of tests images



Pepcid



# Matches preserved by the ratio test



# Robust Fitting

Giacomo Boracchi

CVPR USI, April 24 2020

Credits Luca Magri, Politecnico di Milano

Let's go back to the fitting problem



# Example: Line Fitting for Vanishing Point Estimation

$X$   
 $ax + by + c = 0$   
 $\theta = (a, b, c)$





# Example: Conic Fitting





# Example: Estimating Homographic Transformations

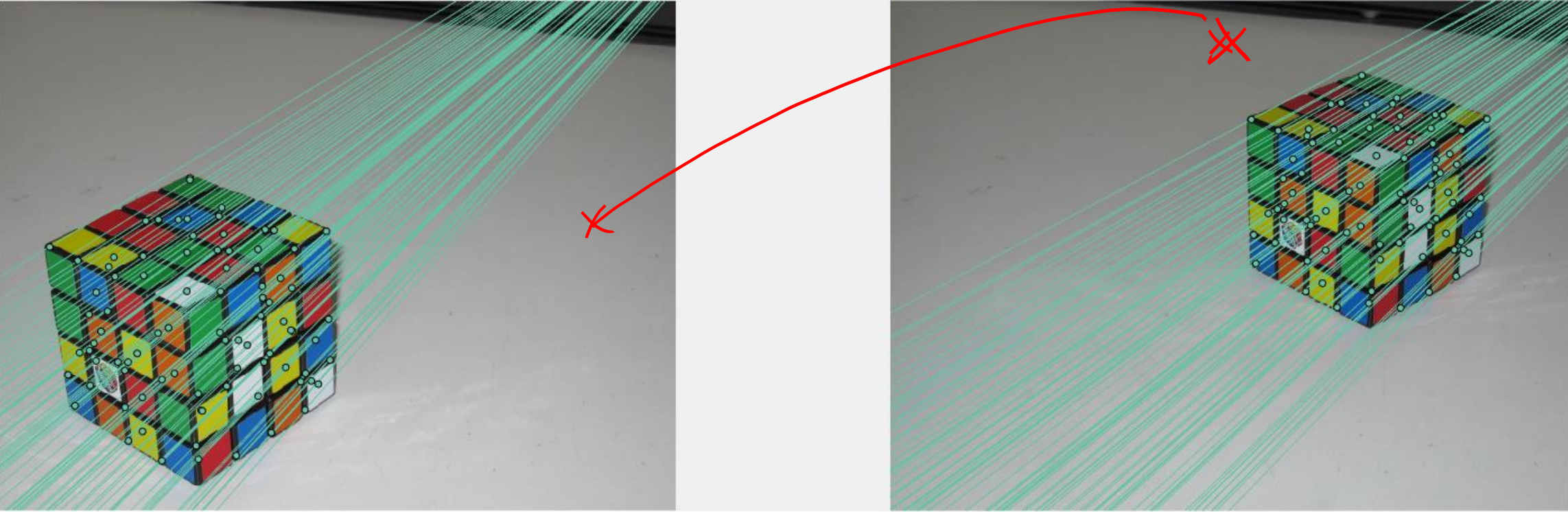
$$X = \{(x_i, y_i)\}$$

$\theta$

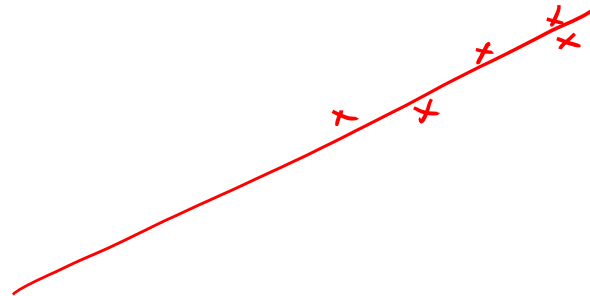




# Example: Estimating Fundamental Matrix



In all these cases the problem boils down  
✗ to fitting a parametric model to  
(presumably) noisy input data



# Ordinary Least Square

# All these problems boils down to..

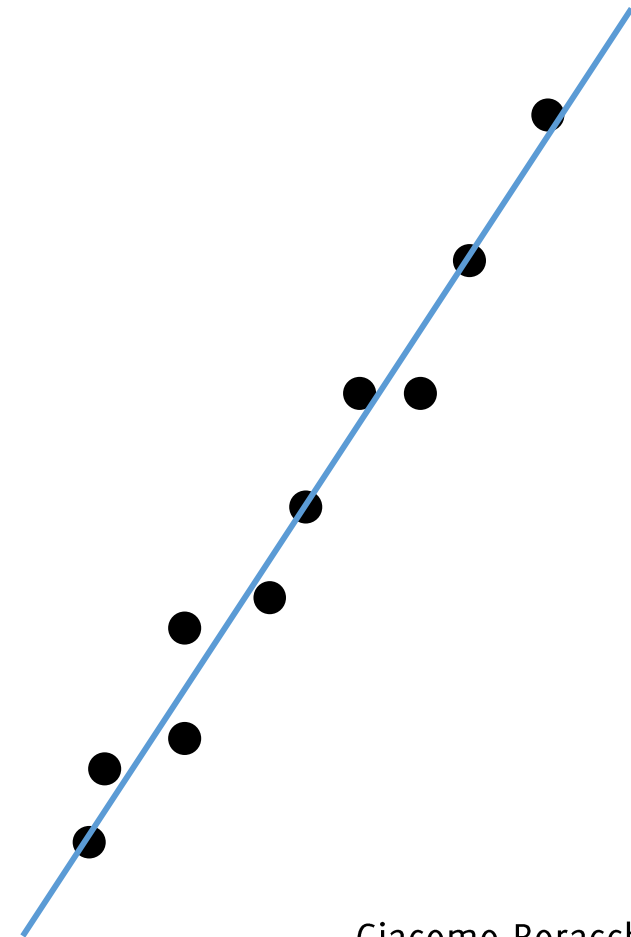
Given a set of  $N$  points (or matches..)

$$X = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

Given a parametric model

$$y = mx + q$$

Estimate the parameters  $m, q$  yielding the **best fit**



# Least Square Regression

Given a set of  $N$  points (or matches..)

$$X = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

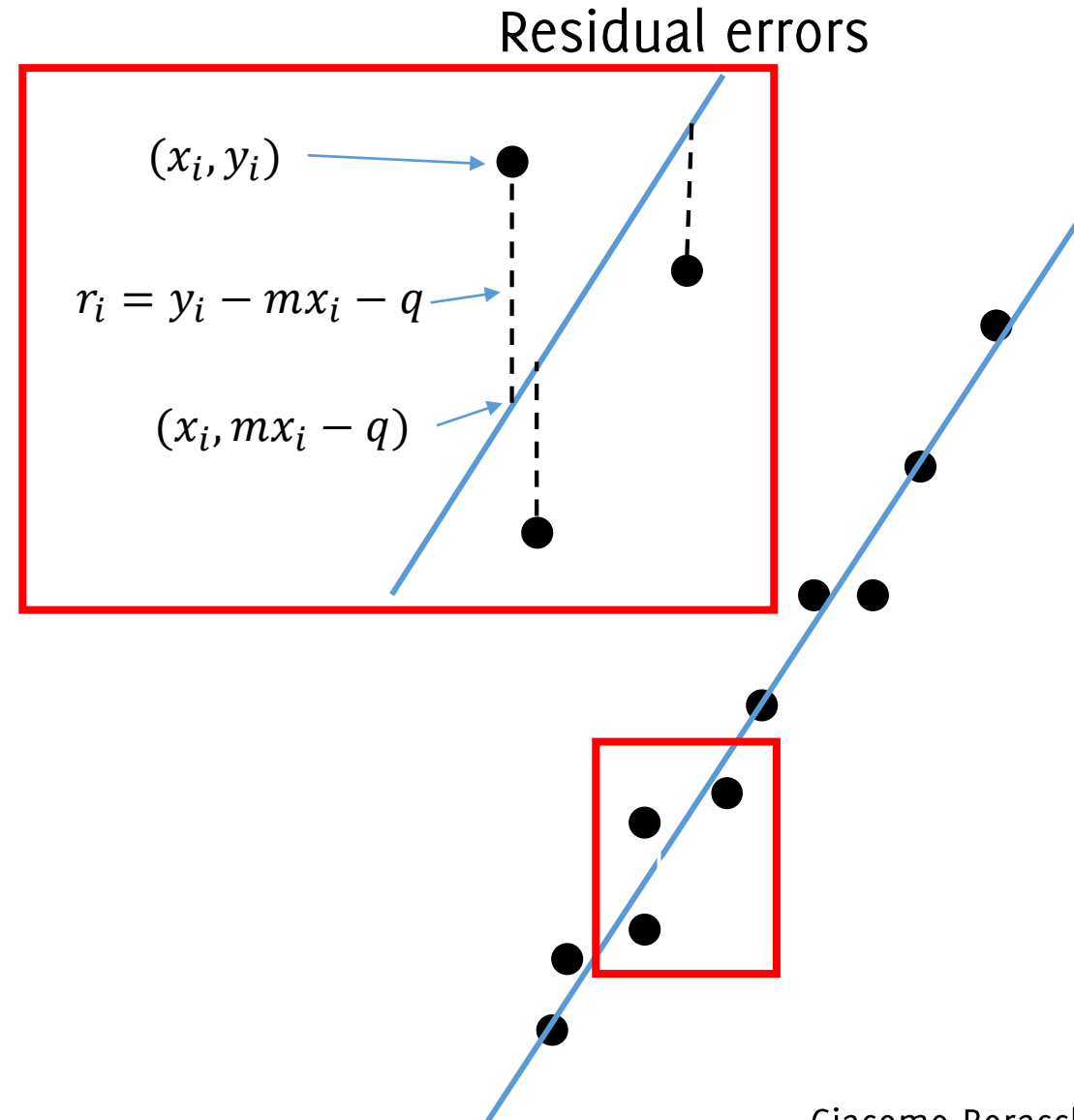
Given a parametric model

$$y = mx + q$$

Estimate the parameters  $m, q$  yielding the **best fit**

The best fit is the one **minimizing some residual error over the whole data**

$$r_i = y_i - mx_i - q$$



# Ordinary Least Square (OLS) Regression

The loss function is the sum of squared residual errors

$$E = \sum_{i=1}^N (r_i)^2 = \sum_{i=1}^N (y_i - mx_i - q)^2 =$$

Which in matrix form becomes

$$r_i = y_i - [x_i \ 1] \begin{bmatrix} m \\ q \end{bmatrix}$$

$$E = \left\| \begin{bmatrix} y_1 \\ \dots \\ y_N \end{bmatrix} - \begin{bmatrix} x_1 & 1 \\ \dots & \dots \\ x_N & 1 \end{bmatrix} \begin{bmatrix} m \\ q \end{bmatrix} \right\|_2^2$$

Then, the solution can be computed via least square regression

$$[\hat{m}, \hat{q}] = \underset{\underline{m, q}}{\operatorname{argmin}} \left\| Y - X \begin{bmatrix} m \\ q \end{bmatrix} \right\|_2^2$$

# Ordinary Least Square (OLS) Regression

OLS consists in solving the following problem

$$[\hat{m}, \hat{q}] = \underset{m, q}{\operatorname{argmin}} \left\| Y - X \begin{bmatrix} m \\ q \end{bmatrix} \right\|_2^2$$

by zeroing the derivative of the loss function

$$\frac{\partial}{\partial \theta} \|Y - X\theta\|_2^2 = 0, \quad \theta = \begin{bmatrix} m \\ q \end{bmatrix}$$

This follows from matrix calculus

$$\frac{\partial}{\partial \theta} \|Y - X\theta\|_2^2 = \underline{2X^T(X\theta - Y)}$$

Thus the solution becomes

$$* \quad 2X^T(X\hat{\theta} - Y) = 0 \rightarrow \hat{\theta} = (X^T X)^{-1} X^T Y$$

# Ordinary Least Square (OLS) Regression

In case the residuals have different variance, one typically wants to minimize the following loss

$$E = \sum_{i=1}^N \left( \frac{r_i}{\sigma_i} \right)^2$$

Being  $\sigma_i$  the standard deviation of the residual  $r_i$

This leads to weighted least squares:

$$2X^T W X \hat{\theta} = -X^T W Y \quad *$$

Being  $W = \text{diag} \left( \left[ \frac{1}{\sigma_1^2}, \dots, \frac{1}{\sigma_N^2} \right] \right)$



This error does not make sense  
when the line is vertical

# What about minimizing point-line distance?

Given a set of  $N$  points (or matches..)

$$X = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

Given a parametric model

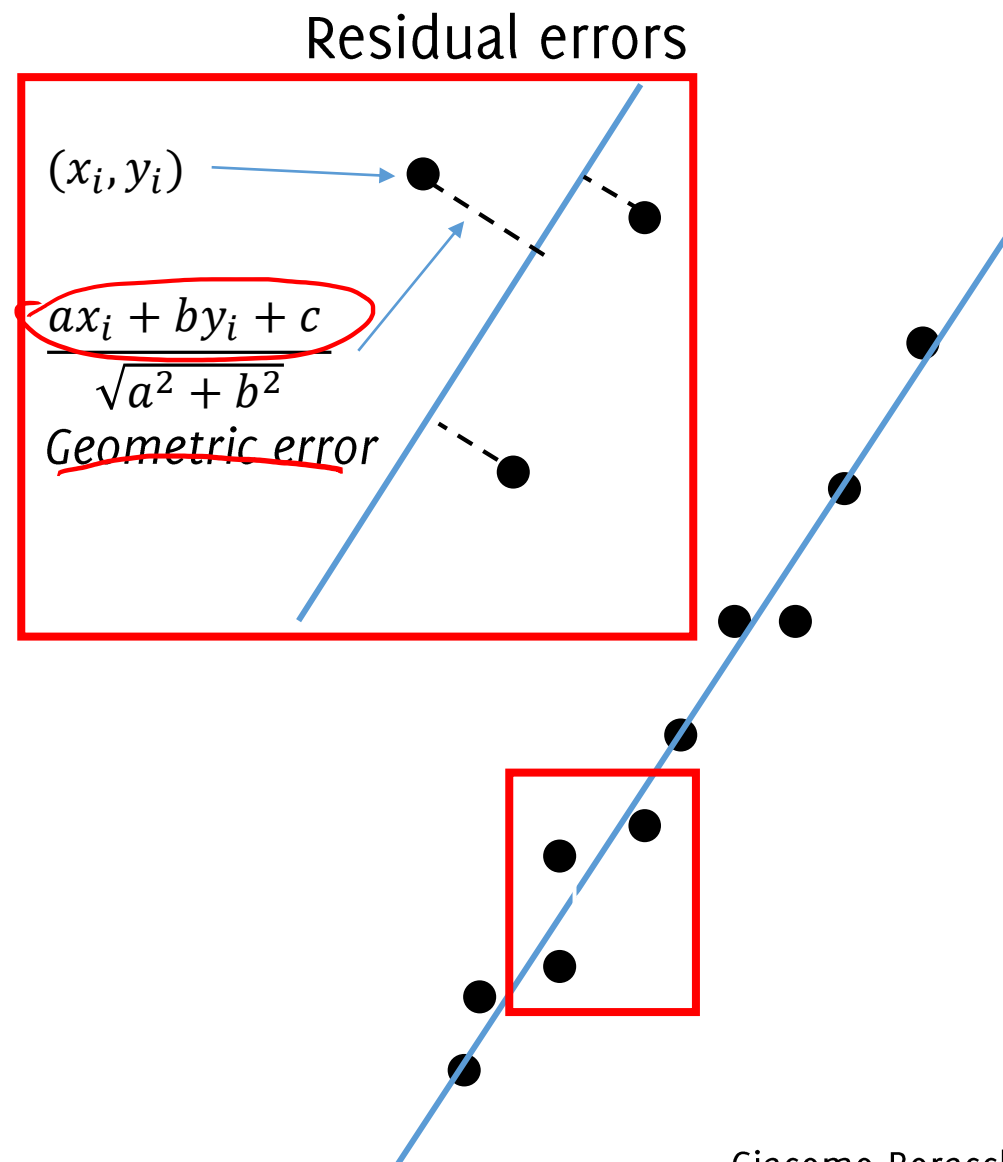
$$ax + by + c = 0$$

Estimate the line parameters  $a, b, c$  yielding the **best fit minimizing** the residual error

$$E = \sum_{i=1}^N (ax_i + by_i + c)^2$$

If we take

$$r_i = ax_i + by_i + c$$



# What about minimizing point-line distance?

Given a set of  $N$  points (or matches..)

$$X = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

Given a parametric model

$$ax + by + c = 0$$

Estimate the line parameters  $a, b, c$  yielding the best fit minimizing the residual error

$$E = \sum_{i=1}^N (ax_i + by_i + c)^2$$

# What about minimizing point-line distance?

$$E = \left\| \begin{bmatrix} x_1 & y_1 & 1 \\ \dots & \dots & \dots \\ x_N & y_N & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \right\|_2^2$$
$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \|A\theta\|_2^2,$$

Being the parameter vector  $\theta = [a; b; c]$  and constraining  $\|\theta\|_2 = 1$  due to the equivalence relation between  $\theta$  and lines

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \|A\theta\|_2^2, \text{ subject to } \|\theta\|_2 = 1$$

**The DLT solves this problem by minimizing the algebraic error!**

$$\theta = V(:, \text{end}), \text{ being } A = UDV^\top$$

# Robust Fit

Giacomo Boracchi

CVPR USI, April 28 2020

# Next Homework

A 20 points homework assignments

- 10 points on template detection (Sequential RanSaC)
- 10 points on image restoration (most probably denoising)
  - 3 weeks for solving each

A 30 points project

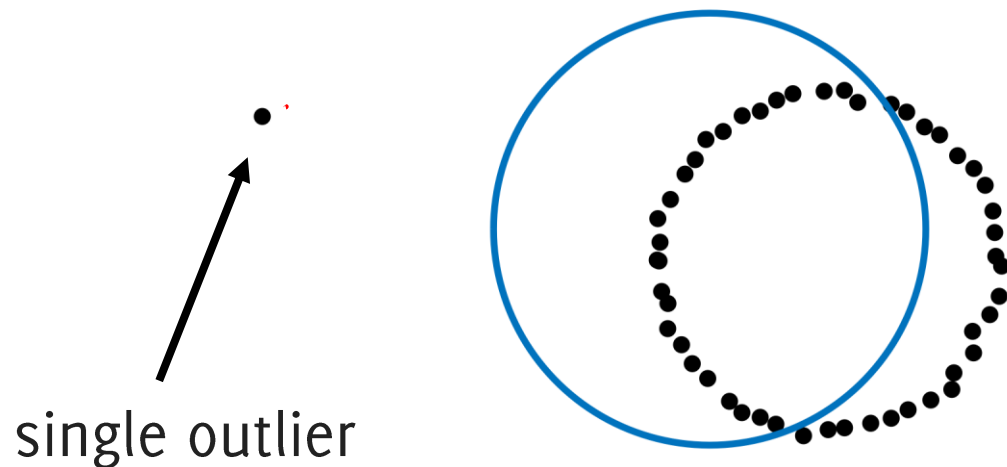
- Related to multiple template detection (multiple instances, multiple templates)
- Scenario and “research direction” up to you
- Outline to be discussed in a class-presentation on May 26° (possible to ask for feedback before)
- Due date on the exam day
- Oral exam [0-25] includes presentation of final results of the project

# Robustness to Outliers

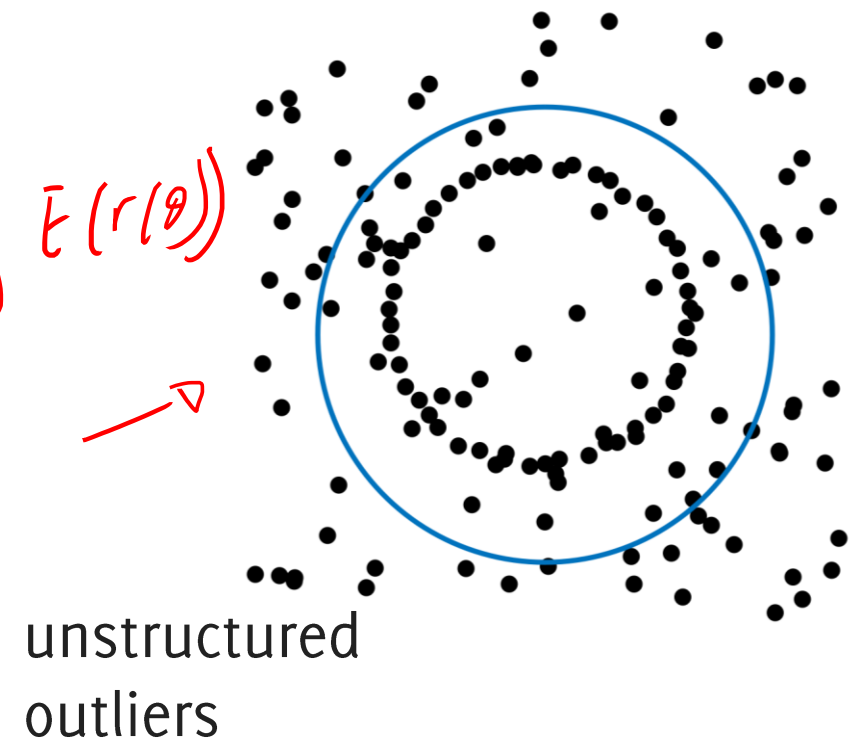
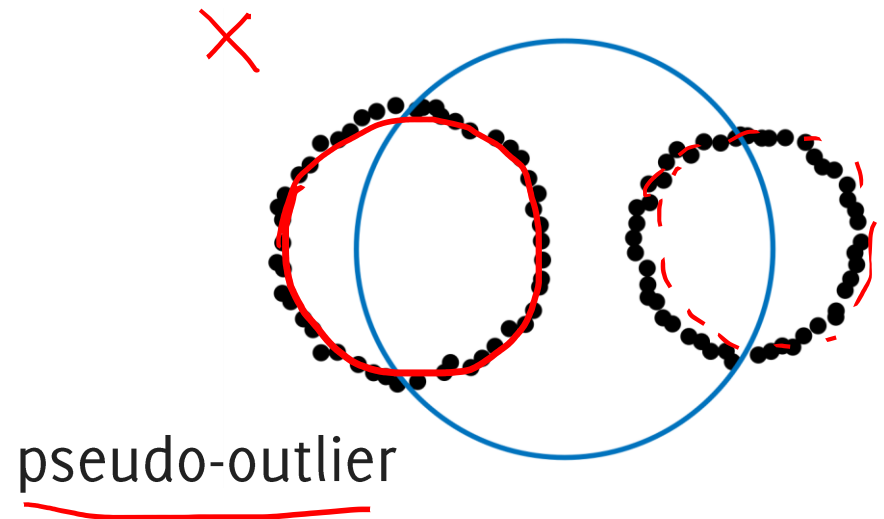
# Least squares breaks down

**Break down point:** the proportion of incorrect observations that can be handled before giving an arbitrarily large incorrect result.

Least squares has 0% breakdown point (the outlier might be arbitrarily large, i.e.,  $\rightarrow \infty$ )



$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} E(r(\theta))$



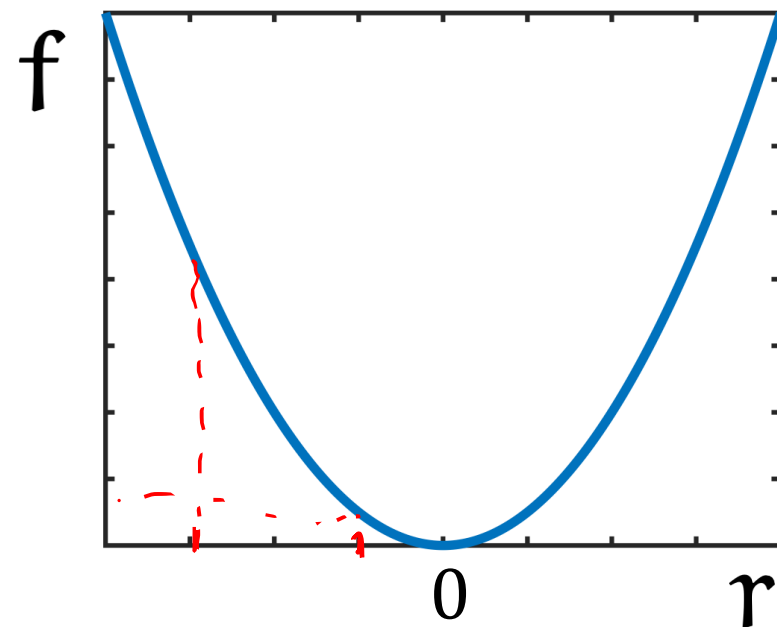


# The loss function in OLS

The loss so far is the sum of a function of all the residuals

$$E = \sum_{i=1}^N f(r_i), \text{ where } f(r_i) = r_i^2$$

However, other options for  $f$  are viable, giving rise to different loss functions, and different results

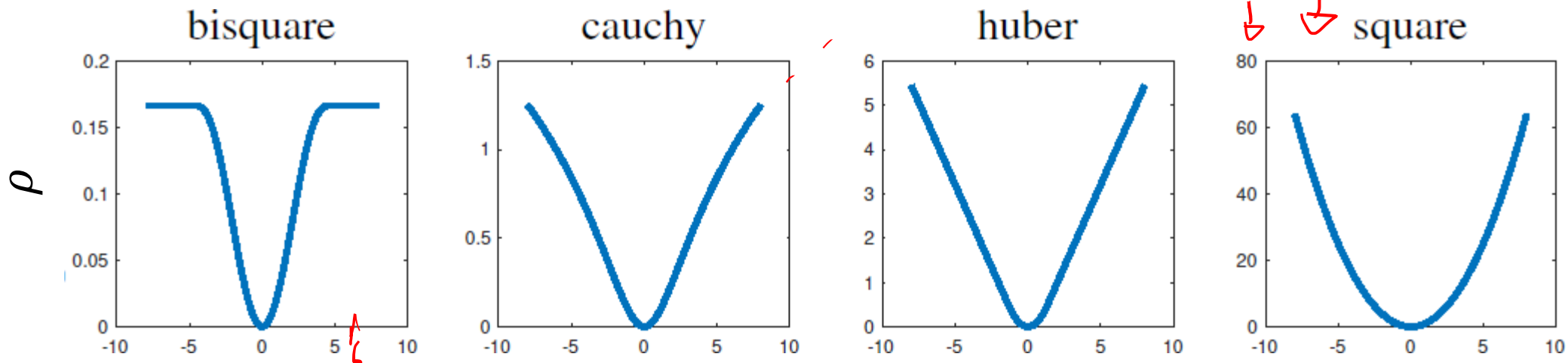


# M-Estimator

Replaces the squared loss in the OLS with a different loss function which penalizes less large residual values (deemed to correspond to outliers)

$$\hat{\theta} = \operatorname{argmin}_{\theta} \sum_{i=1}^N \rho(r_i(\theta)) \quad *$$

Where  $\rho$  a symmetric, positive-definite function having a unique minimum in zero



# M-Estimator

To solve this minimization problem we need to zero the derivative for each dimension of  $\theta$

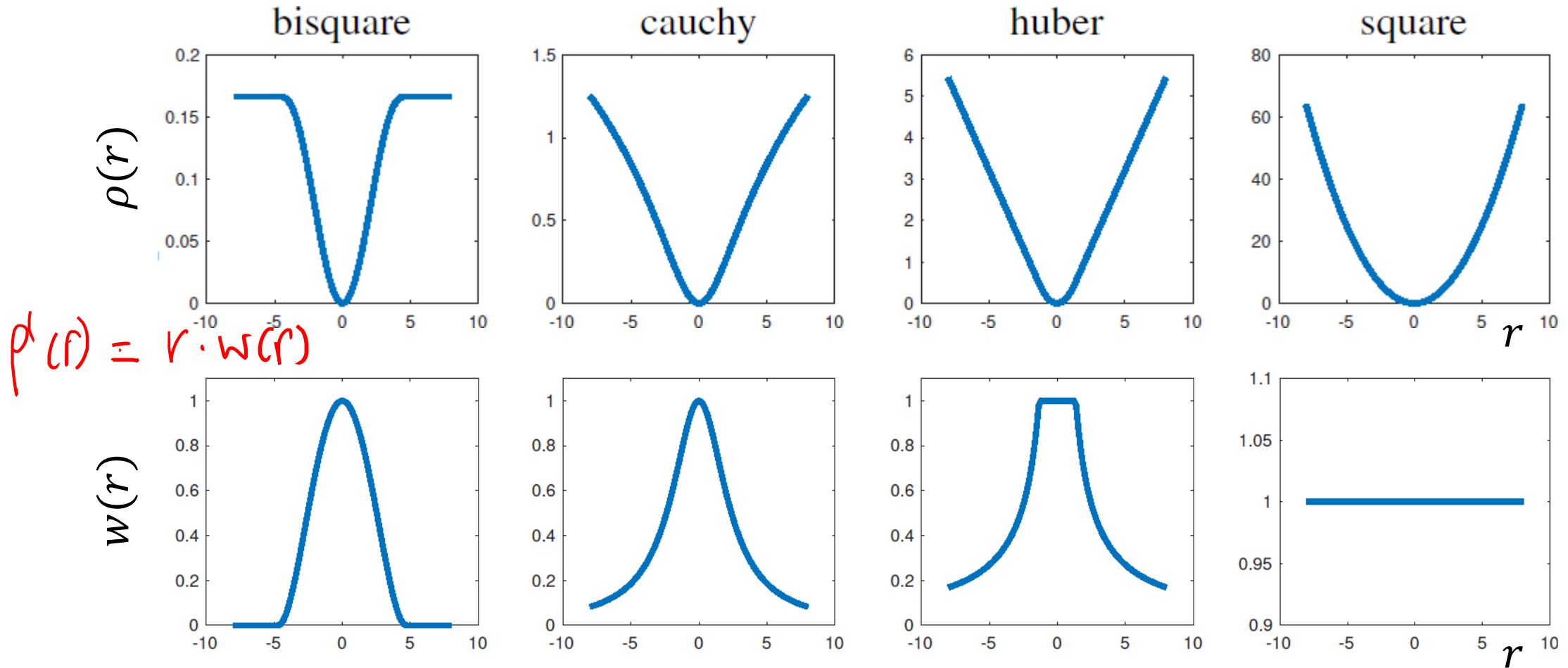
$$\sum_{i=1}^N \rho(r_i(\theta)) \quad \sum_{i=1}^N \rho'(r_i) \frac{\partial r_i}{\partial \theta_j} = 0, \quad j = 1, \dots, M$$

The trick is to associate to  $\rho$  a weight function  $w$  such that

$$\rho'(x) = x * w(x)$$

# M-Estimator

Weights associated to the previous losses are



# M-Estimator

Then the zeroing the derivative corresponds to solving the following

$$\sum_{i=1}^N r_i * \overbrace{w(r_i)}^{\substack{\text{fixed} \\ \uparrow}} \frac{\partial r_i}{\partial \theta_j} = 0 \quad j = 1, \dots, m \quad (1)$$

$w(r^{(k-1)}) \rightarrow \theta \quad r^{(k)}$

If we consider an iterative scheme where at iteration  $k$ , we alternate weight definition and minimization for  $r_i$ . Then, during the  $k$ -th iteration we treat weights as fixed and defined by  $r_i^{(k-1)}$ , such that we solve (1) just w.r.t. to  $r_i$ . In this case,  $\hat{\theta}$  is the same solution of

$$\operatorname{argmin}_{\theta} \sum_{i=1}^N \underbrace{w(r_i^{(k-1)})}_{\substack{\text{weighted} \\ \text{least square}}} r_i^2$$

Which is a weighted least square problem!

# M-Estimator

$$\sum_i p(r_i)$$

Minimizing this

$$\hat{\theta}^{(k)} = \operatorname{argmin}_{\theta} \sum_i w(r_i^{(k-1)}) r_i(\theta)^2$$

Can be done by weighted least square, but weights  $w$  depend on  $r_i(\theta)$

Iterative Reweighted Least Squares (IRLS) alternates

- Define weights  $w(r_i^{(k-1)})$  from residual  $r_i(\hat{\theta}^{(k-1)})$  at the previous iteration. This is done by simply sampling the weight curves.
- Define the model  $\hat{\theta}^{(k)}$  by solving the weighted least square keeping  $w(r_i^{(k-1)})$  as fixed

# The Weights and $\rho$

What actually determines the M-estimator is the influence function  $\rho'$ .

- $\rho'$  in robust estimator should be **bounded**. For example  $r^2$  has not a bounded derivative. The others shown before have a bounded derivative since  $\rho'(x) = w(x) * x$  decay faster than linearly.
- The function  $\rho$  should be strictly convex, to yield a unique minimum

Obviously, the weights should decrease as  $r$  increases, to bound residual contribution.

# RanSaC



# Robust single model fitting: consensus maximisation

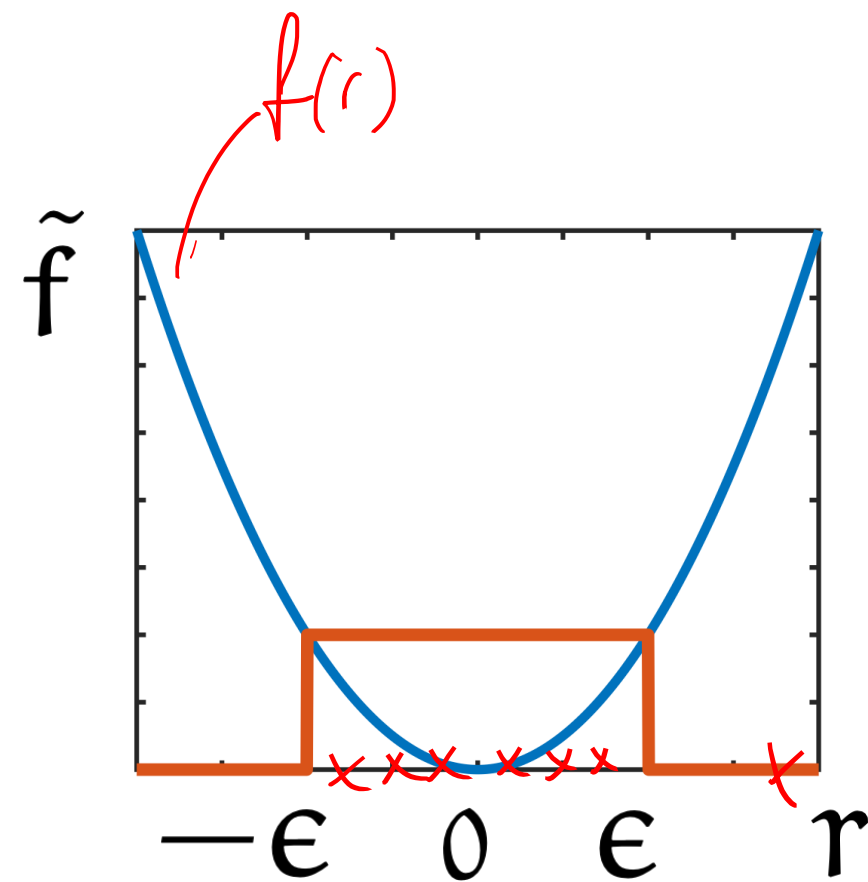
Instead of minimizing the residuals, **maximize the consensus**. Define:

- an inlier threshold  $\epsilon > 0$
- a consensus function  $\tilde{f}$  which is

$$\tilde{f}(r_i) = \begin{cases} 1, & |r_i| \leq \epsilon \\ 0, & |r_i| > \epsilon \end{cases}$$

Identify  $\hat{\theta}$  that maximizes the consensus

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^N \tilde{f}(r_i(\theta))$$



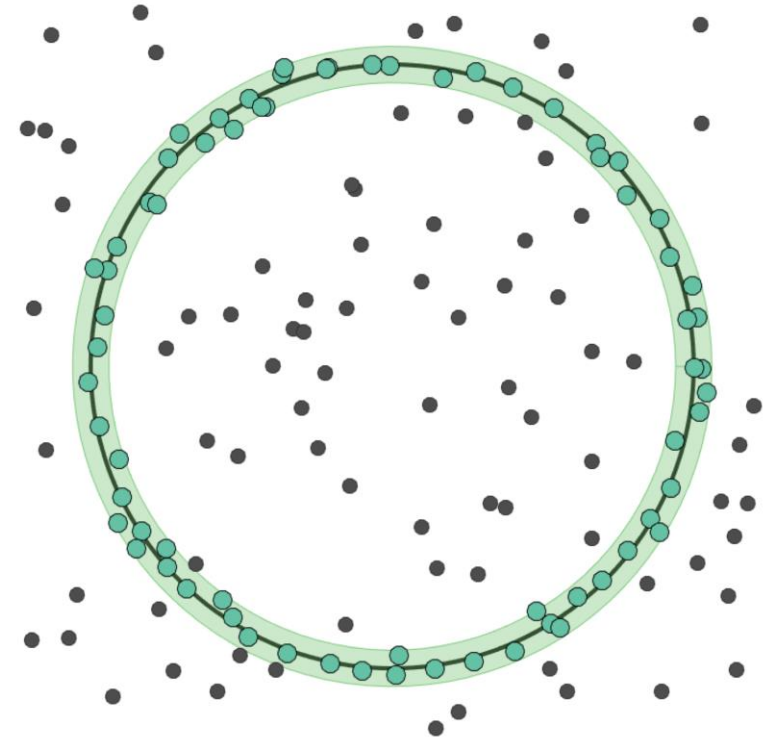
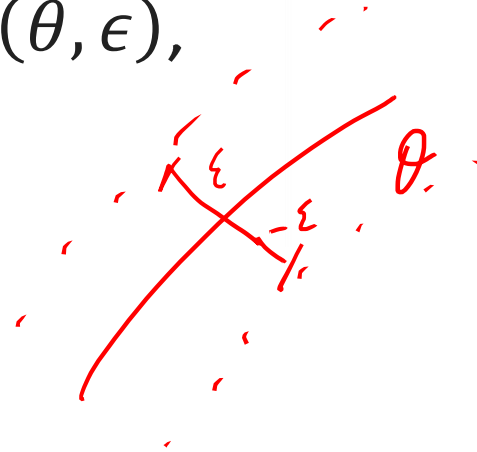
# The Consensus Set

Consensus set

$$CS(\theta, \epsilon) = \{x_i \mid |r_i| \leq \epsilon\}$$

Being  $r_i = r(x_i, \theta)$ , the residual of the model  $\theta$  at a point  $x_i$

The larger the consensus set  $CS(\theta, \epsilon)$ , the better the model  $\theta$



# The Consensus Set

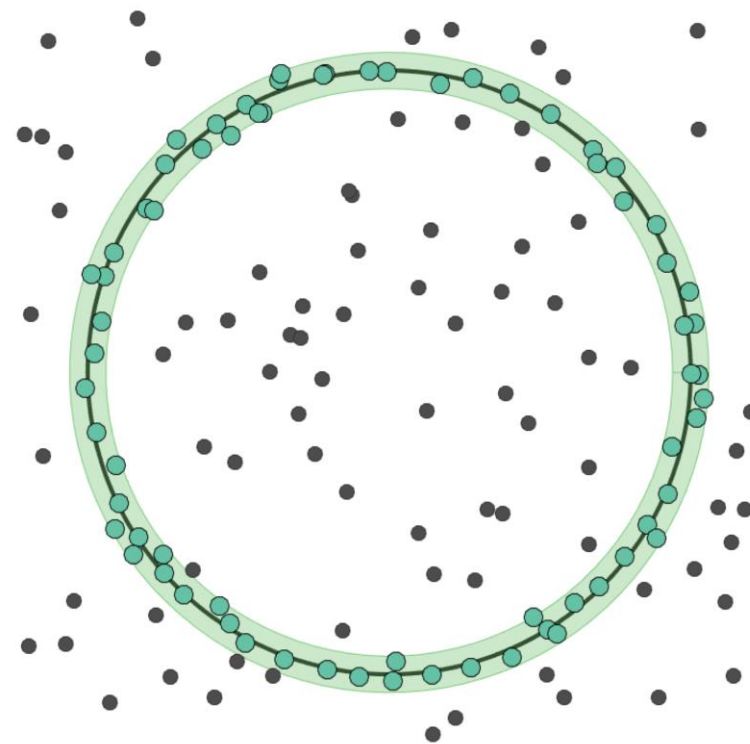
Consensus set

$$CS(\theta, \epsilon) = \{x_i \mid r_i \leq \epsilon\}$$

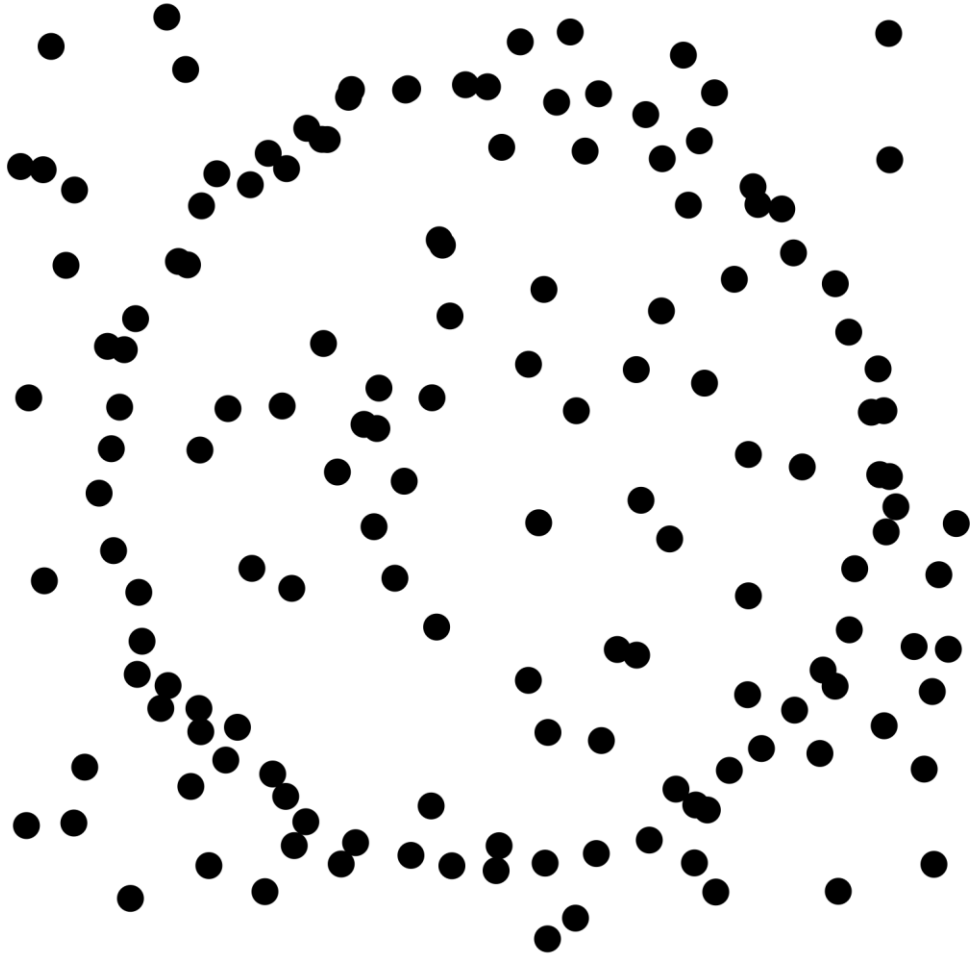
Being  $r_i = r(x_i, \theta)$ , the residual of the model  $\theta$  at a point  $x_i$

The larger the consensus set  $CS(\theta, \epsilon)$ , the better the model  $\theta$

We have been fitting lines so far, but **everything** holds for any parametric model (e.g. circle, conics, homographies, fundamental matrices)



# Randomized Sample Consensus [Fischler and Bolles 1981]



**Input:**  $X$  data,  $\epsilon$  inlier threshold,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = -\infty, k = 0;$

**repeat**

    Select randomly a minimal sample set  $S \subset X$ ;

    Estimate parameters  $\theta$  on  $S$ ;

    Evaluate  $J(\theta) = \sum_{x \in X} \hat{f}_\epsilon(r(x, \theta));$

**if**  $J(\theta) > J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

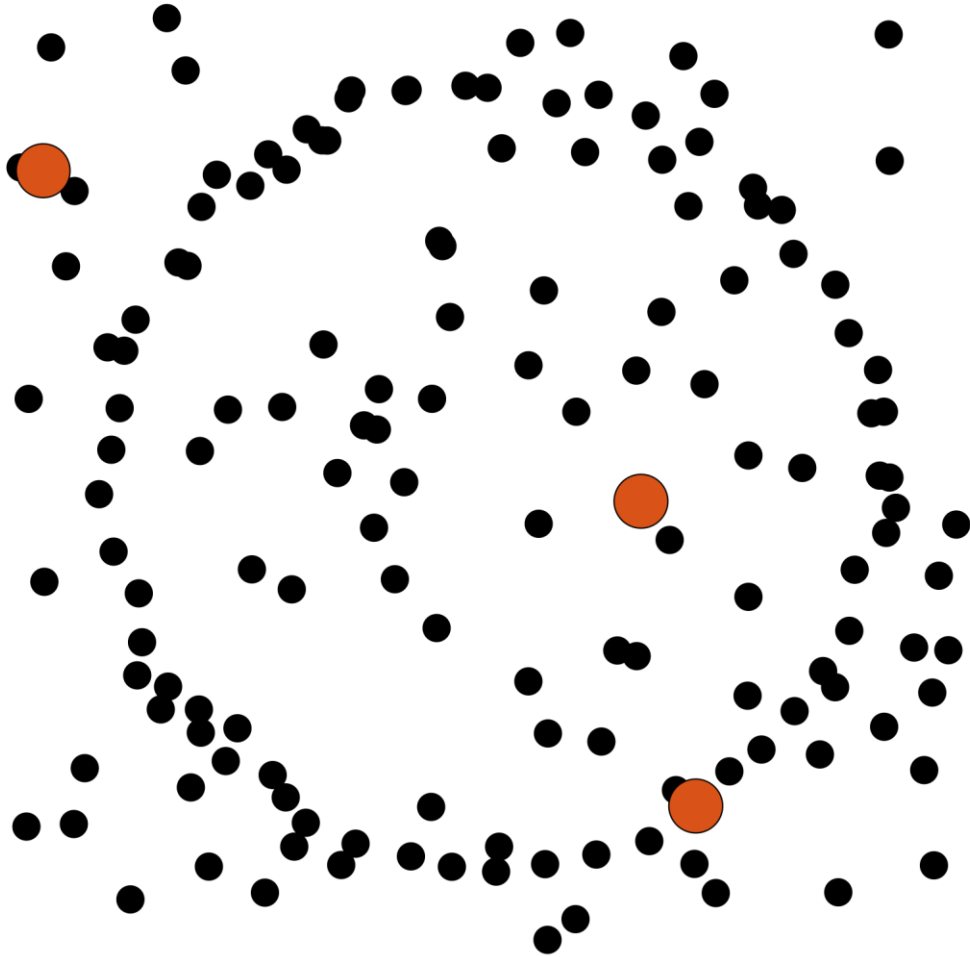
**end**

$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

# Randomized Sample Consensus [Fischler and Bolles 1981]



**Input:**  $X$  data,  $\epsilon$  inlier threshold,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = -\infty, k = 0;$

**repeat**

Select randomly a minimal sample set  $S \subset X;$

Estimate parameters  $\theta$  on  $S;$

Evaluate  $J(\theta) = \sum_{x \in X} \hat{f}_\epsilon(r(x, \theta));$

**if**  $J(\theta) > J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

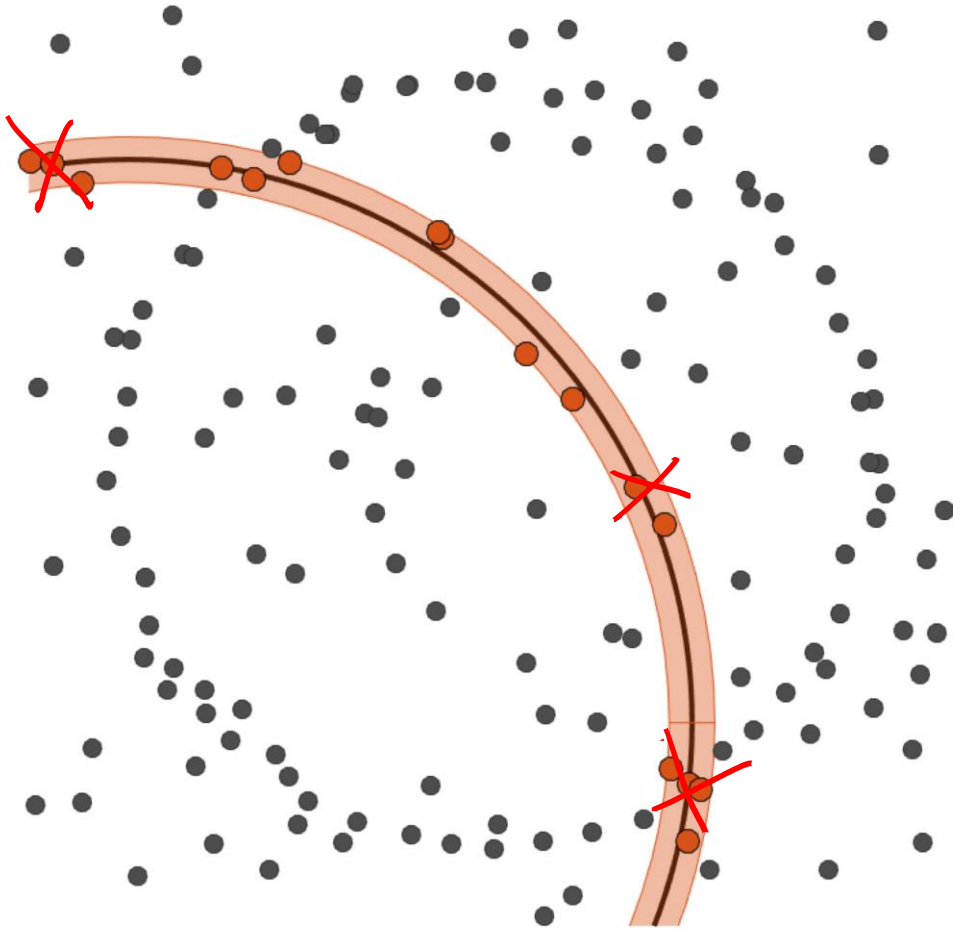
**end**

$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

# Randomized Sample Consensus [Fischler and Bolles 1981]



**Input:**  $X$  data,  $\epsilon$  inlier threshold,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = -\infty, k = 0;$

**repeat**

Select randomly a minimal sample set  $S \subset X;$

Estimate parameters  $\theta$  on  $S;$

Evaluate  $J(\theta) = \sum_{x \in X} \hat{f}_\epsilon(r(x, \theta));$

**if**  $J(\theta) > J^*$  **then**

$\theta^* = \theta;$

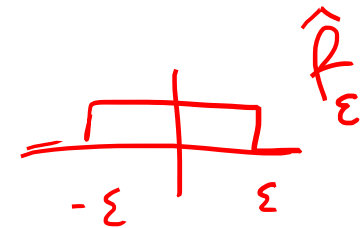
$J^* = J(\theta);$

**end**

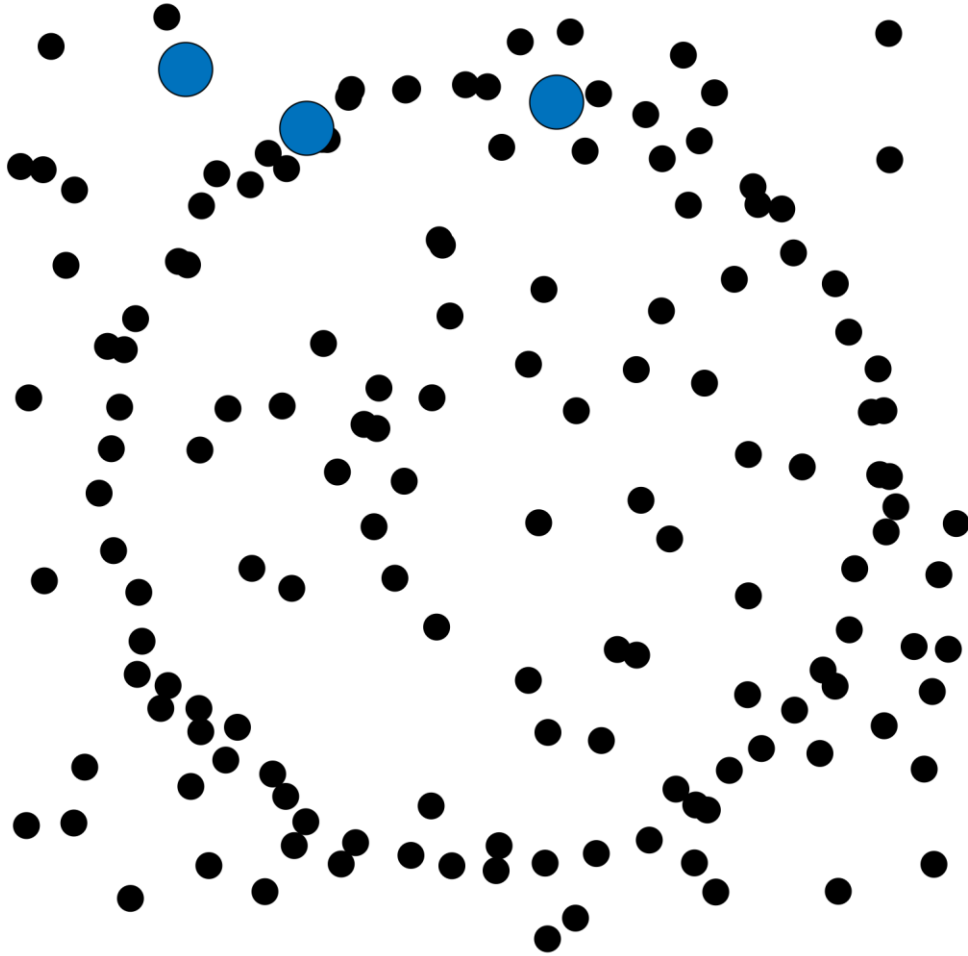
$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.



# Randomized Sample Consensus [Fischler and Bolles 1981]



**Input:**  $X$  data,  $\epsilon$  inlier threshold,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = -\infty, k = 0;$

**repeat**

Select randomly a minimal sample set  $S \subset X;$

Estimate parameters  $\theta$  on  $S;$

Evaluate  $J(\theta) = \sum_{x \in X} \hat{f}_\epsilon(r(x, \theta));$

**if**  $J(\theta) > J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

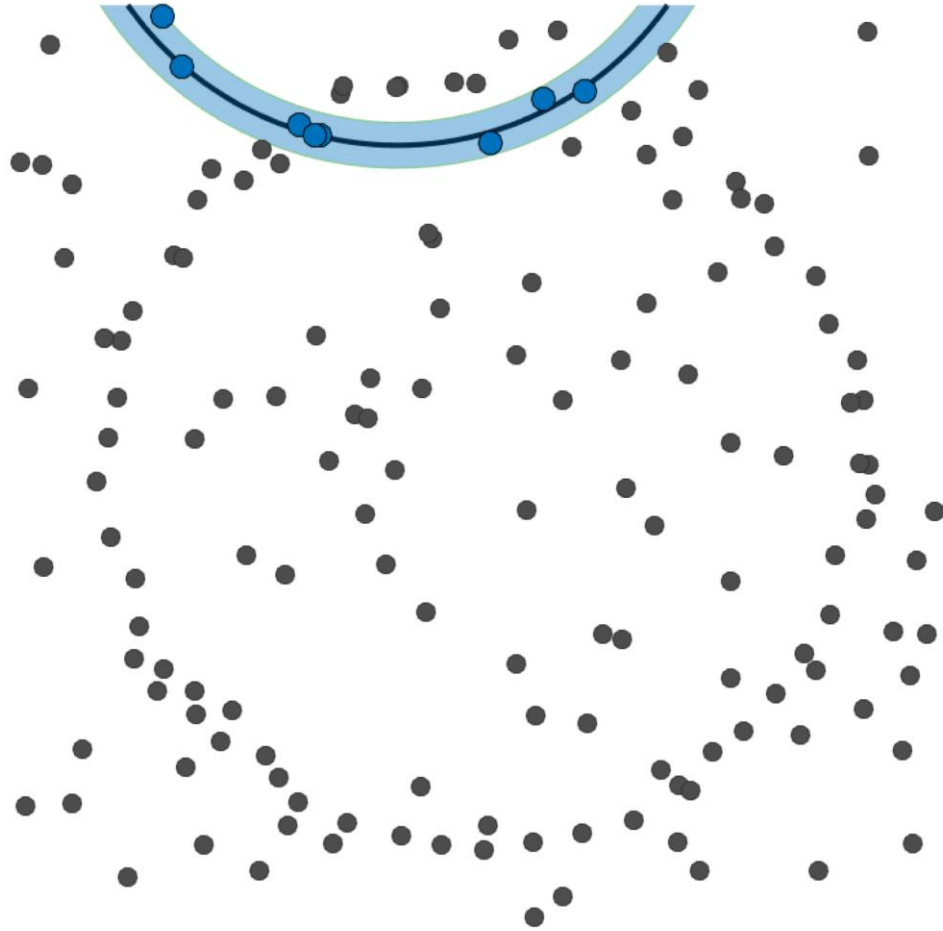
**end**

$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

# Randomized Sample Consensus [Fischler and Bolles 1981]



**Input:**  $X$  data,  $\epsilon$  inlier threshold,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = -\infty, k = 0;$

**repeat**

    Select randomly a minimal sample set  $S \subset X;$

    Estimate parameters  $\theta$  on  $S;$

    Evaluate  $J(\theta) = \sum_{x \in X} \hat{f}_\epsilon(r(x, \theta));$

**if**  $J(\theta) > J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

**end**

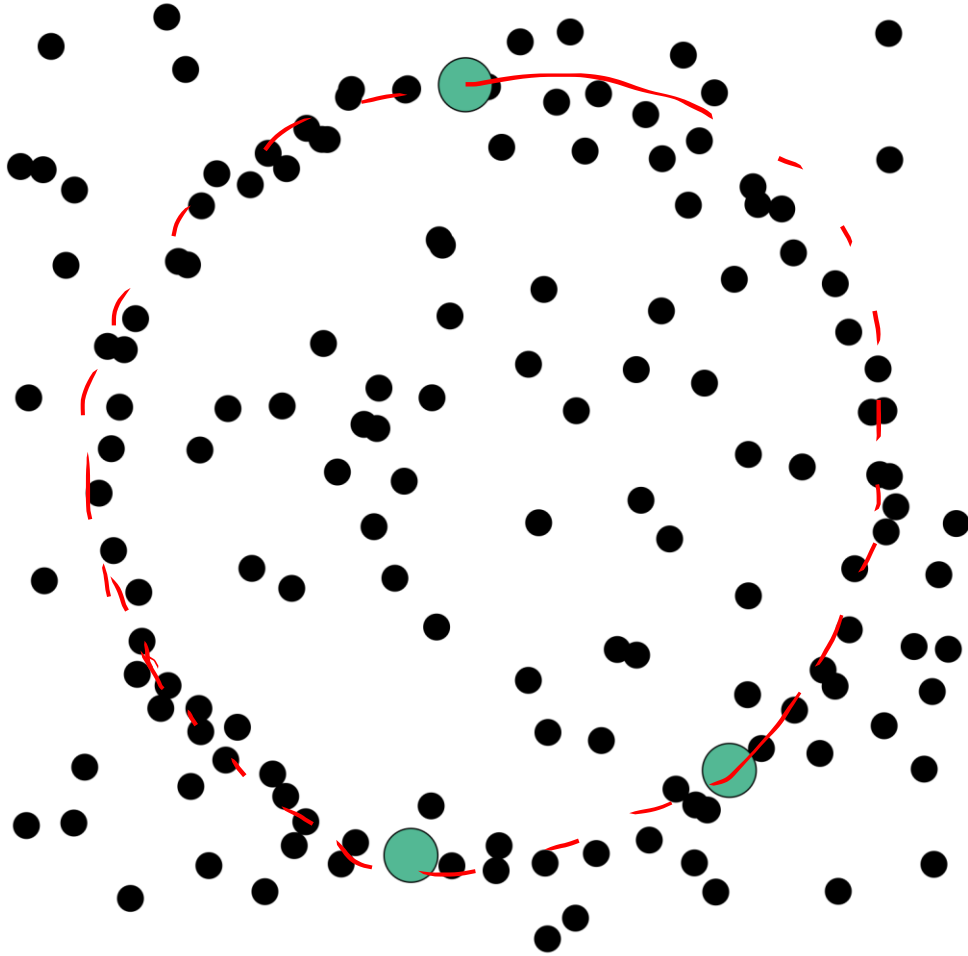
$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.



# Randomized Sample Consensus [Fischler and Bolles 1981]



**Input:**  $X$  data,  $\epsilon$  inlier threshold,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = -\infty, k = 0;$

**repeat**

Select randomly a minimal sample set  $S \subset X;$

Estimate parameters  $\theta$  on  $S;$

Evaluate  $J(\theta) = \sum_{x \in X} \hat{f}_\epsilon(r(x, \theta));$

**if**  $J(\theta) > J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

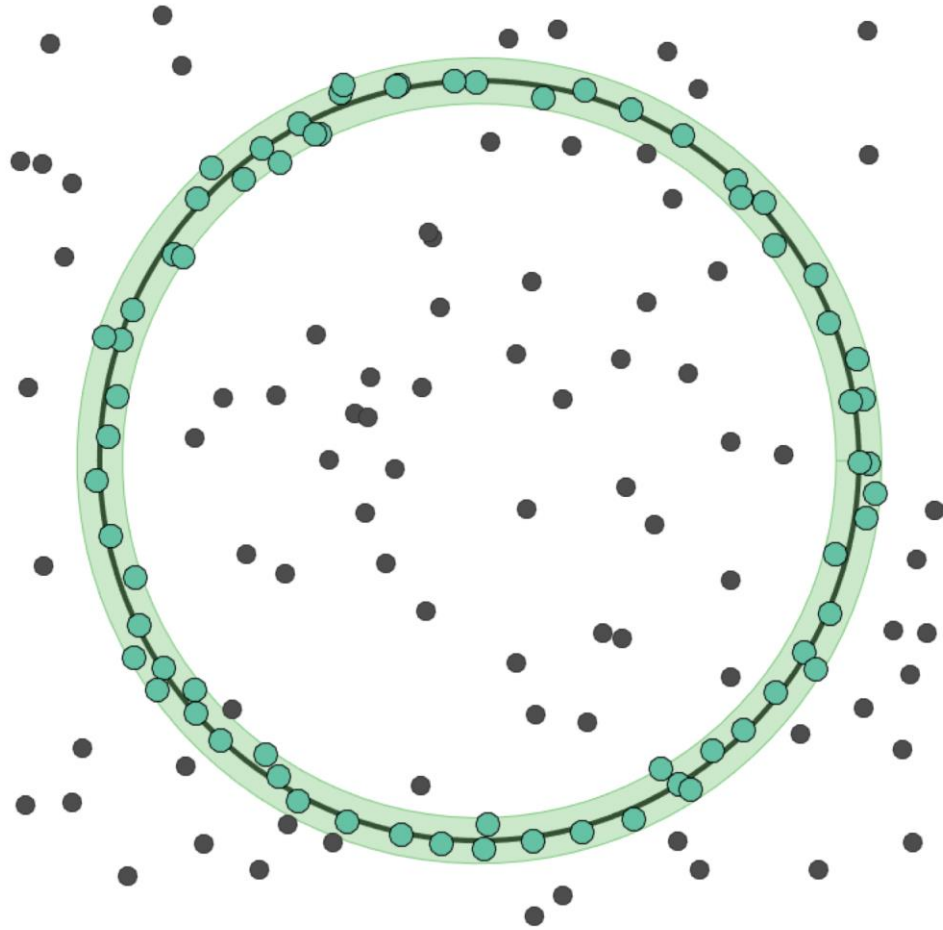
**end**

$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

# Randomized Sample Consensus [Fischler and Bolles 1981]



**Input:**  $X$  data,  $\epsilon$  inlier threshold,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = -\infty, k = 0;$

**repeat**

    Select randomly a minimal sample set  $S \subset X;$

    Estimate parameters  $\theta$  on  $S;$

    Evaluate  $J(\theta) = \sum_{x \in X} \hat{f}_\epsilon(r(x, \theta));$

**if**  $J(\theta) > J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

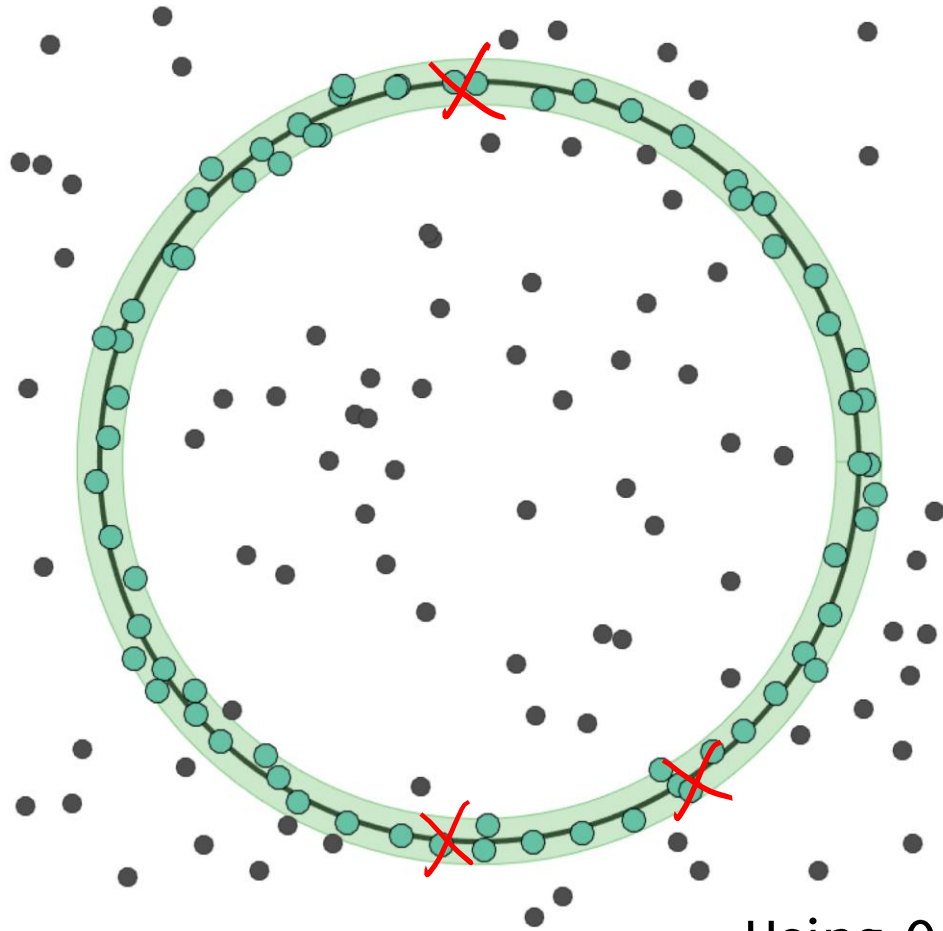
**end**

$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

# Randomized Sample Consensus [Fischler and Bolles 1981]



Using OLS,  
Increase stability of  
the results

**Input:**  $X$  data,  $\epsilon$  inlier threshold,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = -\infty, k = 0;$

**repeat**

    Select randomly a minimal sample set  $S \subset X;$

    Estimate parameters  $\theta$  on  $S;$

    Evaluate  $J(\theta) = \sum_{x \in X} \hat{f}_\epsilon(r(x, \theta));$

**if**  $J(\theta) > J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

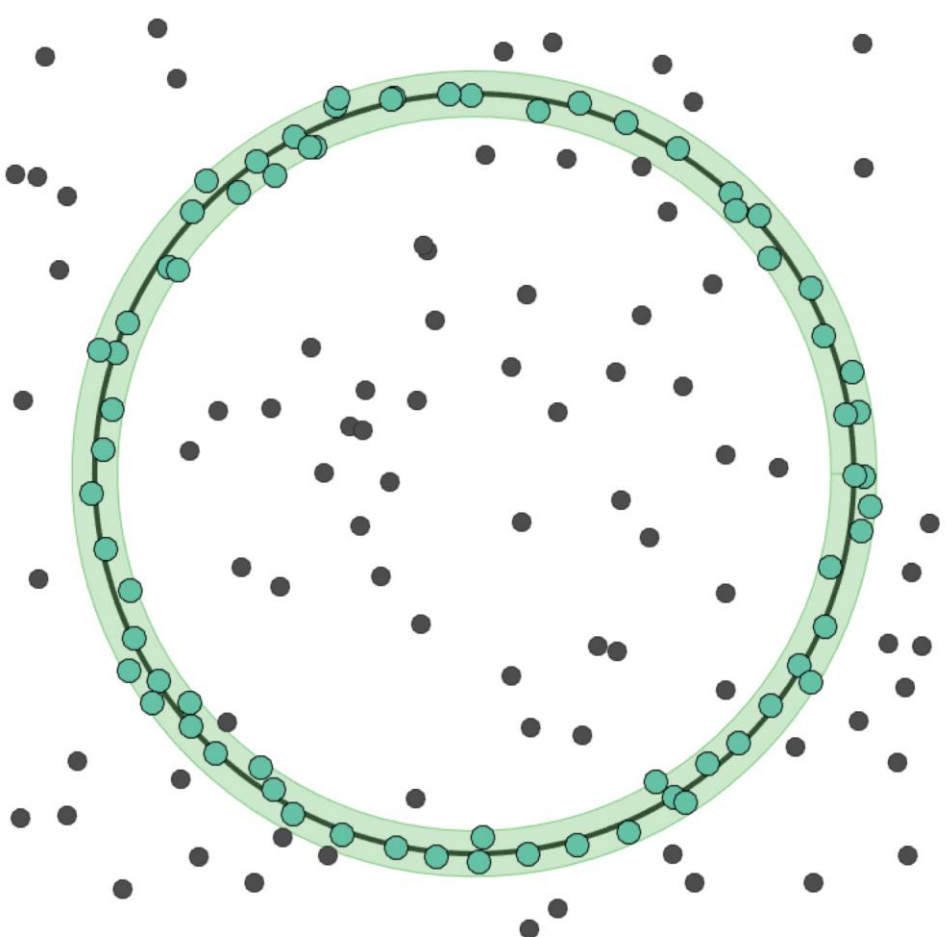
**end**

$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

# Randomized Sample Consensus

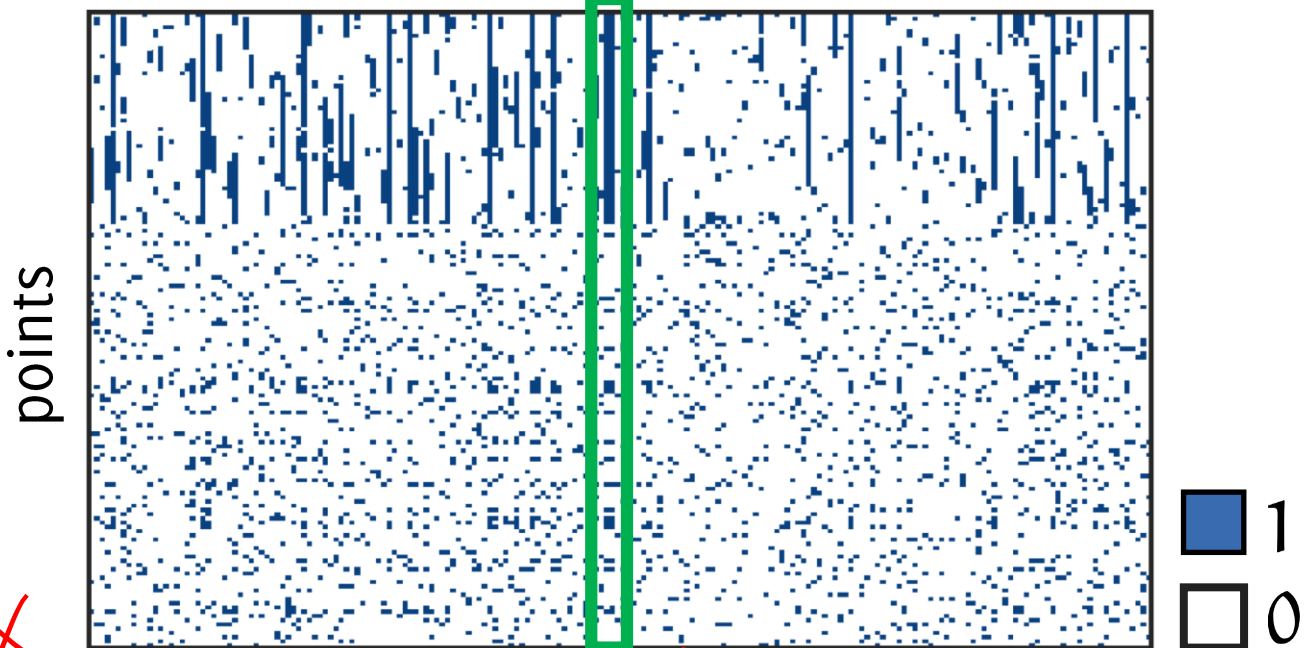


Data driven search of model space

$$H = \{\theta_1, \theta_2, \dots, \theta_m\} \approx \Theta$$

tentative models

$k_{max}$

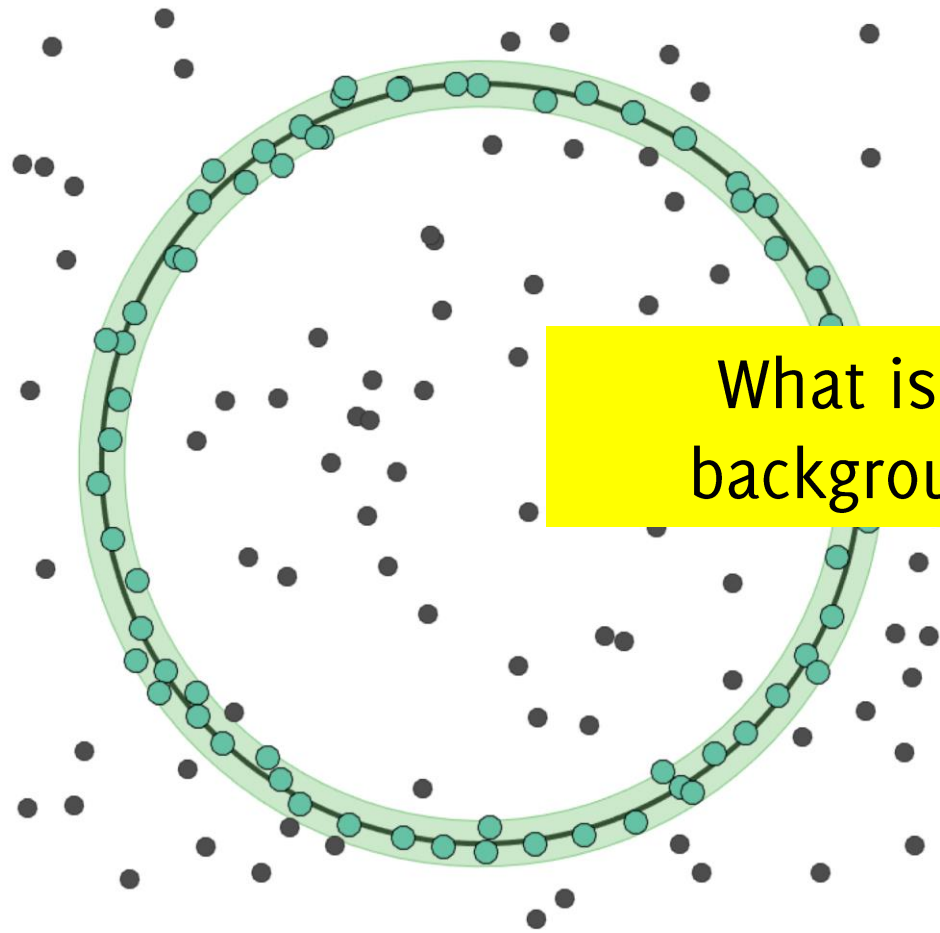


# X

$\hat{\theta}$

pick the column with the maximum sum

# Randomized Sample Consensus

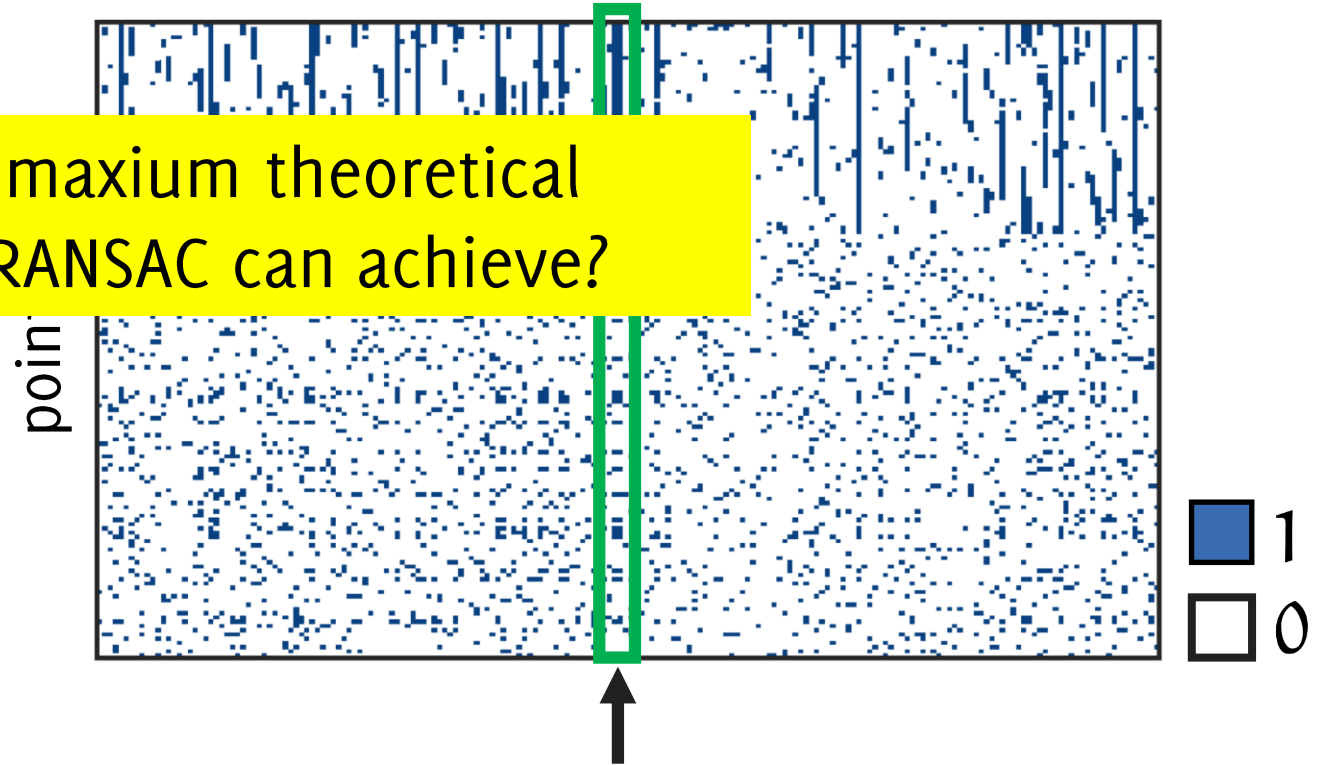


Data driven search of model space

$$H = \{\theta_1, \theta_2, \dots, \theta_m\} \approx \Theta$$

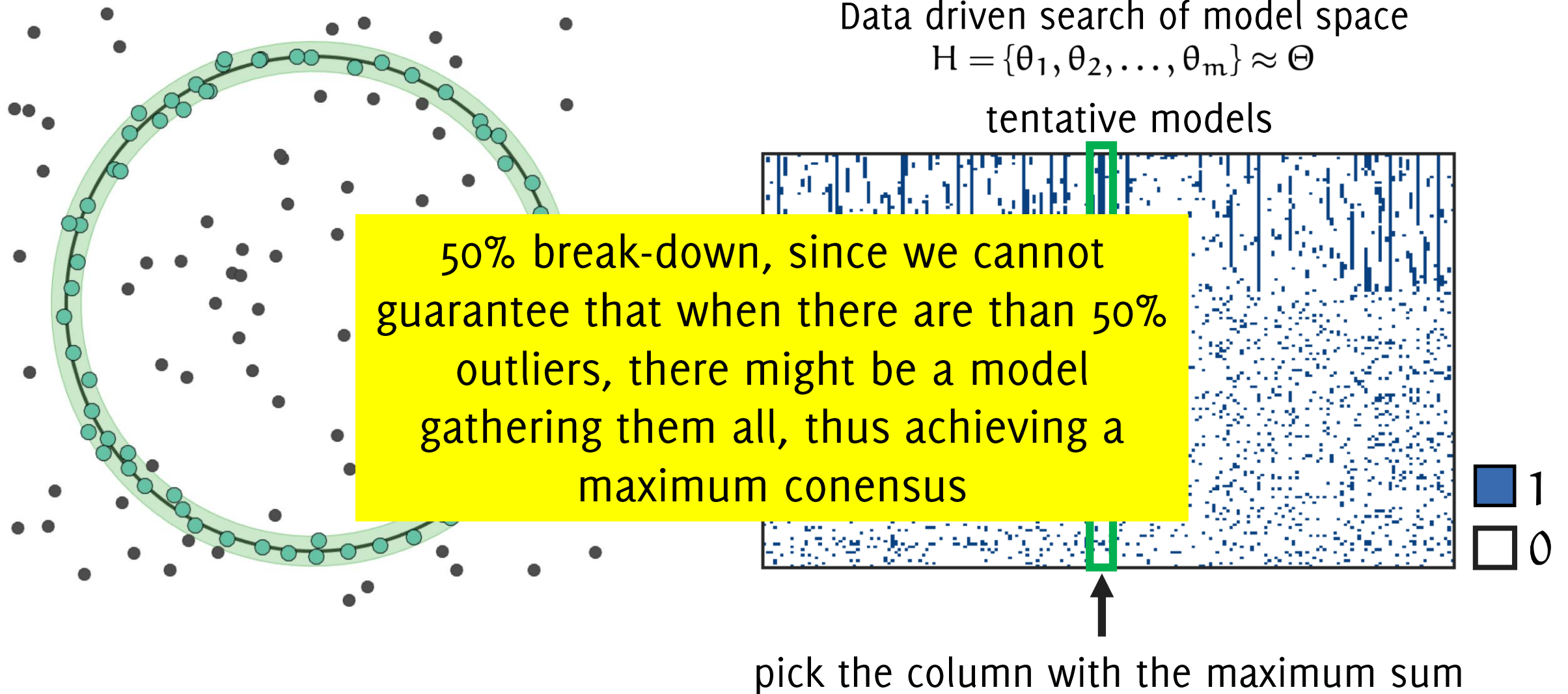
tentative models

What is the maximum theoretical background RANSAC can achieve?



pick the column with the maximum sum

# Randomized Sample Consensus



# Ransac: practical issues

**The size of the minimum sample  $S$ :** this is the bare minimum number of points to fit the parametric model at hand

**The inlier threshold  $\epsilon$ :** this can be estimated from the noise in the data

**The number of iterations  $n$ :**  <sup>$k_{max}$</sup>  the criteria for selecting the number of samples  $n$ : “Choose  $n$  so that, with probability  $p$ , at least **one random sample is without outliers** (e.g.  $p = 0.99$ ) “



# The maximum number of iterations $n$

Let  $e$  the probability of a sample to be an outlier and  $1 - e$  the probability of an inlier (can be estimated / provided by a-priori information)

The probability that all  $s$  points are inliers:  $(1 - e)^s$

$s = \# S$

The probability that at least one point in  $S$  is an outlier:  $1 - (1 - e)^s$   
(this is the probability for a sample  $S$  to yield the right model)

The probability that all the  $n$  selected set contain outliers

$$(1 - (1 - e)^s)^n$$

The probability that at least one the  $n$  set is without outliers:

$$1 - (1 - (1 - e)^s)^n$$

Set  $n$  to have the above probability below a parameter  $p$

$$p = 1 - (1 - (1 - e)^s)^n \rightarrow n = \log(1 - p) / \log(1 - (1 - e)^s)$$



# Ransac: practical issues

Choose  $n$  so that, with probability  $p$ , at least one random sample is free from outliers (e.g.  $p = 0.99$ ) (outlier ratio:  $e$ )

$$n = \log(1 - p) / \log(1 - (1 - e)^s)$$

s	proportion of outliers $e$						
	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

# Ransac: details

Repeat  $n$  times:

- Draw  $s$  points uniformly at random
- Fit line to these  $s$  points
- Find inliers to this line among the remaining points (i.e., points whose distance from the line is less than  $t$ )

- Update  $n$

- $e = 1 - \frac{\text{number of inliers}}{\text{number of points}}$

- $n = \log(1 - p) / \log(1 - (1 - e)^s)$

Choose the best model

Re-estimate the line with the inliers only through ordinary least square

# Ransac

## Pros:

- very popular (>22900 citations in Google Scholar)
- many improvements have been proposed
- very versatile
- agnostic on outlier percentage
- mild assumption: know the scale noise to set the inlier threshold  $\epsilon$

## Cons:

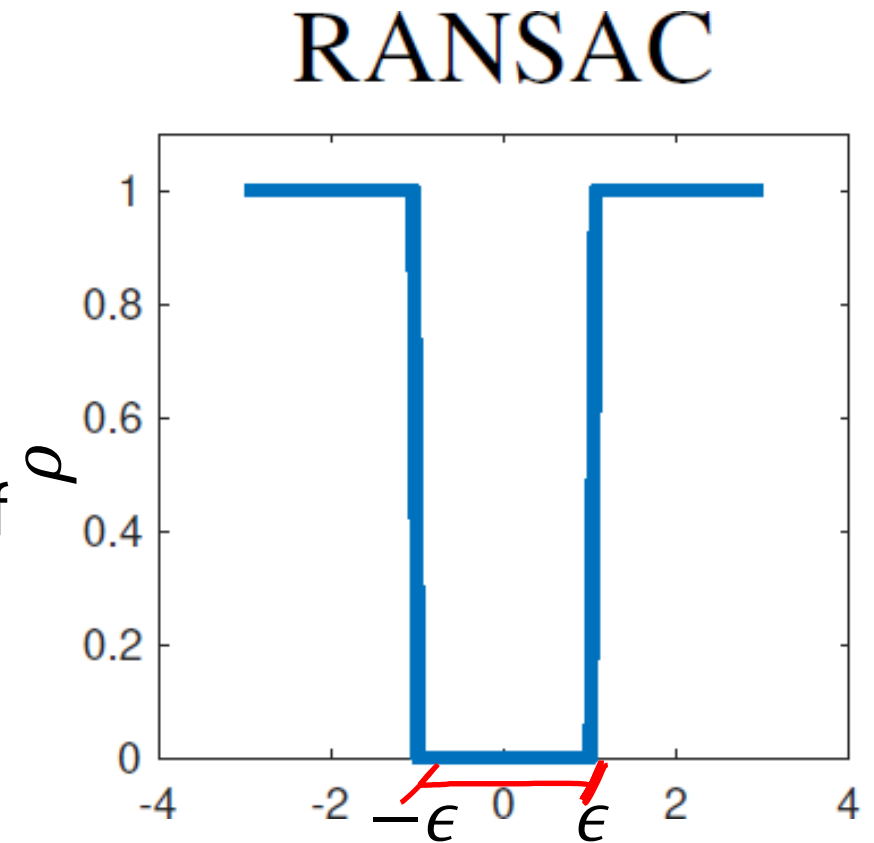
- can take longer than expected

# Ransac as M-estimator

(Steward 1999) Ransac can be seen as a particular M-estimator since the **loss it minimizes** is the number of points having residual above the inlier threshold  $\epsilon$

$$f(r_i) = \begin{cases} 1, & r_i > \epsilon \\ 0, & r_i \leq \epsilon \end{cases}$$

Of course selecting inlier threshold  $\epsilon$  is very critical  
Ransac achieves a theoretical breakdown of 50% of outliers, but in practice, provided a good selection of  $\epsilon$ , this can be even higher

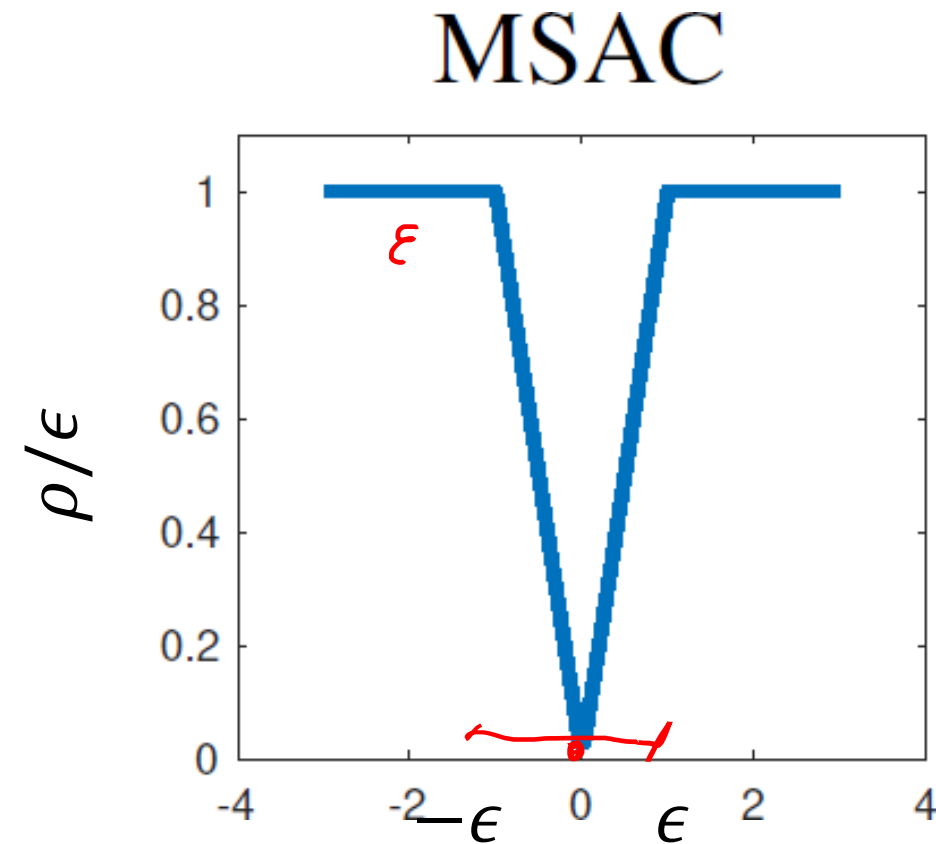


# MSAC

(Torr and Zisserman 2000) a different loss function to be minimized within the RanSaC framework

$$f(r_i) = \begin{cases} r_i, & r_i > \epsilon \\ \epsilon, & r_i \leq \epsilon \end{cases}$$

This turns to be more effective and should be preferred to RanSaC



# Ransac vs MSaC

**Input:**  $X$  data,  $\epsilon$  inlier threshold,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = -\infty, k = 0;$

**repeat**

Select randomly a minimal sample set  $S \subset X;$

Estimate parameters  $\theta$  on  $S;$

Evaluate  $J(\theta) = \sum_{x \in X} \hat{f}_\epsilon(r(x, \theta));$

**if**  $J(\theta) > J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

**end**

$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

**Input:**  $X$  data,  $\epsilon$  inlier threshold,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = +\infty, k = 0;$

**repeat**

Select randomly a minimal sample set  $S \subset X;$

Estimate parameters  $\theta$  on  $S;$

Estimate inlier set  $I = \{x \in X: r(x, \theta)^2 < \epsilon^2\};$

Evaluate  $J(\theta) = \sum_{x \in I} r(x, \theta) + (|X| - |I|)\epsilon;$

**if**  $J(\theta) < J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

**end**

$k = k + 1;$

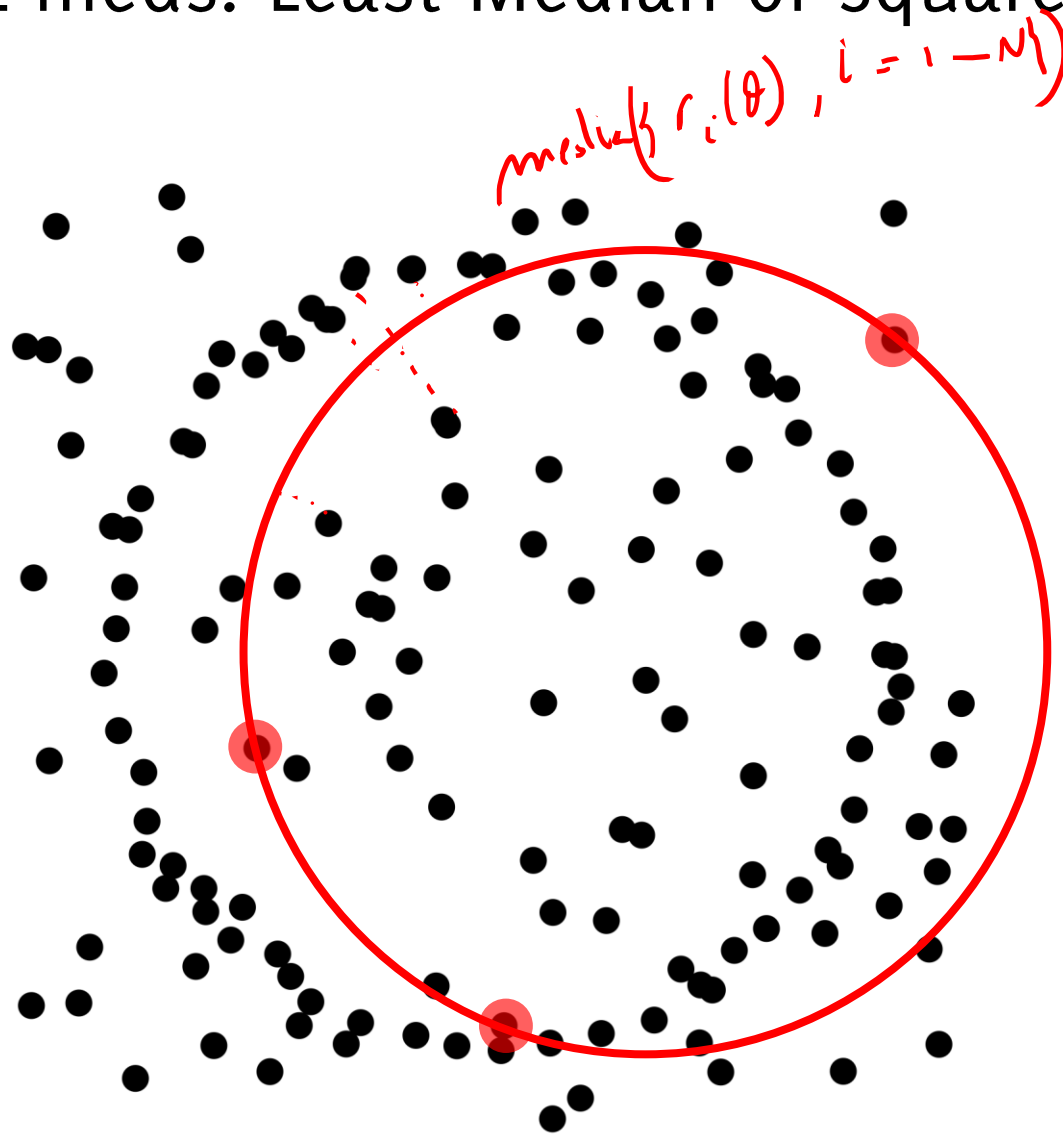
**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

$\epsilon$

# Least Median of Squares

# L-med: Least Median of Squares, Rousseeuw e Leroy (1987)



**Input:**  $X$  data,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = +\infty, k = 0;$

**repeat**

    Select randomly a minimal sample set  $S \subset X;$

    Estimate parameters  $\theta$  on  $S;$

    Evaluate  $J(\theta) = \text{median}_{x \in X}(r(x, \theta));$

**if**  $J(\theta) < J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

**end**

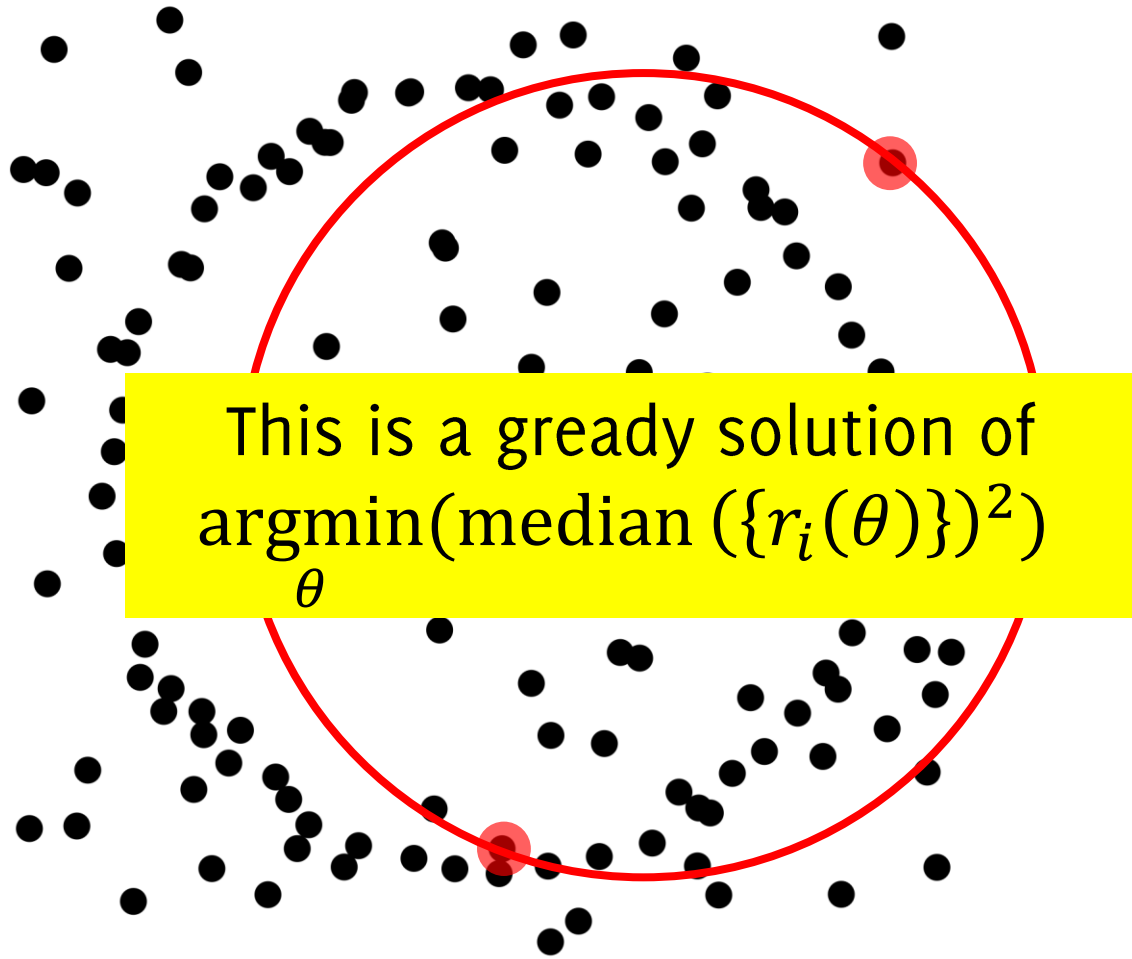
$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.



# L-med: Least Median of Squares, Rousseeuw e Leroy (1987)



**Input:**  $X$  data,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = +\infty, k = 0;$

**repeat**

    Select randomly a minimal sample set  $S \subset X;$

    Estimate parameters  $\theta$  on  $S;$

    Evaluate  $J(\theta) = \operatorname{median}_{x \in X}(r(x, \theta));$

**if**  $J(\theta) < J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

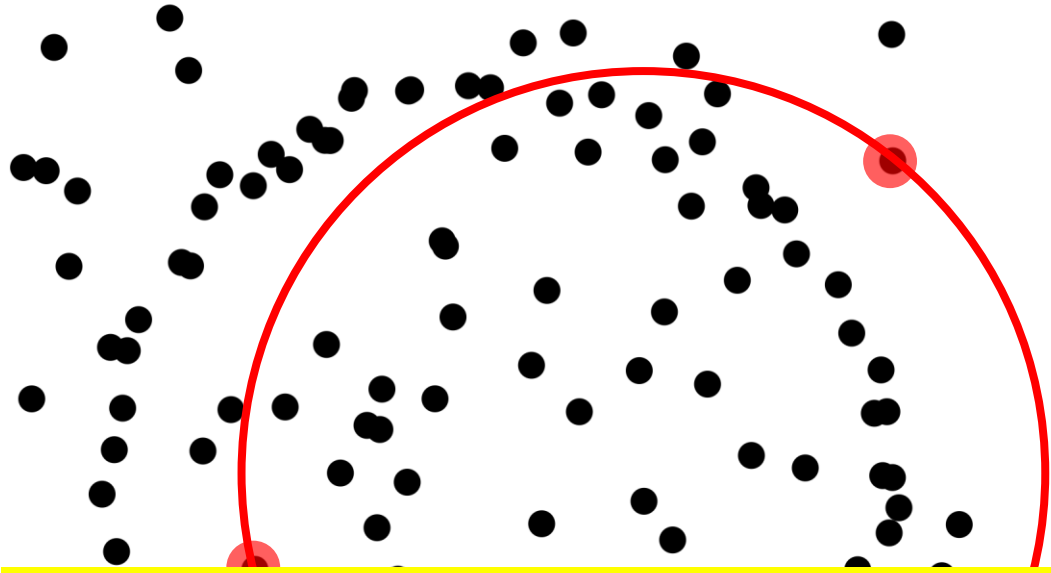
**end**

$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

# L-med: Least Median of Squares, Rousseeuw e Leroy (1987)



Since there is no explicit definition of inliers here, inliers can be identified as points having residuals (w.r.t. to the final model) that are smaller than  $2.5\sigma$

**Input:**  $X$  data,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = +\infty, k = 0;$

**repeat**

    Select randomly a minimal sample set  $S \subset X;$

    Estimate parameters  $\theta$  on  $S;$

    Evaluate  $J(\theta) = \text{median}_{x \in X}(r(x, \theta));$

**if**  $J(\theta) < J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

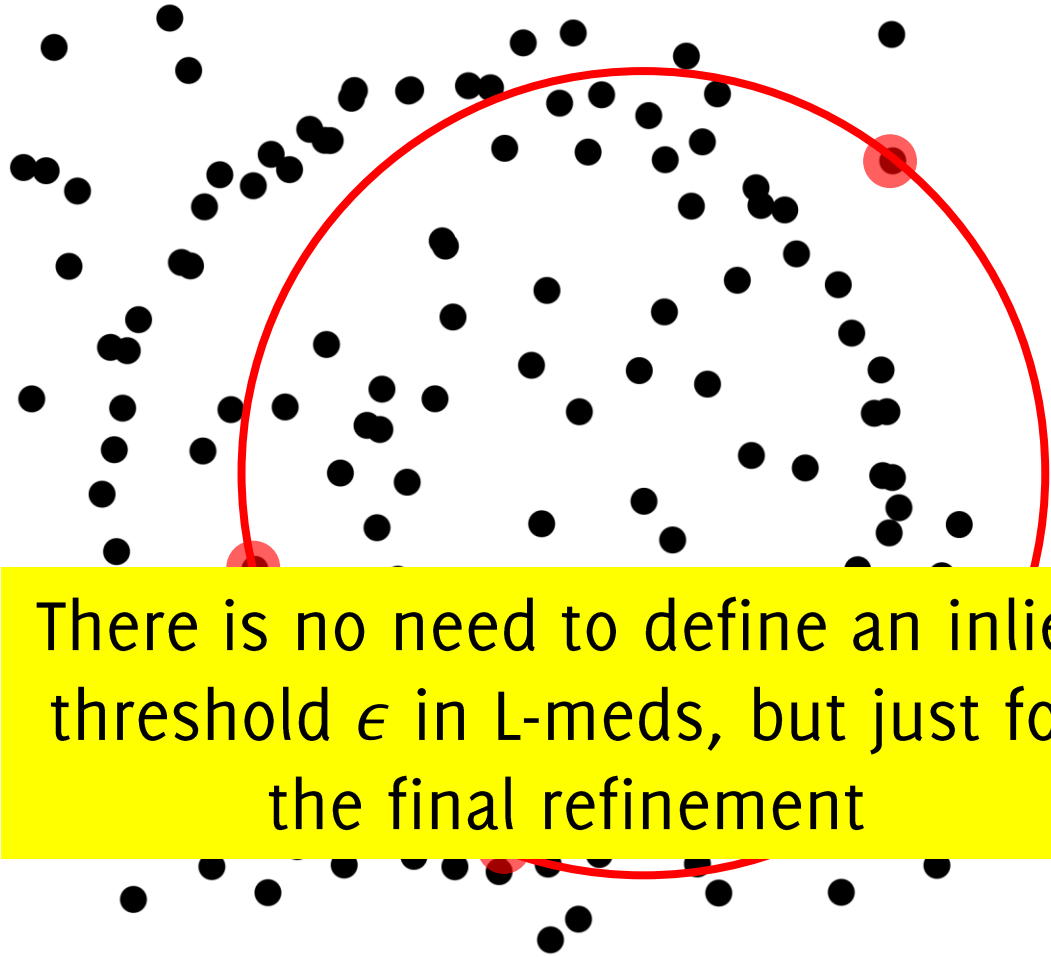
**end**

$k = k + 1;$

**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

# L-meds: Least Median of Squares, Rousseeuw e Leroy (1987)



**Input:**  $X$  data,  $k_{\max}$  max iteration

**Output:**  $\theta^*$  model estimate

$J^* = +\infty, k = 0;$

**repeat**

    Select randomly a minimal sample set  $S \subset X;$

    Estimate parameters  $\theta$  on  $S;$

    Evaluate  $J(\theta) = \text{median}_{x \in X}(r(x, \theta));$

**if**  $J(\theta) < J^*$  **then**

$\theta^* = \theta;$

$J^* = J(\theta);$

**end**

$k = k + 1;$

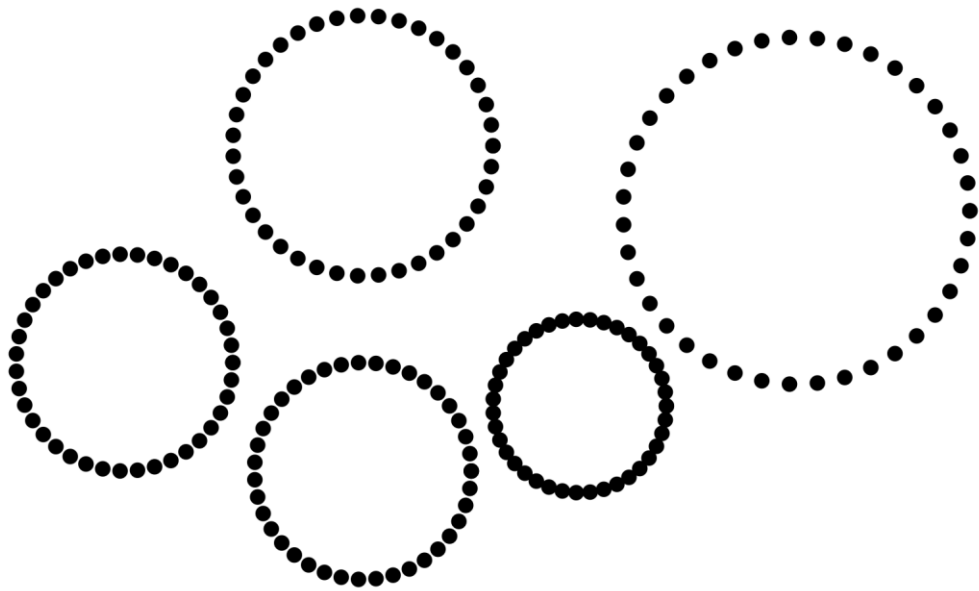
**until**  $k > k_{\max};$

Optimize  $\theta^*$  on its inliers.

# Multi-Model Fitting

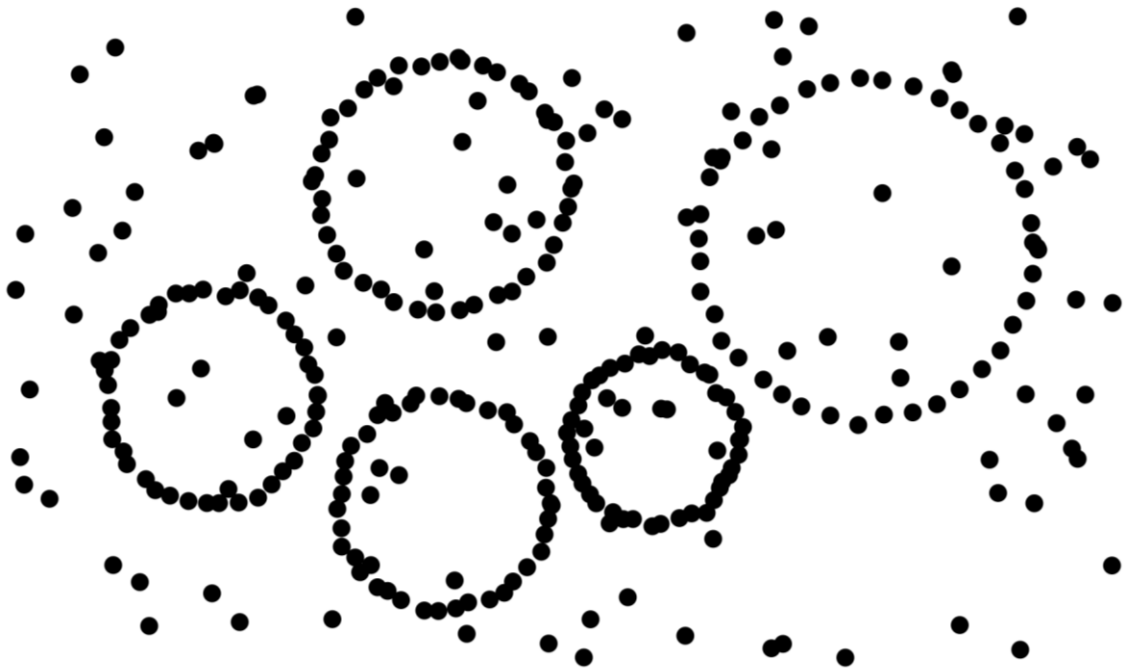
# The problem of multi-model fitting (or structure recovery)

**Given** a set of data  $X = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$ , possibly corrupted by noise and outliers, and a family of geometric models  $\Theta$



# The problem of multi-model fitting (or structure recovery)

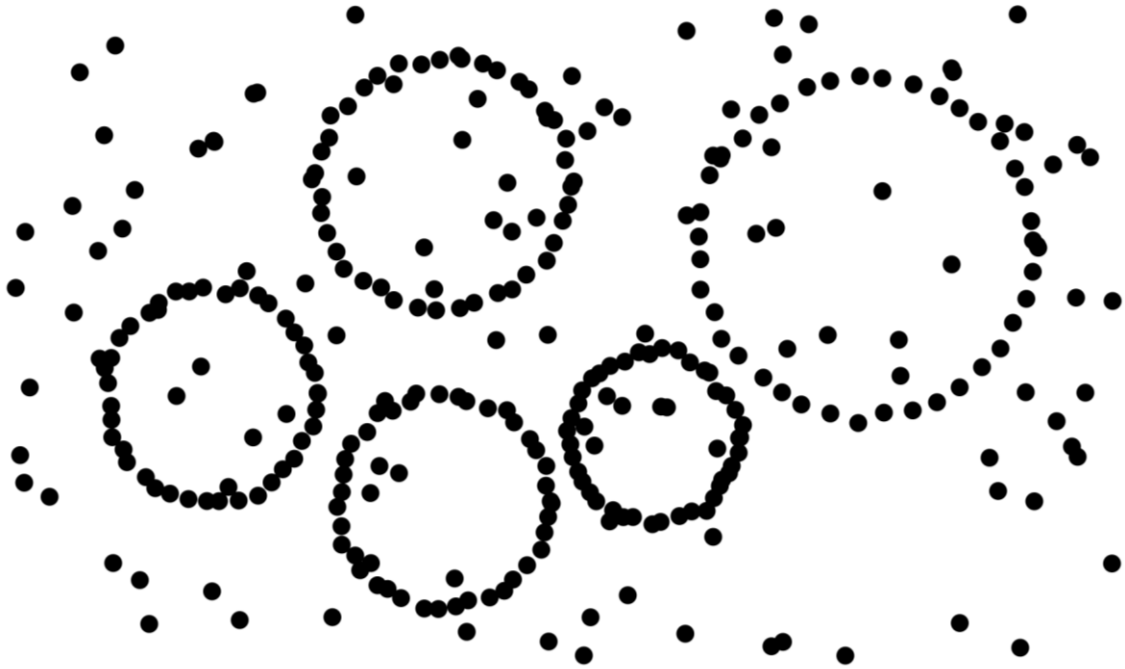
**Given** a set of data  $X = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$ , possibly corrupted by noise and outliers, and a family of geometric models  $\Theta$



# The problem of multi-model fitting (or structure recovery)

**Given** a set of data  $X = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$ , possibly corrupted by noise and outliers, and a family of geometric models  $\Theta$

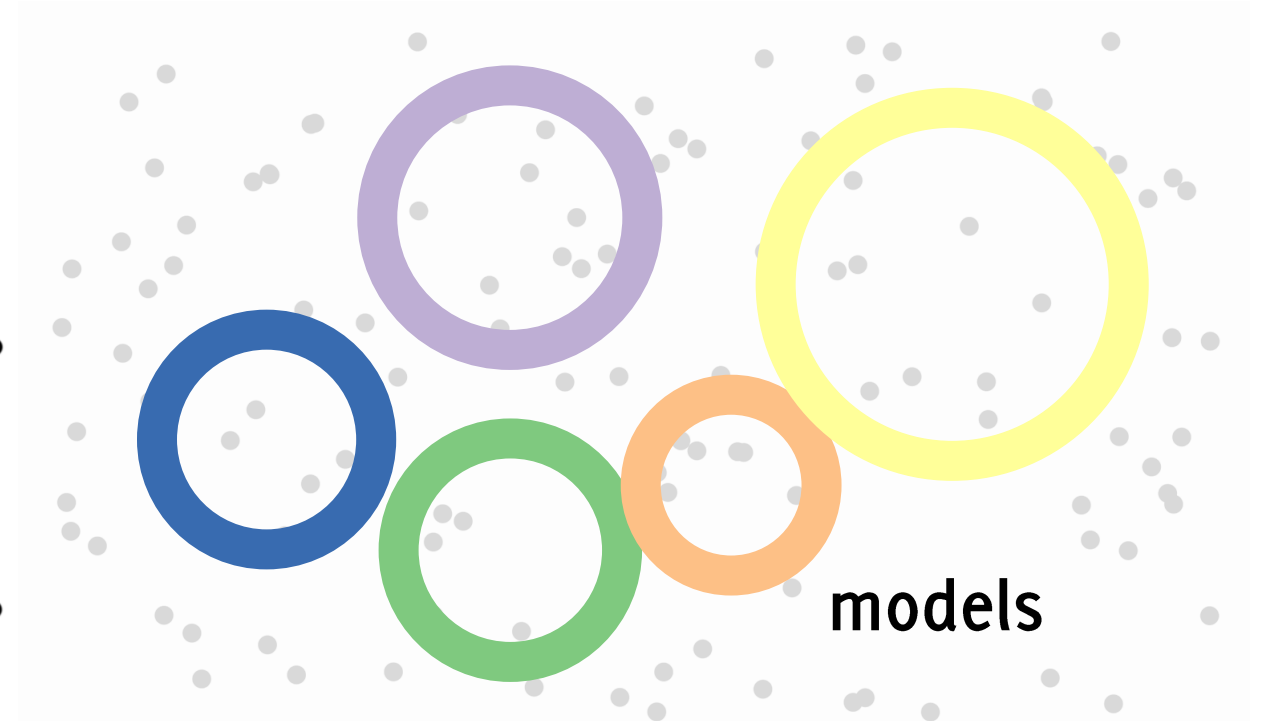
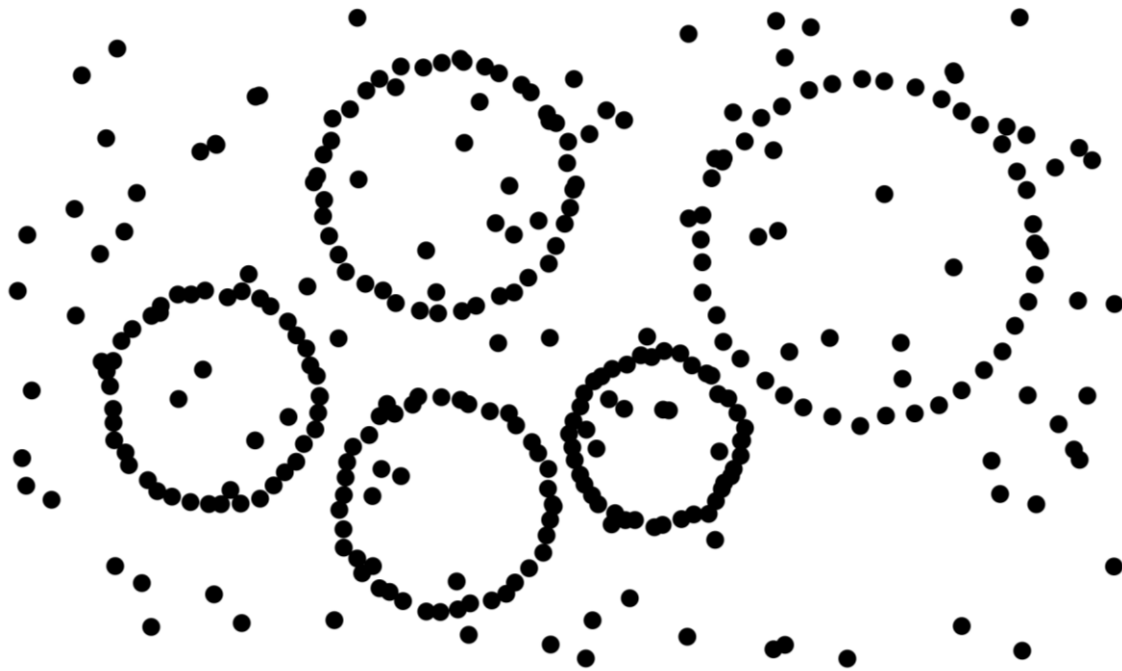
**Goal:** automatically estimate the models that best explain the data/discover the structures hidden in the data



# The problem of multi-model fitting (or structure recovery)

**Given** a set of data  $X = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$ , possibly corrupted by noise and outliers, and a family of geometric models  $\Theta$

**Goal:** automatically estimate the models that best explain the data/discover the structures hidden in the data



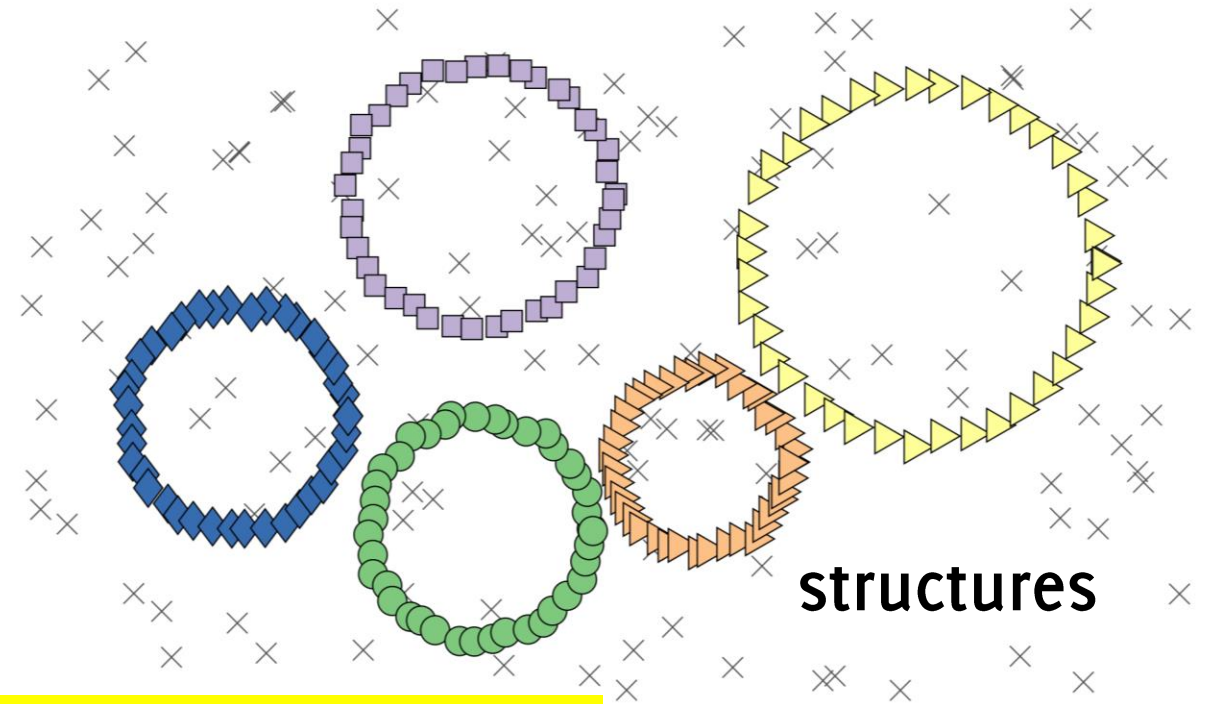
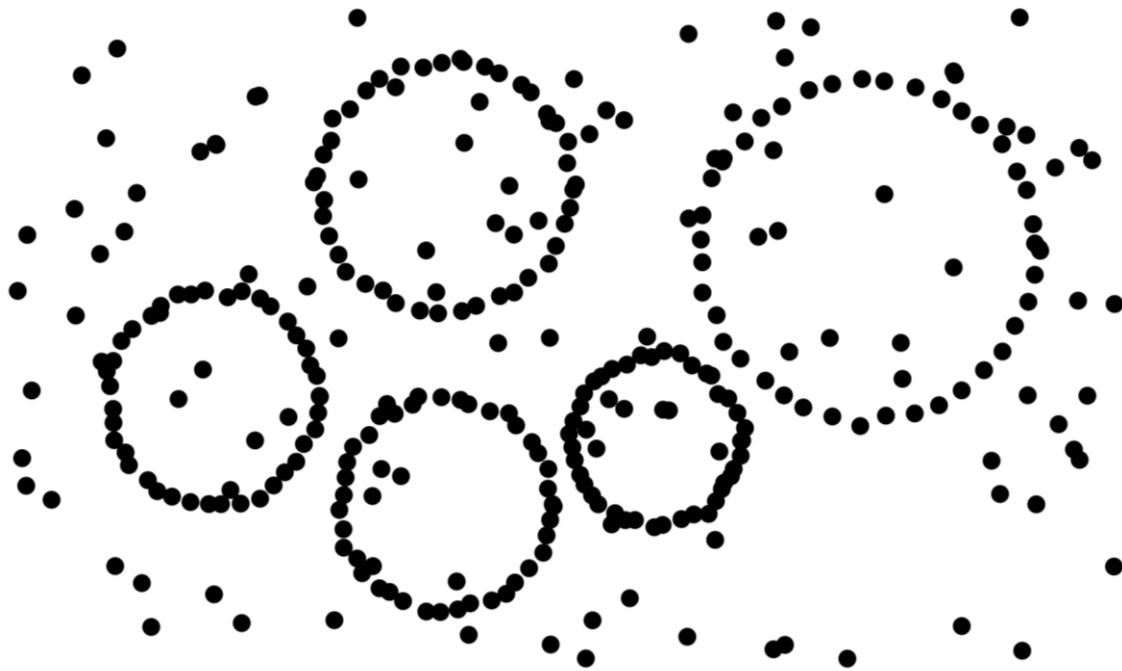
“in the eye of the beholder”, mathematical descriptions of the data that an observer fits



# The problem of multi-model fitting (or structure recovery)

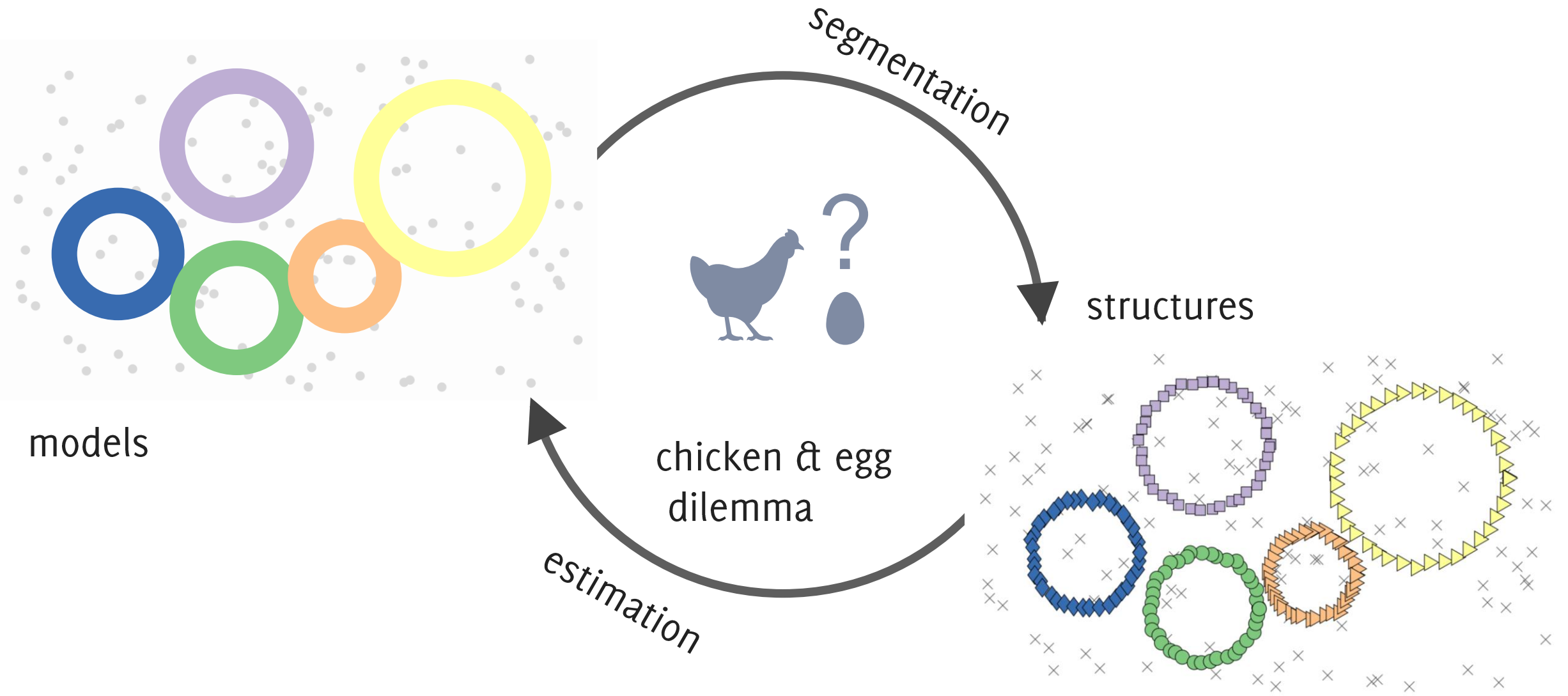
**Given** a set of data  $X = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$ , possibly corrupted by noise and outliers, and a family of geometric models  $\Theta$

**Goal:** automatically estimate the models that best explain the data/discover the structures hidden in the data



relations among the data, intrinsic to data

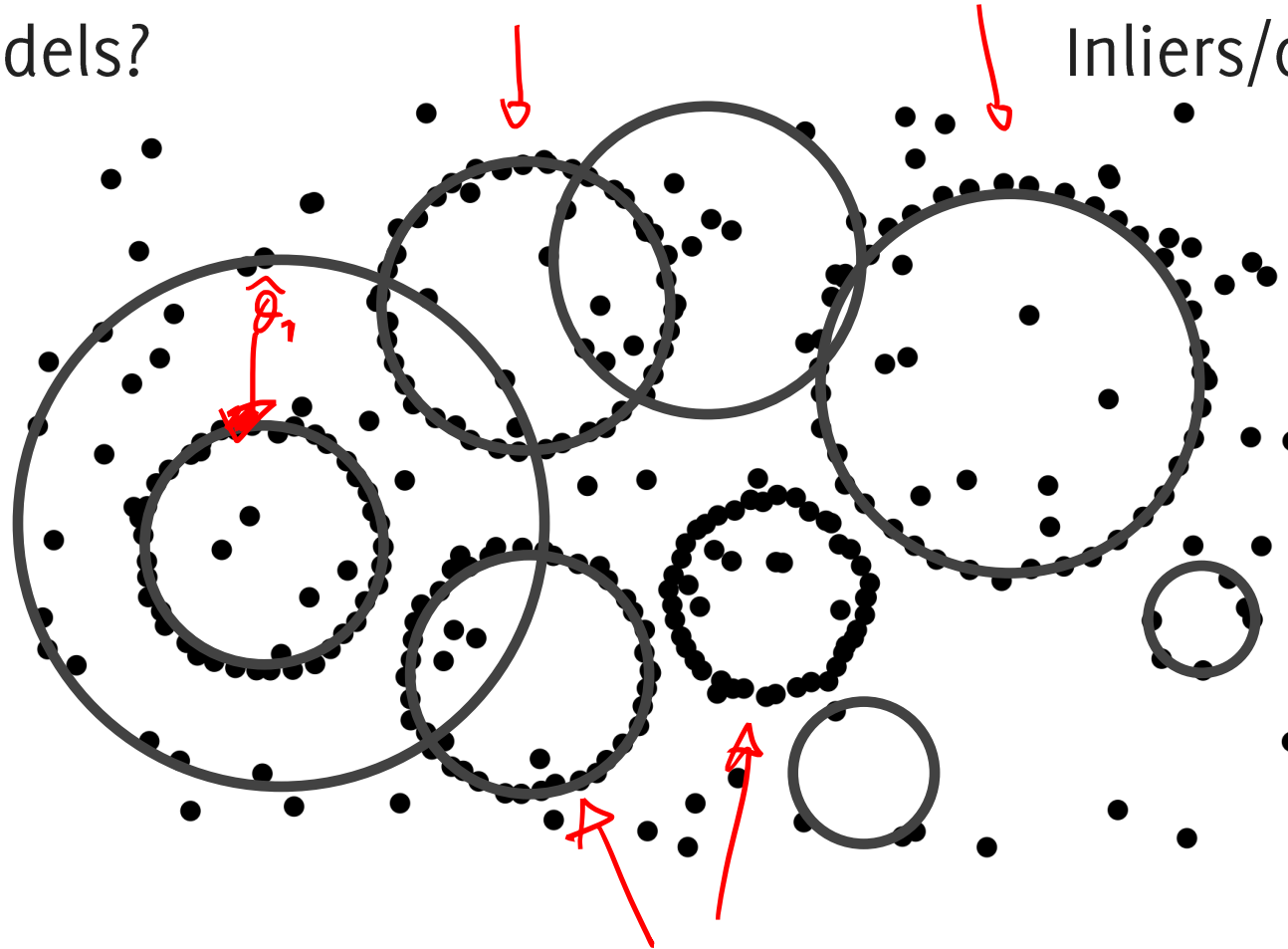
# The Challenges of multi-model fitting



# The Challenges of multi-model fitting

Number of models?

Inliers/outliers?




ill posed

# Multi-Model Fitting

Giacomo Boracchi

CVPR USI, May 8th 2020

# Outline

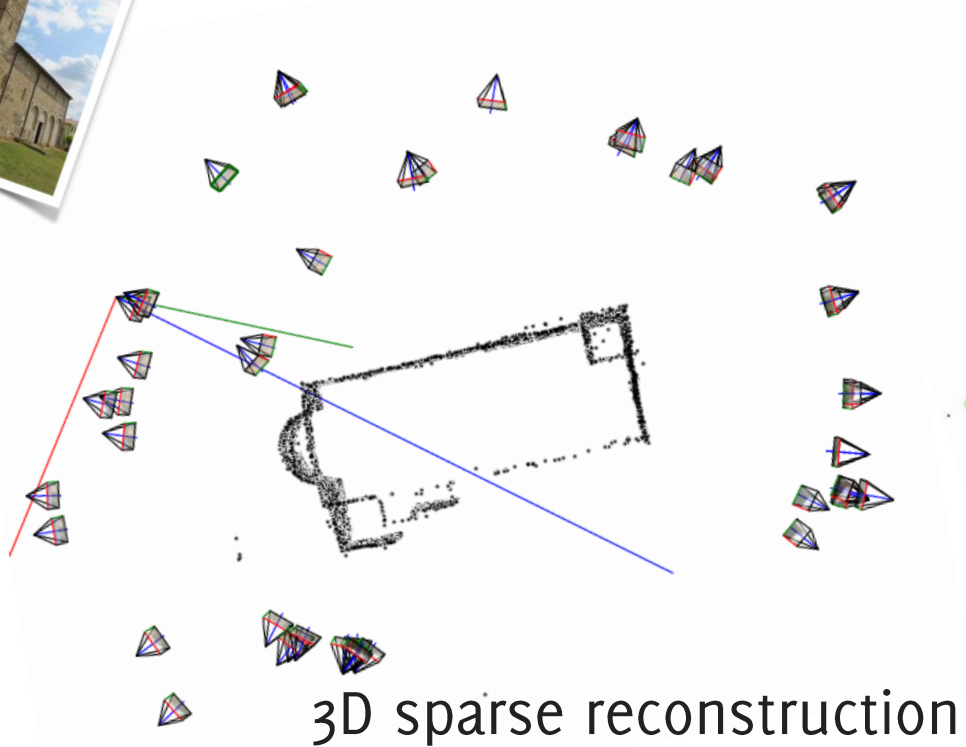
- Lab on Image Segmentation
- Preference-based methods: J-linkage 
- Project description
- Image Classification and Retrieval by Image features

# Multi model fitting applications: primitive fitting

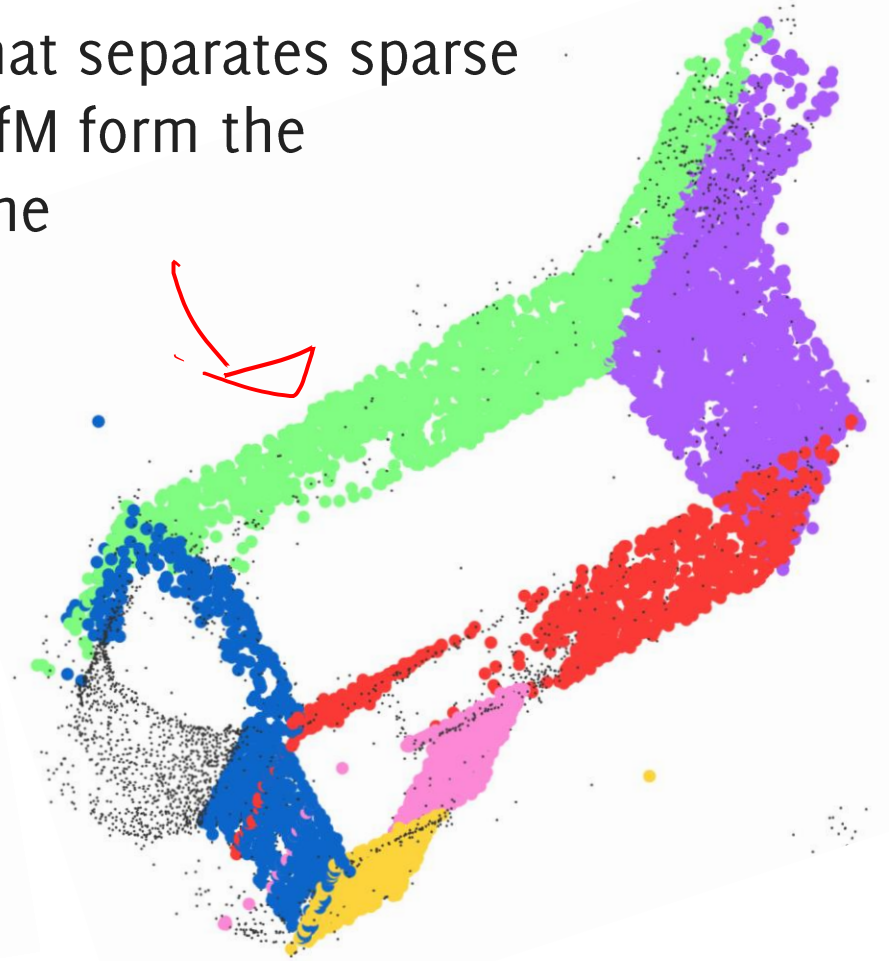


input images

Bridge the semantic gap that separates sparse point cloud coming from SfM from the understanding of a 3D scene

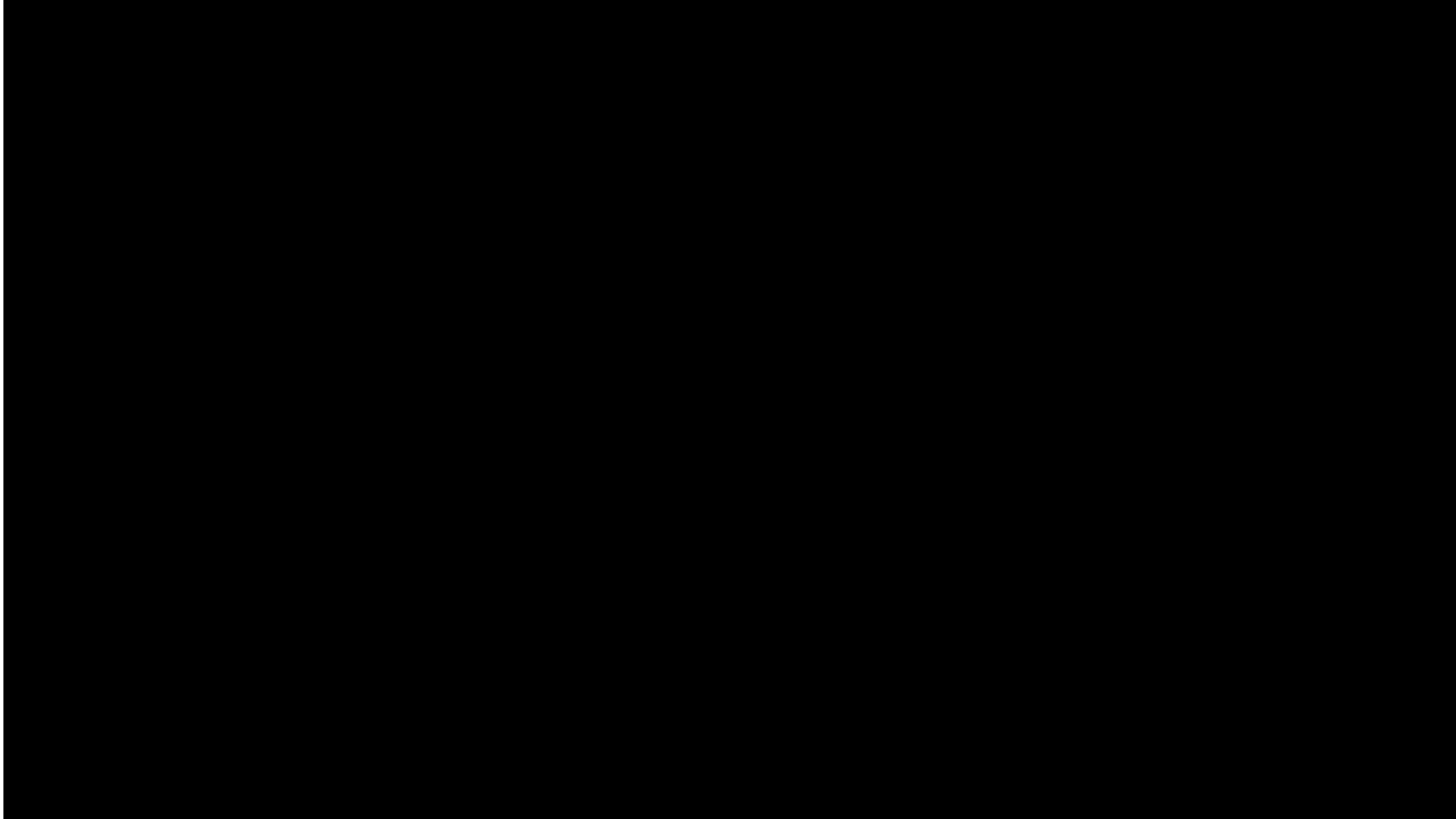


3D sparse reconstruction



$$X \subset \mathbb{R}^3, \Theta = \text{planes}$$

# Multimodel fitting for 3D scattered data



L. Magri, and A. Fusiello. "Reconstruction of interior walls from point cloud data with min-hashed J-linkage." 2018 3DV

L. Magri, and A. Fusiello. "IMPROVING AUTOMATIC RECONSTRUCTION OF INTERIOR WALLS FROM POINT CLOUD DATA." International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences (2019).

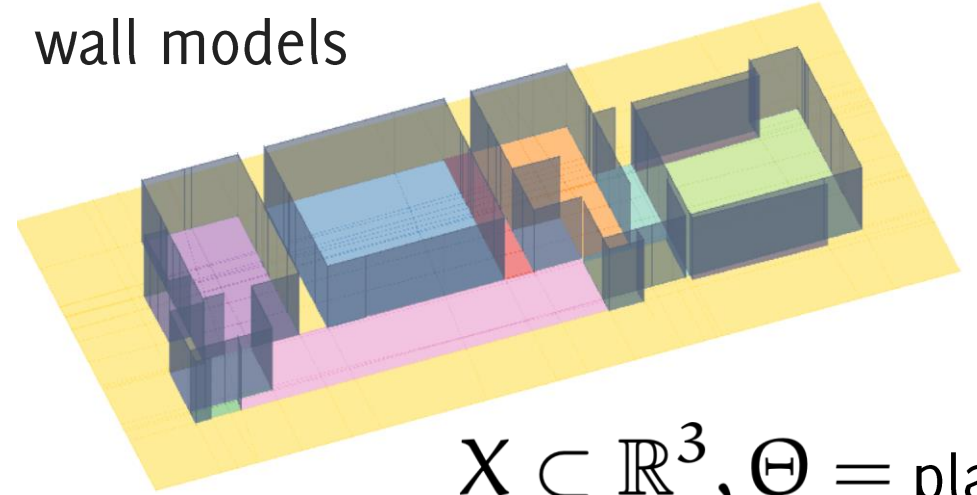
L. Magri, and Andrea Fusiello. "T-linkage: A continuous relaxation of j-linkage for multi-model fitting." CVPR 2014

# Multi model fitting applications: scan2bim

scanned point cloud



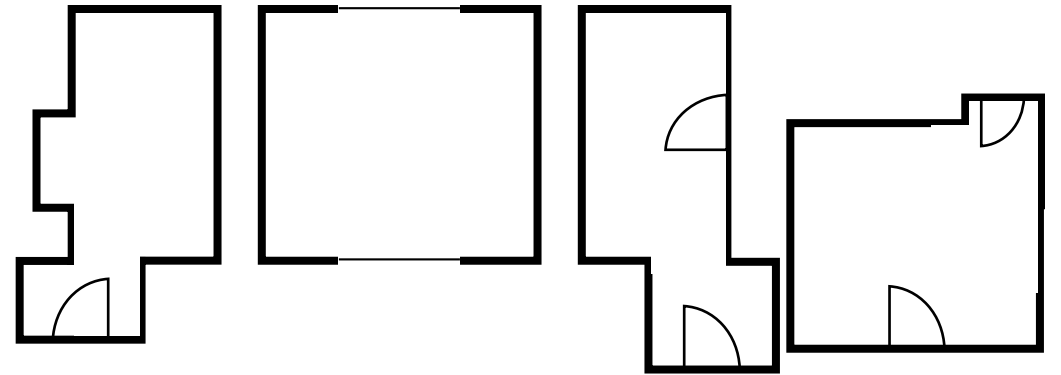
wall models



$$X \subset \mathbb{R}^3, \Theta = \text{planes}$$

Given a scanned point cloud of an interior environment, detect its primary facility surfaces – such as floors, walls, and ceilings.

floor-plan



$$X \subset \mathbb{R}^2, \Theta = \text{lines}$$



# Multi model fitting applications: two view geometry

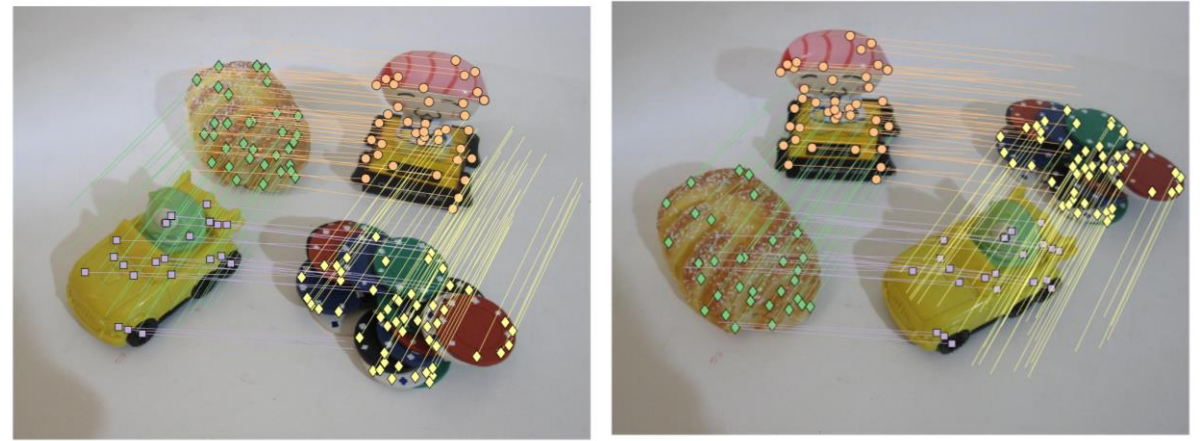
Geometric fit on corresponding matches across two images

plane detection



$X \subset \mathbb{R}^4, \Theta = \text{homographies}$

epipolar geometry

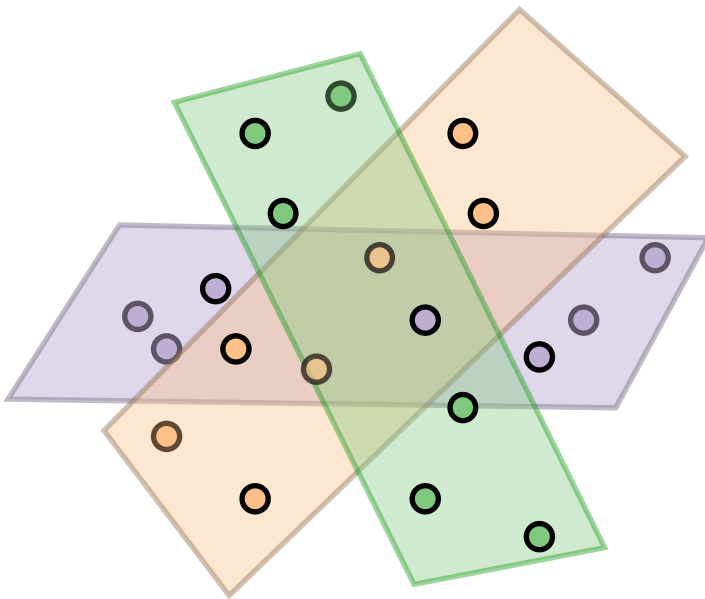


$X \subset \mathbb{R}^4, \Theta = \text{fundamental matrices}$

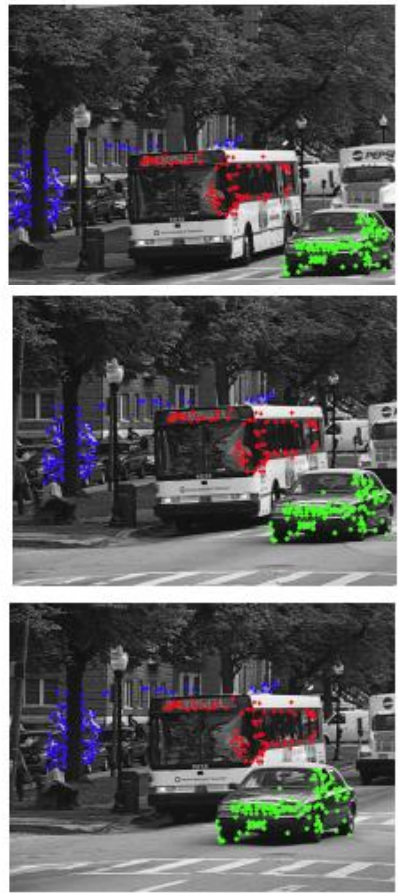
# Multi model fitting applications: subspace clustering

3D Video segmentation

Face clustering



$$X \subset \mathbb{R}^d, \Theta = \text{subspaces}$$



*Projection*



# Template Detection



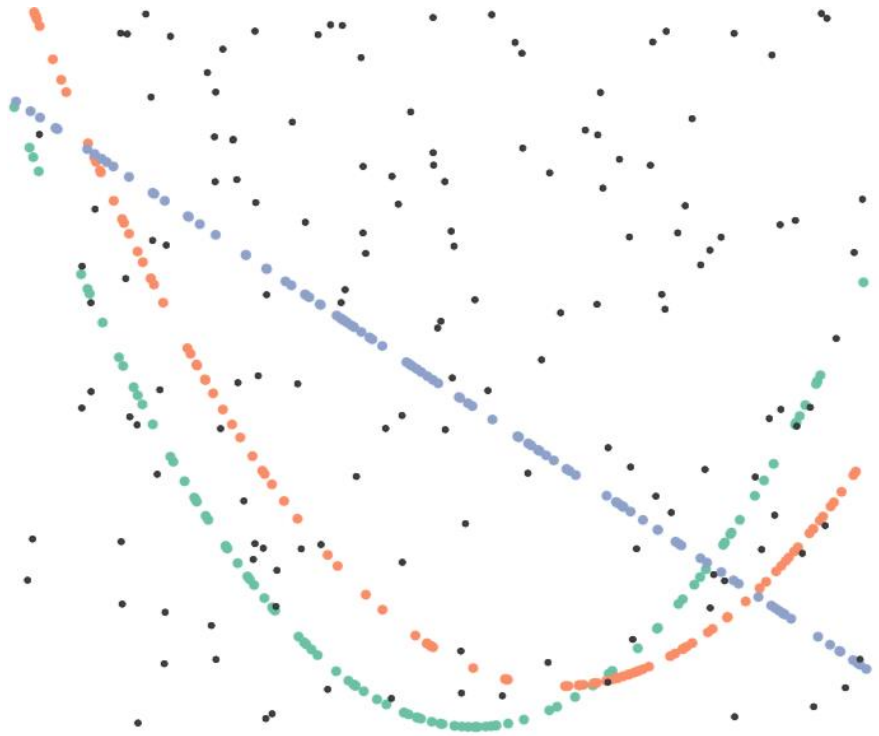
ASPIRINA

LASONIL

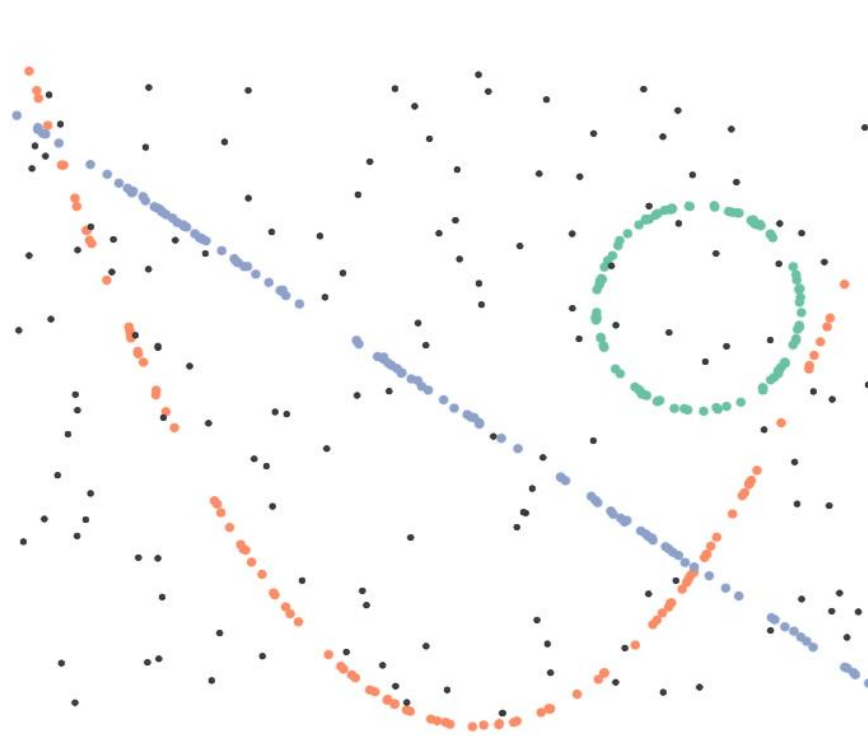




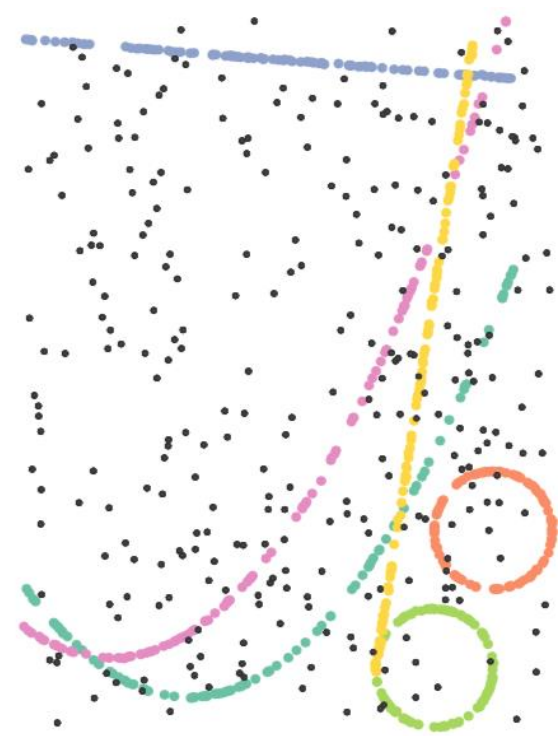
# Multimodel (and multi-class) fitting



(a)



(b)



(c)

# Multimodel (and multi-class) fitting

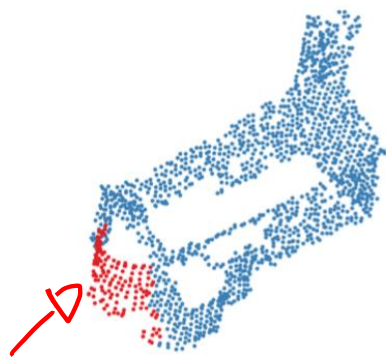
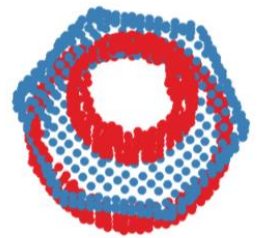
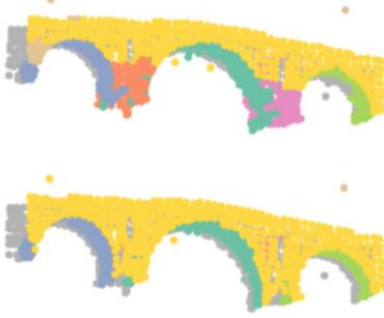
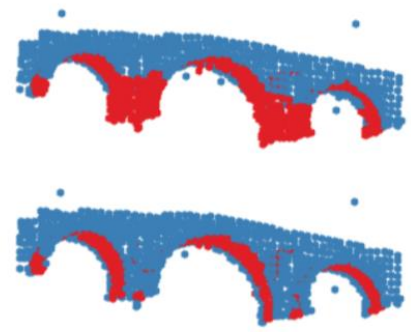


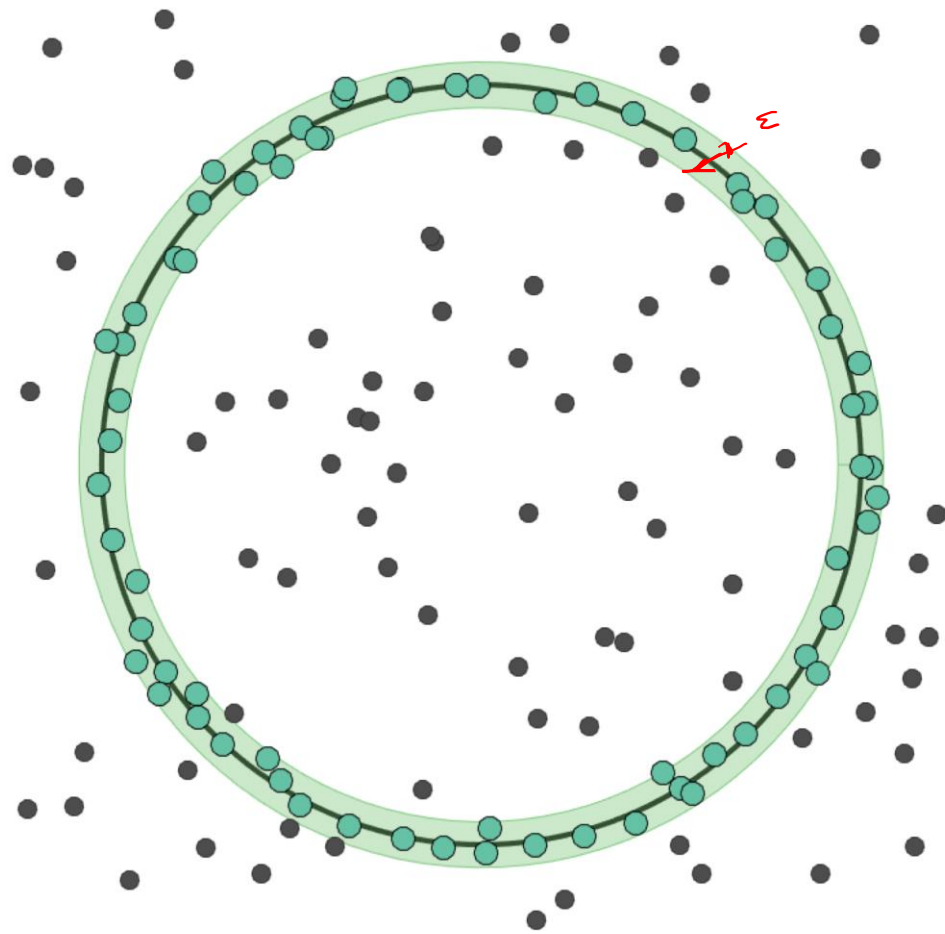
Image of the scene

colour coded class

colour coded model

# Multi-model Fitting Solutions

# Let's go back to RanSaC

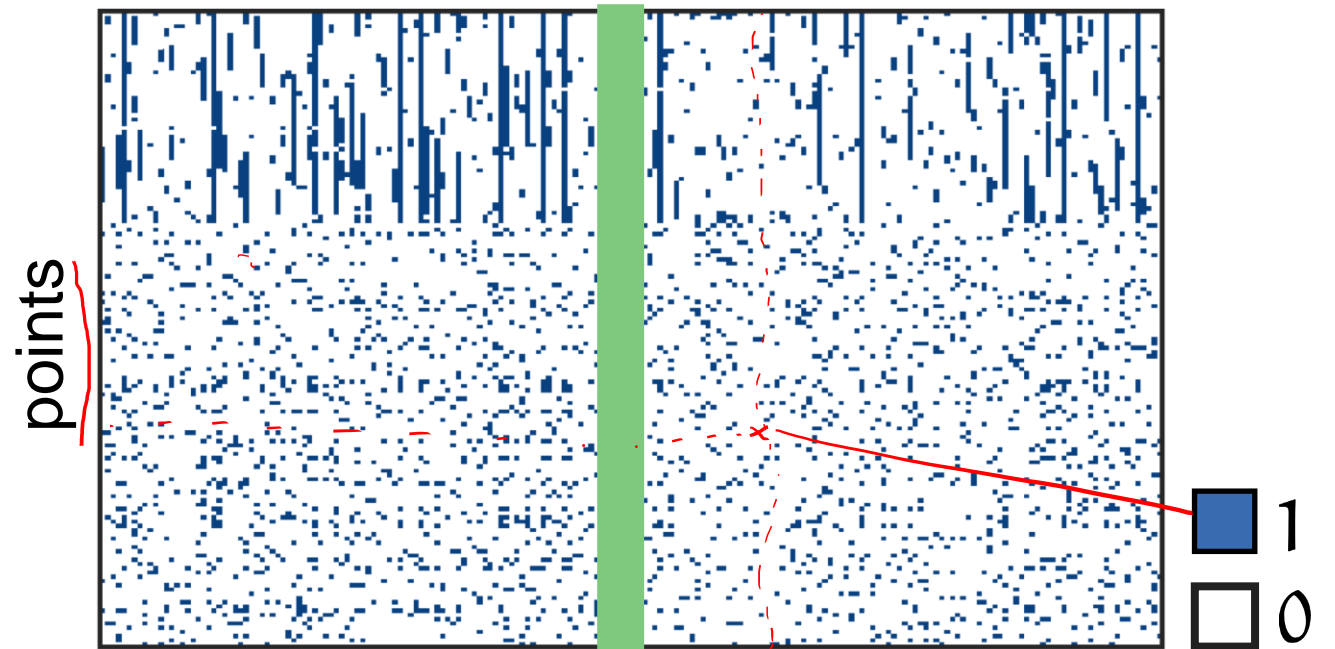


sum  $(P, \mathcal{T})$

Data driven search of model space

$$H = \{\theta_1, \theta_2, \dots, \theta_m\} \approx \Theta$$

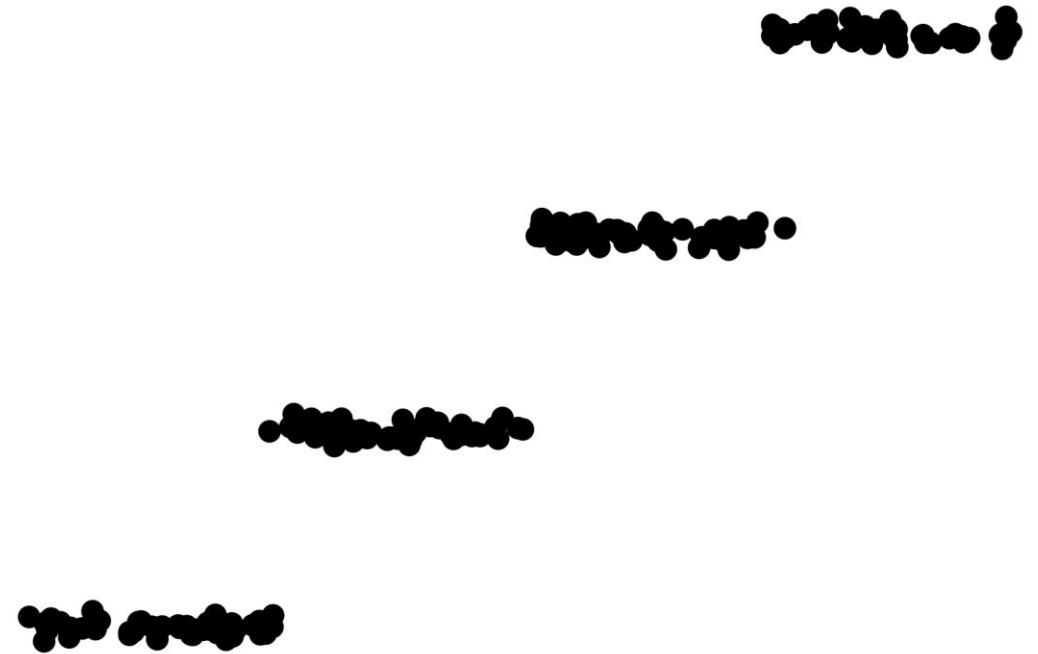
tentative models



pick the column with the maximum sum

# Sequential RanSaC [Zuliani 05]

Start RanSaC on the dataset  $X$  searching for the best fit for a single instance of the model



Line fitting example



# Sequential RanSaC [Zuliani 05]

Start RanSaC on the dataset  $X$  searching for the best fit for a single instance of the model

Once detected a model  $\theta_1$ , keep the model and remove all the inliers



Line fitting example

# Sequential RanSaC [Zuliani 05]

Start RanSaC on the dataset  $X$  searching for the best fit for a single instance of the model

Once detected a model  $\theta_1$ , keep the model and remove all the inliers from  $X$



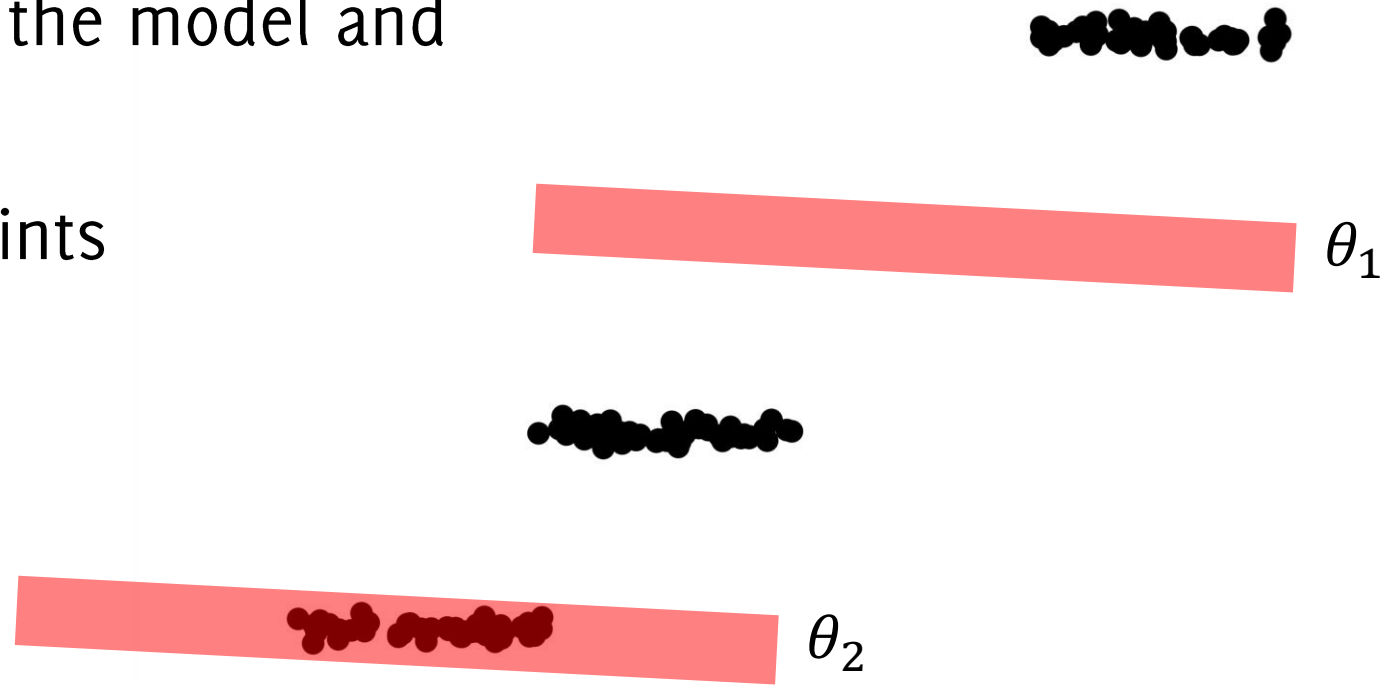
Line fitting example

# Sequential RanSaC [Zuliani 05]

Start RanSaC on the dataset  $X$  searching for the best fit for a single instance of the model

Once detected a model  $\theta_1$ , keep the model and remove all the inliers from  $X$

Iterate through the remaining points



Line fitting example

# Sequential RanSaC [Zuliani 05]

Start RanSaC on the dataset  $X$  searching for the best fit for a single instance of the model

Once detected a model  $\theta_1$ , keep the model and remove all the inliers from  $X$

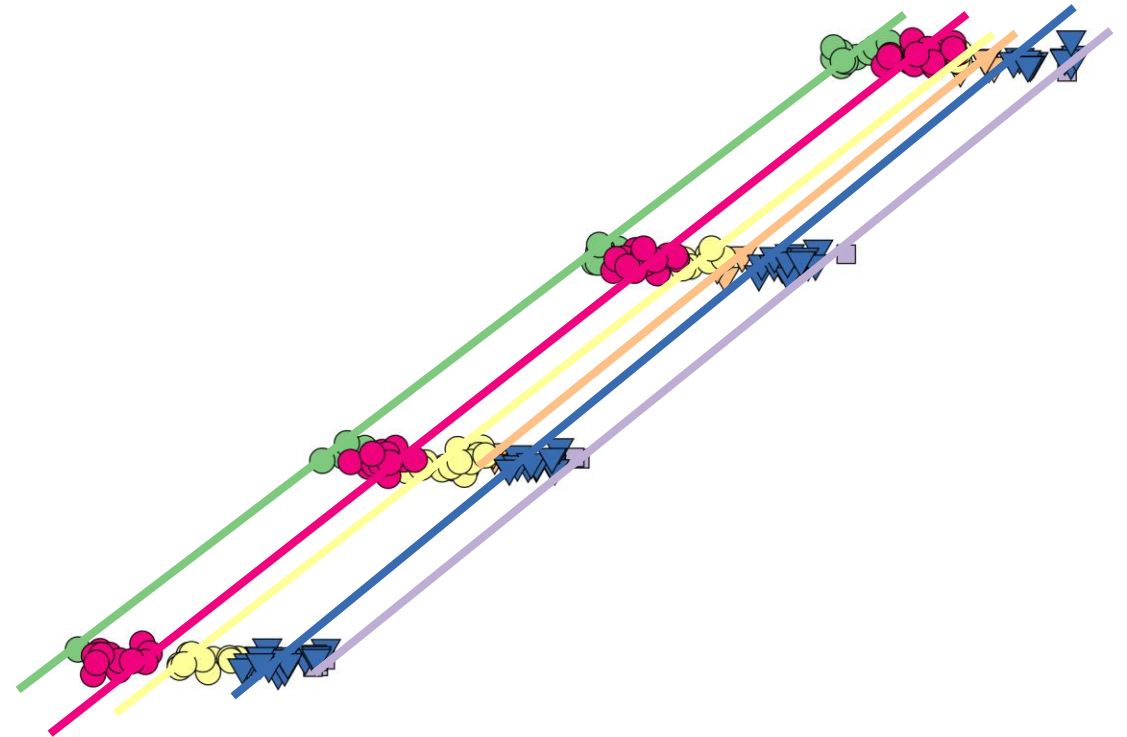
Iterate through the remaining points until there are no models with a sufficiently large consensus



Line fitting example

# Sequential RanSaC [Zuliani 05]

Unfortunately, this does not fit well with the multi-model scenario and the problem becomes even more severe in presence of outliers



Line fitting example

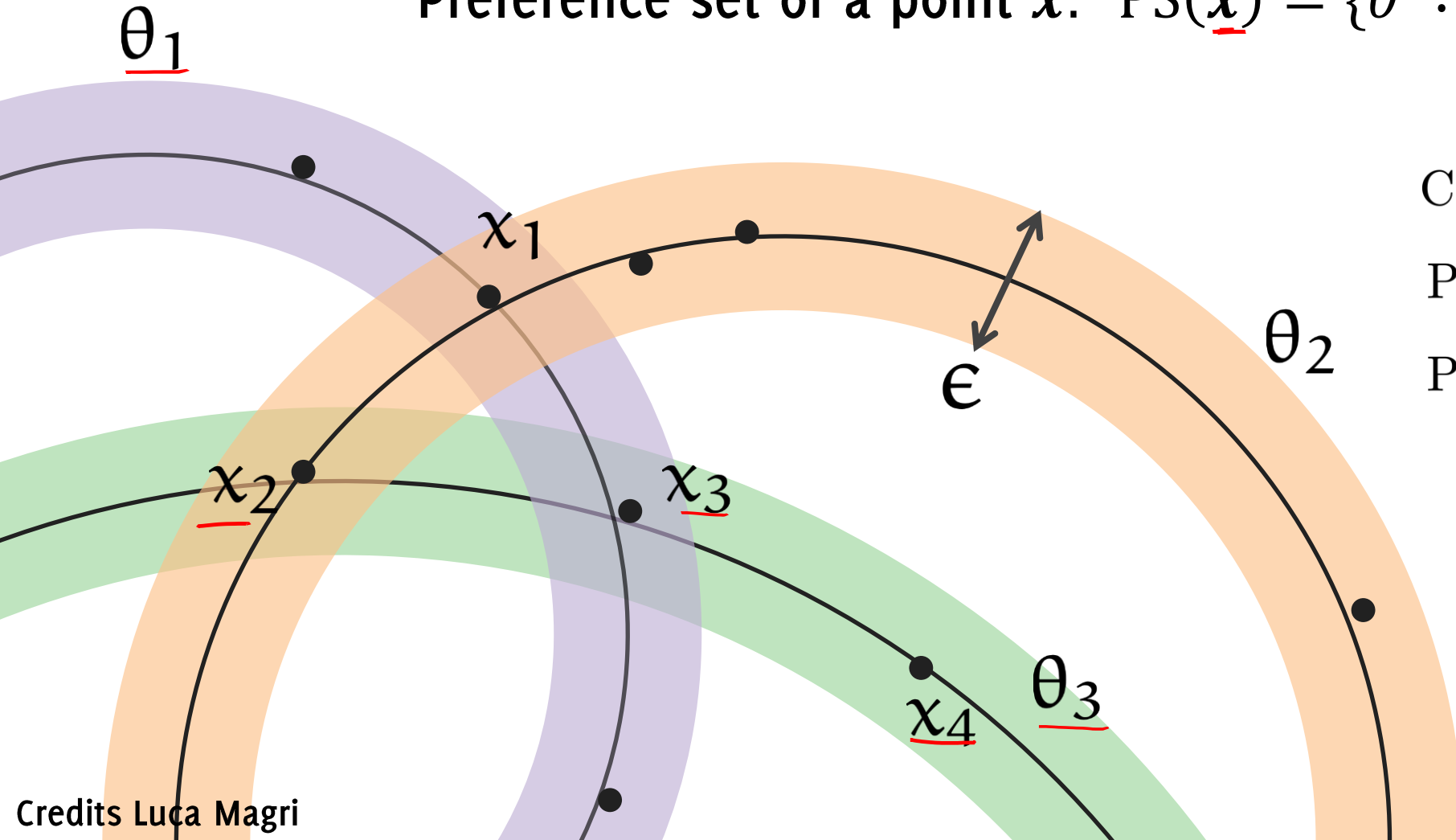
# Preference-based methods

# From model consensus to point preferences

*distance from the rest of*  
↓

Consensus set of a model  $\theta$ :  $CS(\theta) = \{x : r(x, \theta) < \epsilon\}$

Preference set of a point  $x$ :  $PS(x) = \{\theta : r(x, \theta) < \epsilon\}$

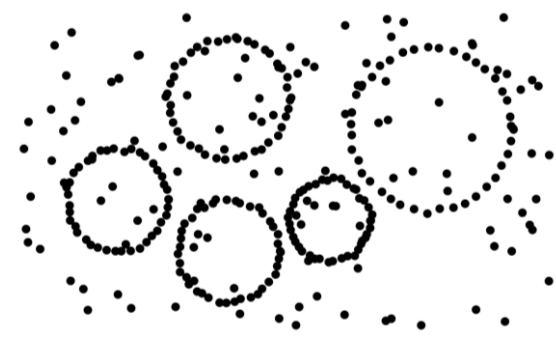


$$CS(\theta_3) = \{x_2, x_3, x_4\}$$

$$PS(x_1) = \{\theta_1, \theta_2\}$$

$$PS(x_2) = \{\theta_2, \theta_3\}$$

# Multi model fitting: the preference trick

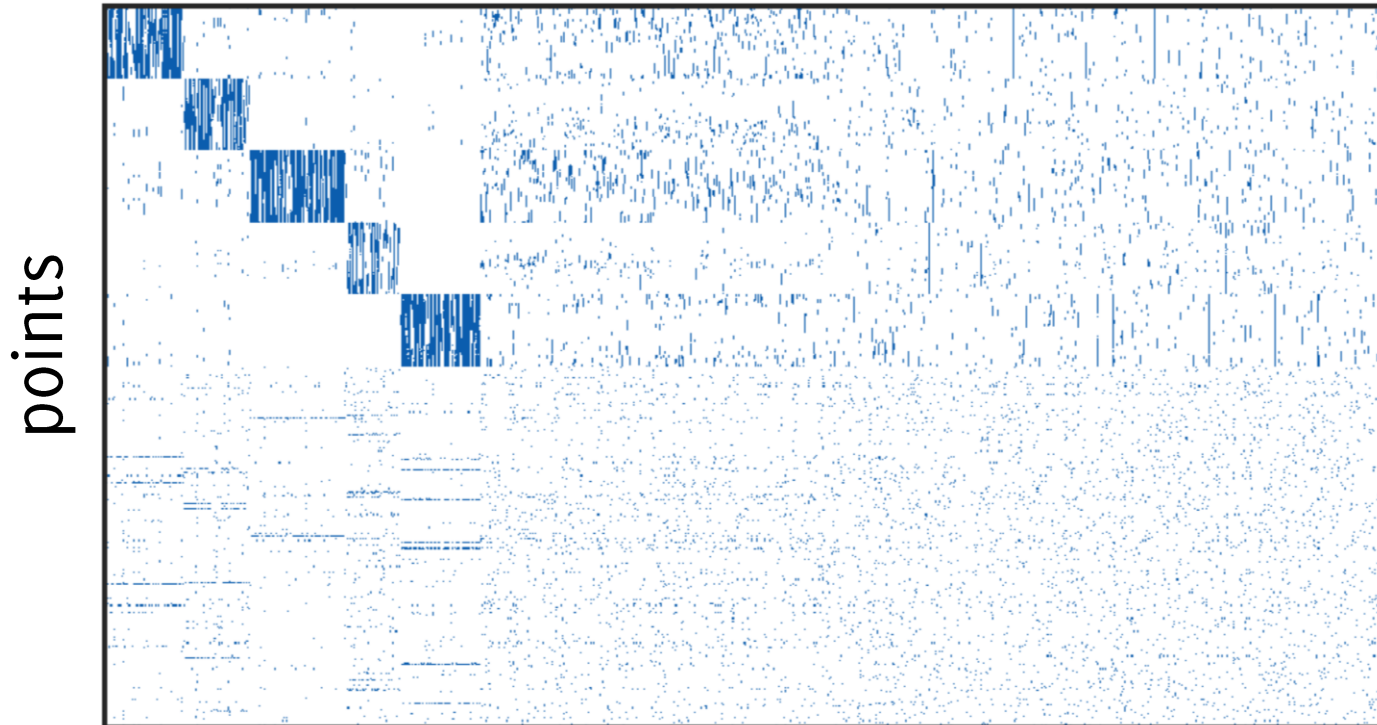


Generate a pool of  $m$  random models (as in RanSaC)

$$H = \{\theta_1, \theta_2, \dots, \theta_m\}$$

Build the Preference Matrix

tentative models



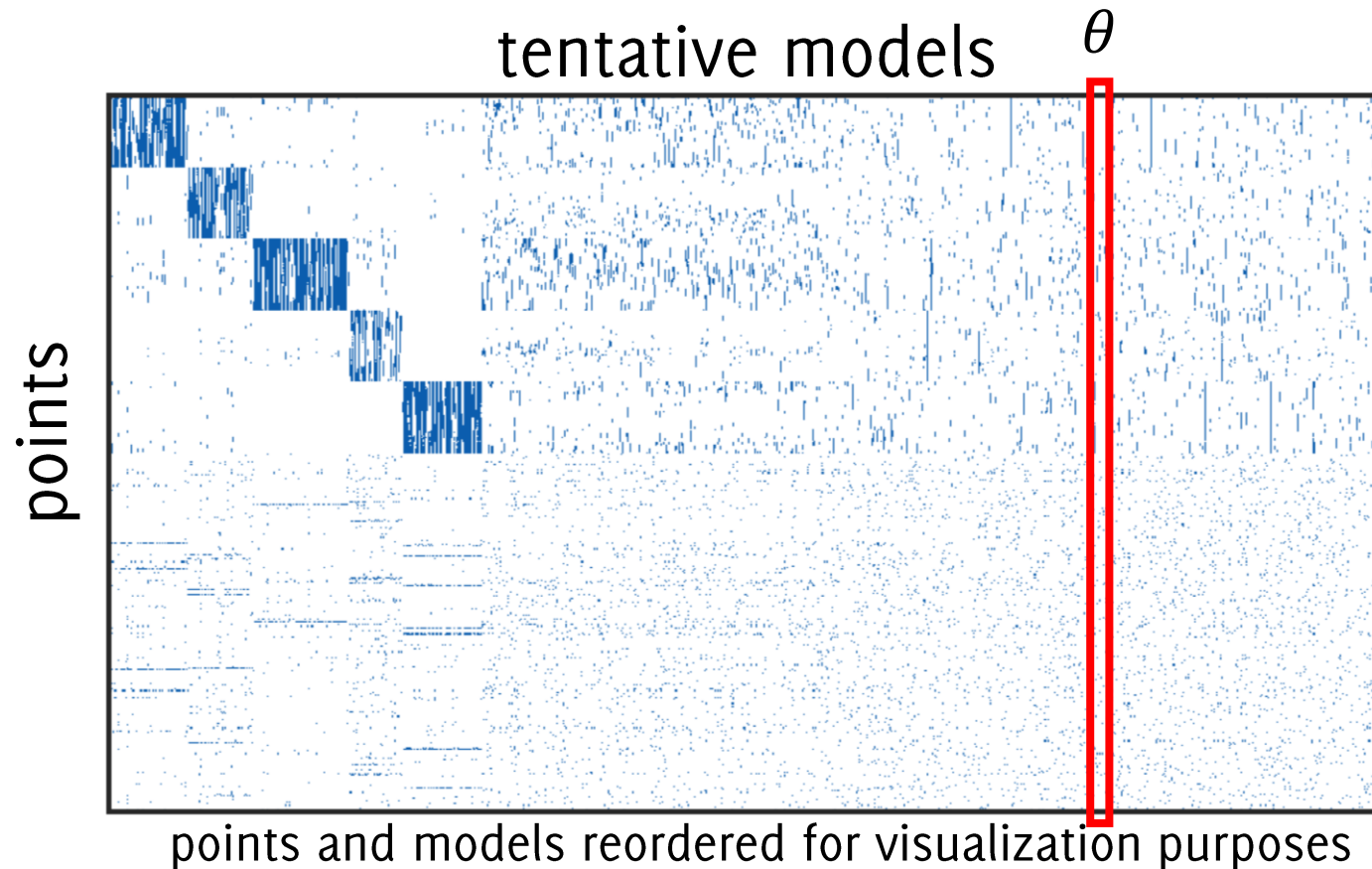
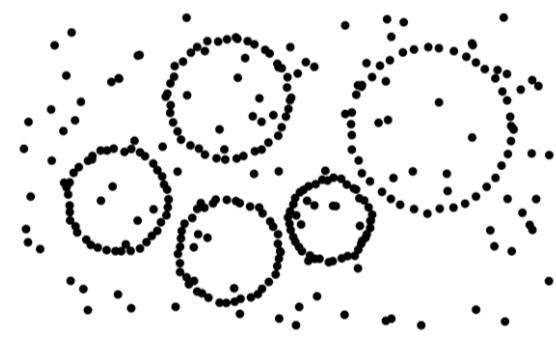
points and models reordered for visualization purposes



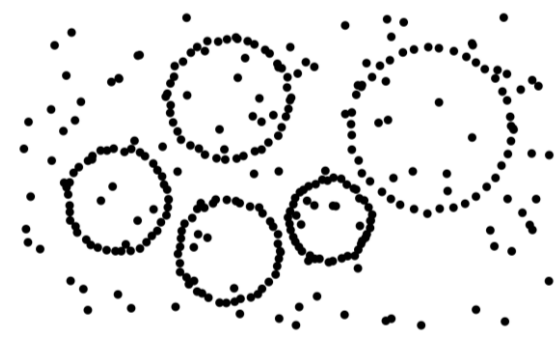
# Multi model fitting: the preference trick

Consensus set of a model  $\theta$ :  $CS(\theta) = \{\mathbf{x} : r(\mathbf{x}, \theta) < \epsilon\}$

Preference set of a point  $\mathbf{x}$ :  $PS(\mathbf{x}) = \{\theta : r(\mathbf{x}, \theta) < \epsilon\}$

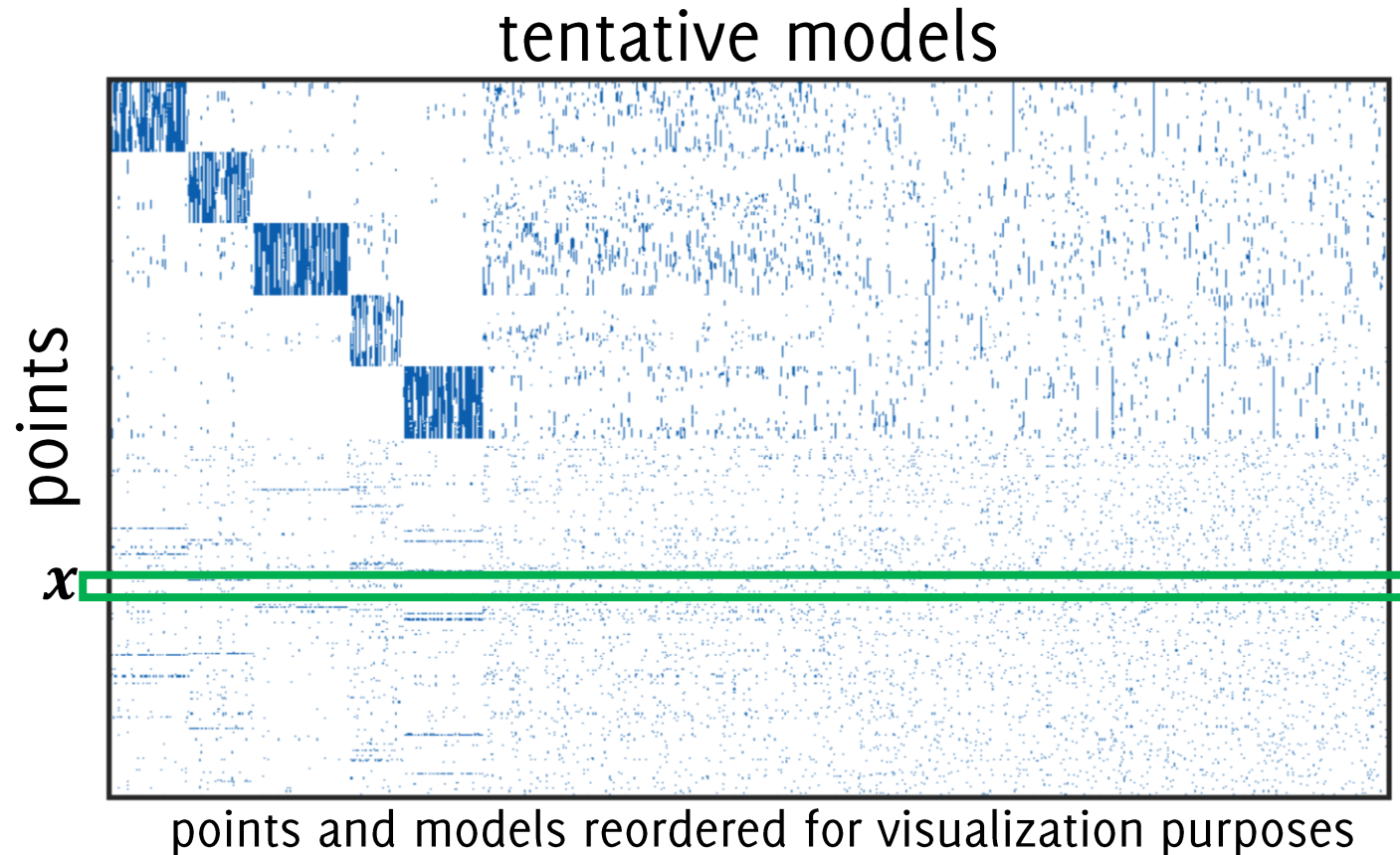


# Multi model fitting: the preference trick



Consensus set of a model  $\theta$ :  $CS(\theta) = \{\mathbf{x} : r(\mathbf{x}, \theta) < \epsilon\}$

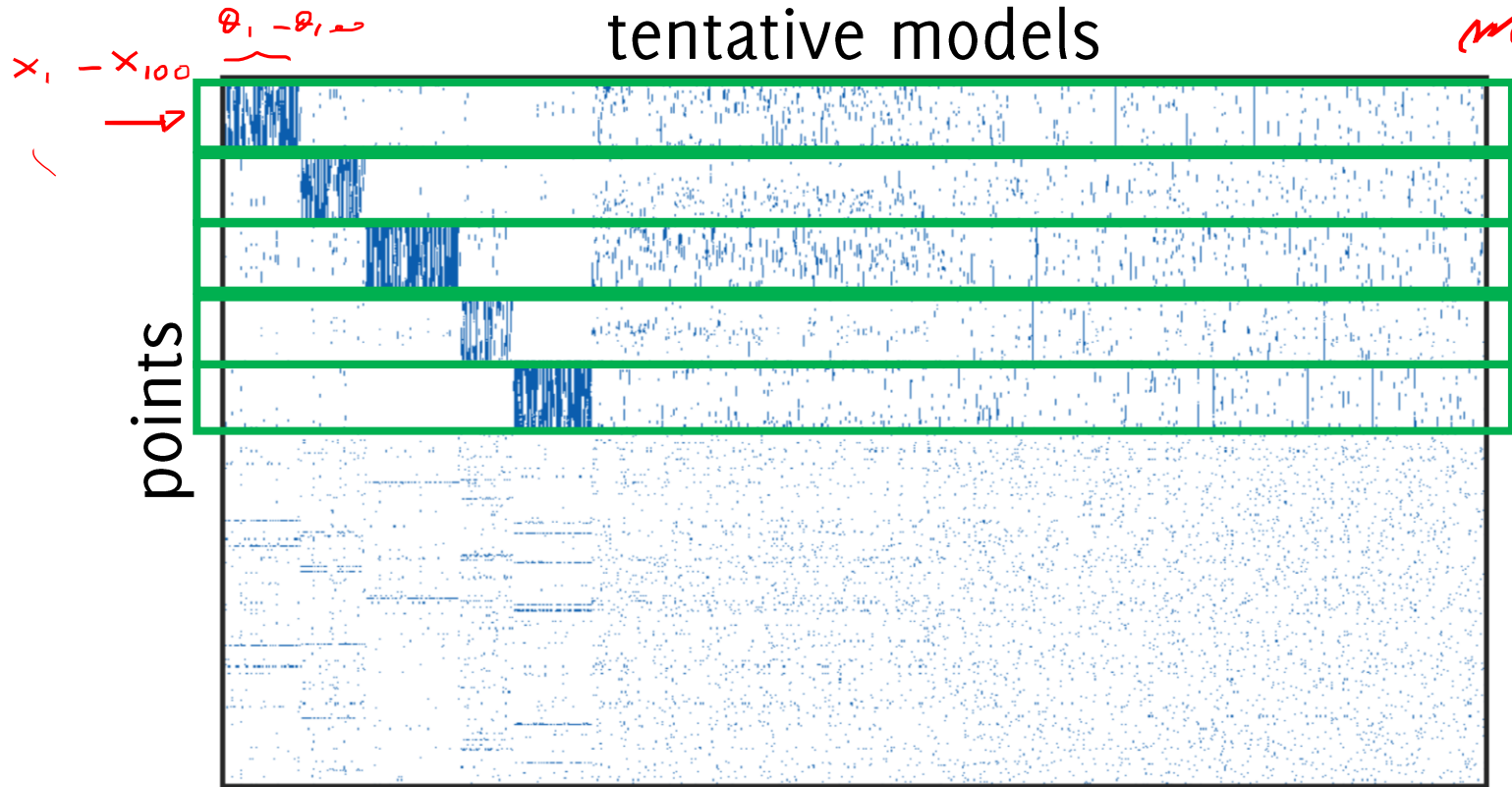
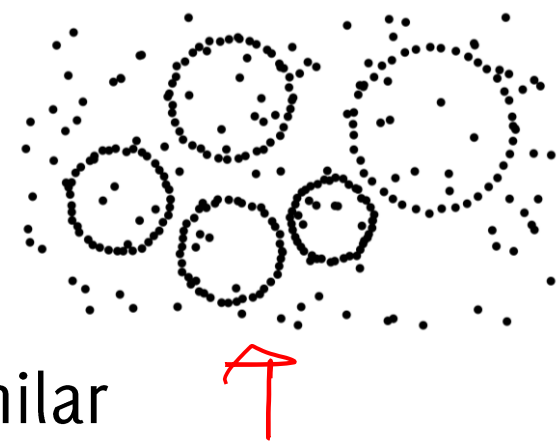
Preference set of a point  $\mathbf{x}$ :  $PS(\mathbf{x}) = \{\theta : r(\mathbf{x}, \theta) < \epsilon\}$



# Multi model fitting: the preference trick

Point  $\leftrightarrow$  subset of preferred sampled models

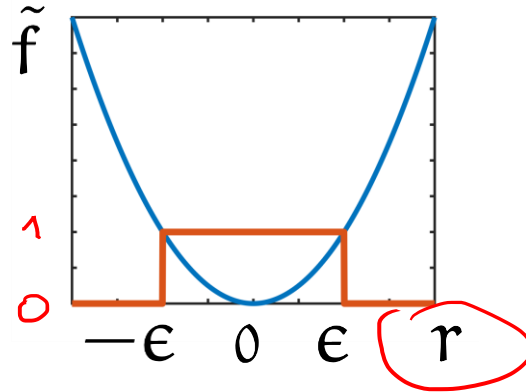
Block diagonal matrix  $\Rightarrow$  point of the same structure have similar preferences



points and models reordered for visualization purposes

# Multi model fitting: a lift to Preference Space

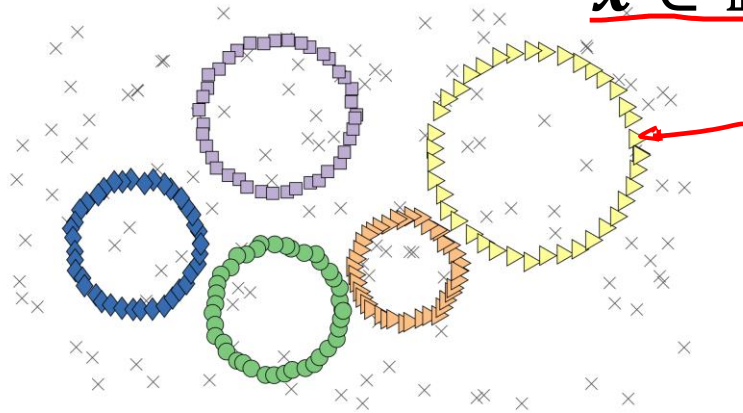
Voting function



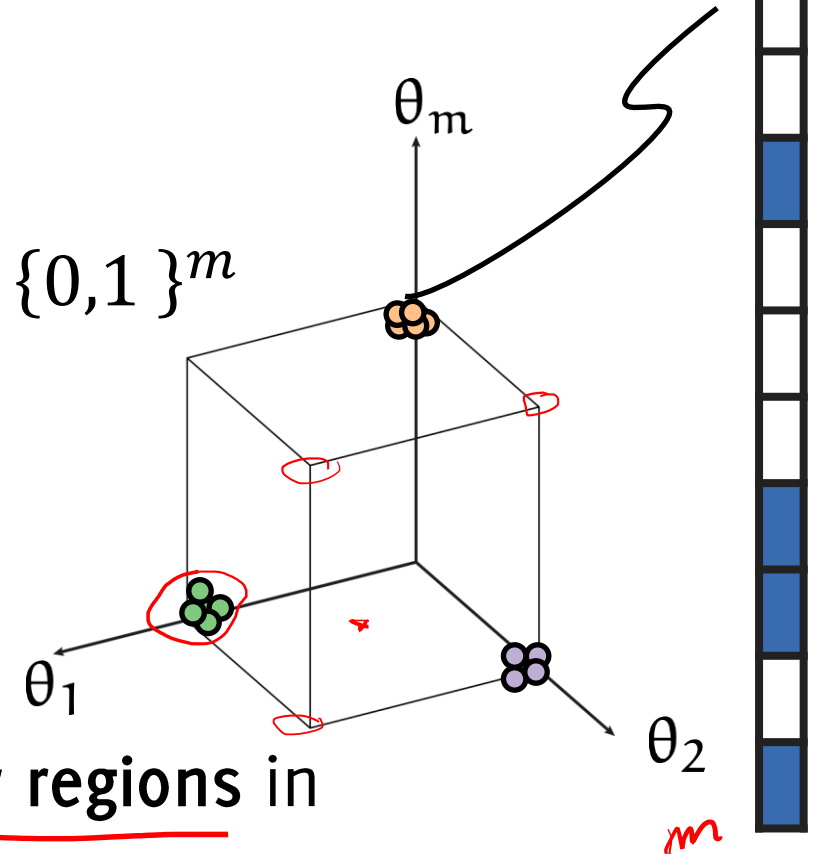
Vector of binary votes to sampled models

$$\mathbf{x} \mapsto [\hat{f}(r(\mathbf{x}, \theta_1)), \dots, \hat{f}(r(\mathbf{x}, \theta_m))] \in \{0, 1\}^m$$

$\mathbf{x} \in \mathbb{R}^d$



$\text{PS}(\mathbf{x}) \in \{0, 1\}^m$

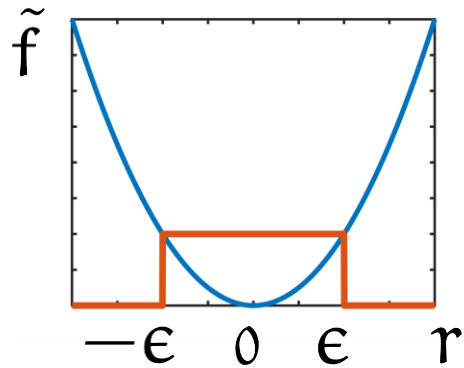


Structures correspond to high-density regions in the preference space

# Multi model fitting: a lift to Preference Space

PS( $\mathbf{x}$ )

Voting function



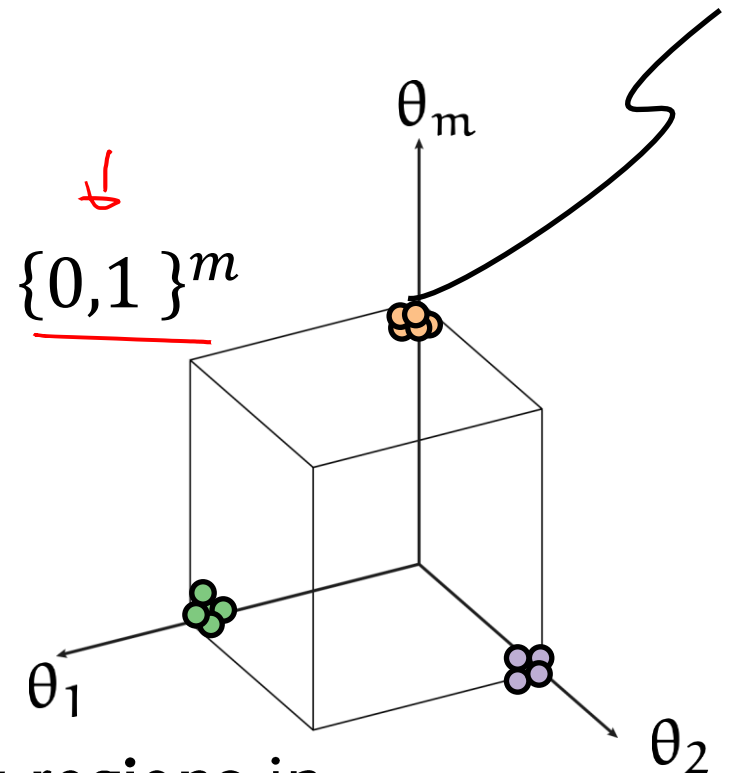
Vector of binary votes to sampled models

$$\mathbf{x} \mapsto [\hat{f}(r(\mathbf{x}, \theta_1)), \dots, \hat{f}(r(\mathbf{x}, \theta_m))] \in \{0, 1\}^m$$

$\mathbf{x} \in \mathbb{R}^d$

PS( $\mathbf{x}$ )  $\in \{0, 1\}^m$

Structures can be identified by clustering points in the preference space, i.e. by clustering preference sets.



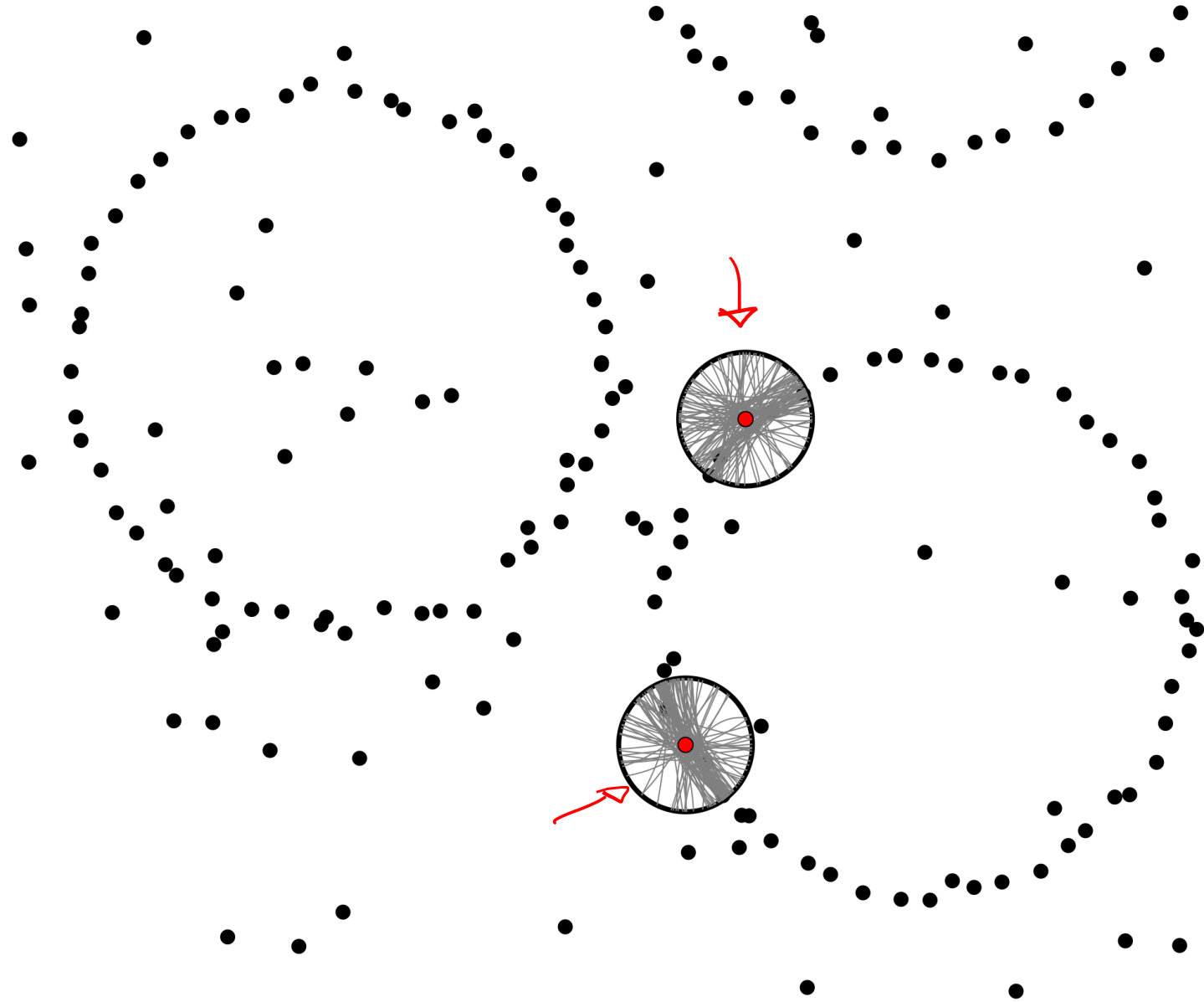
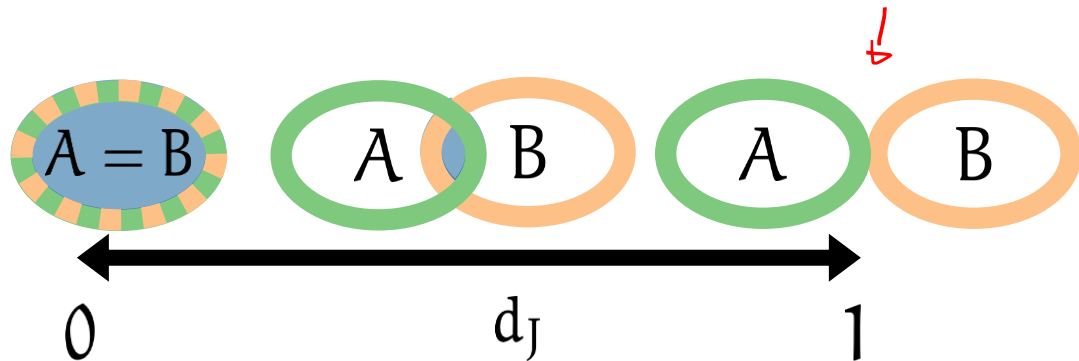
Structures correspond to high-density regions in the preference space

# Multi model fitting: a lift to Preference Space

The **Jaccard distance** can be used to measure distance between preference sets.

*0.14<sup>m</sup>*

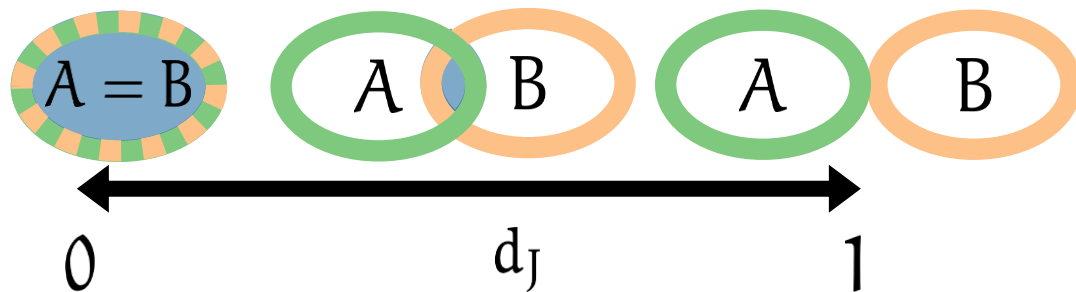
$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$



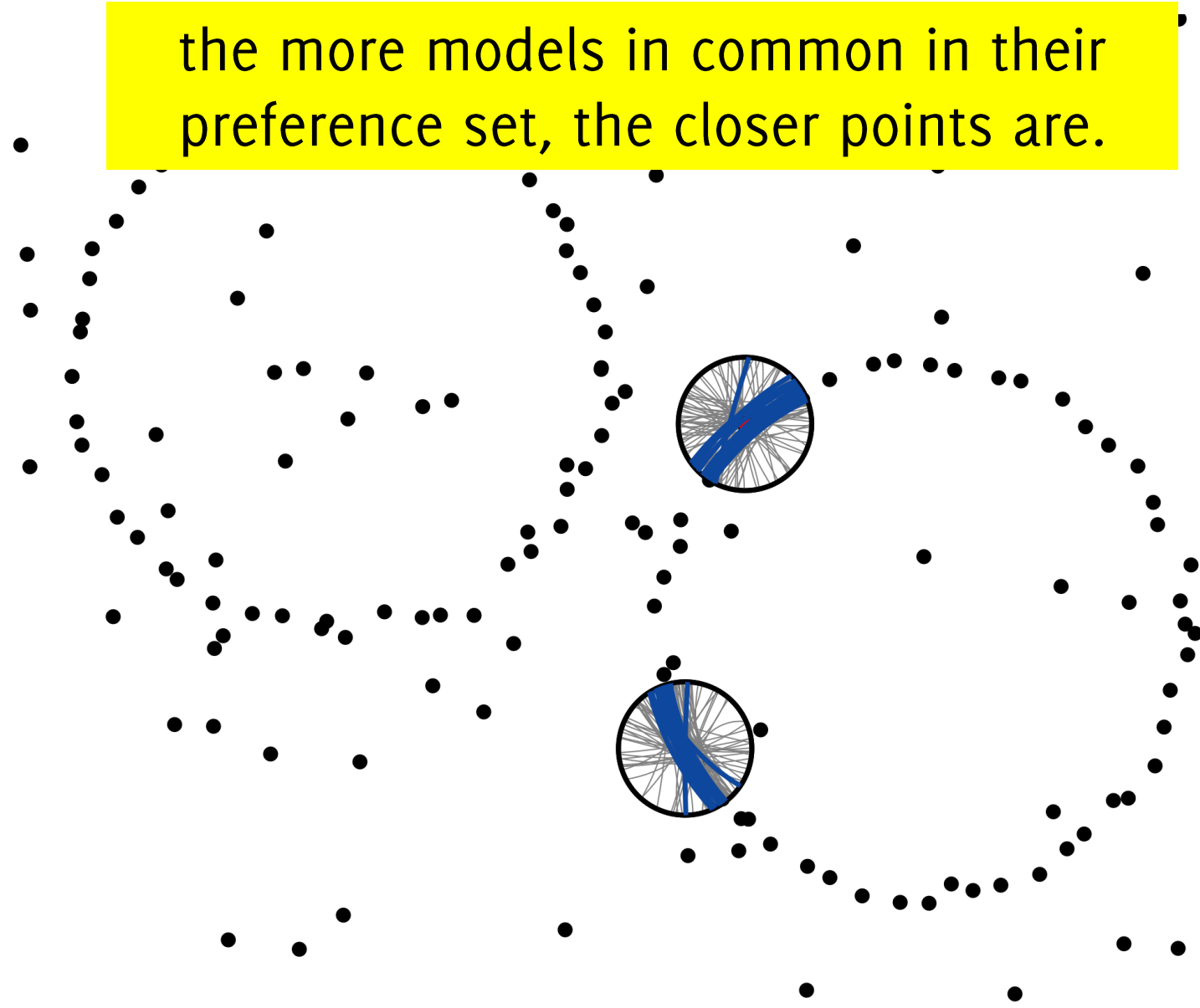
# Multi model fitting: a lift to Preference Space

The **Jaccard distance** can be used to measure distance between preference sets.

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$



the more models in common in their preference set, the closer points are.

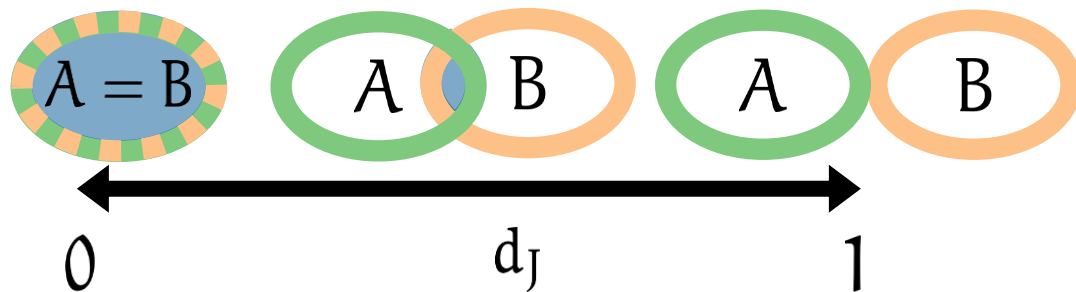




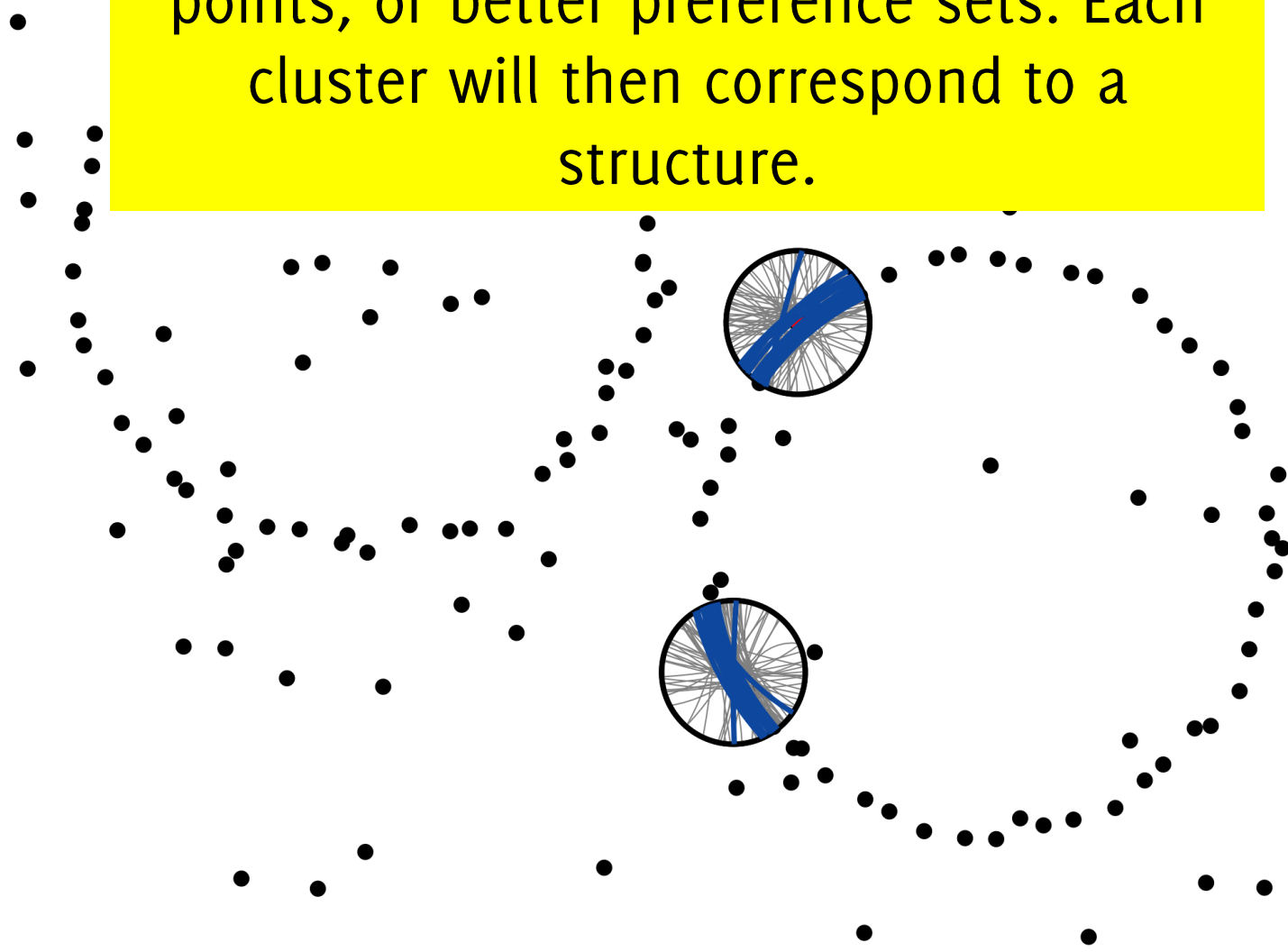
# Identify Structures by Clustering Preferences

The **Jaccard distance** can be used to measure distance between preference sets.

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$



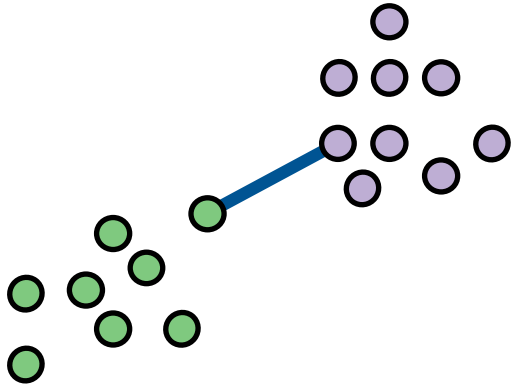
Structures are identified by clustering points, or better preference sets. Each cluster will then correspond to a structure.



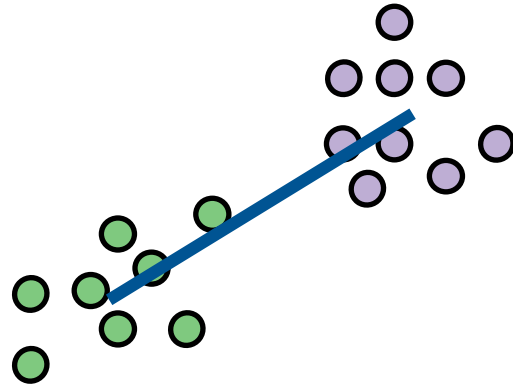


# Structure Identification by Clustering in PS

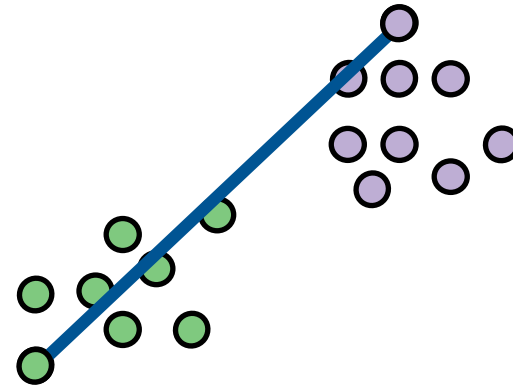
**Hierarchical clustering** can be used in the Preference Space to recover the structures.



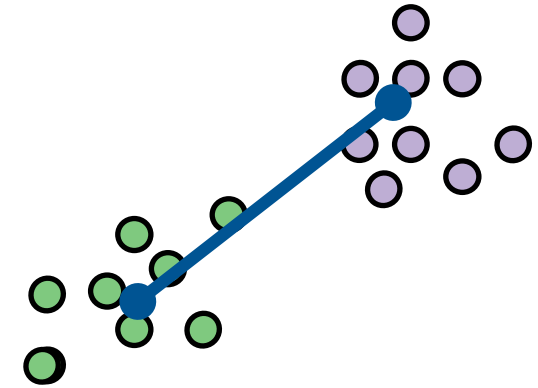
Single linkage



Average linkage



Complete linkage



Centroid linkage

Distances are measured in the Preference Space, and each element of clustering is identified by a preference set.

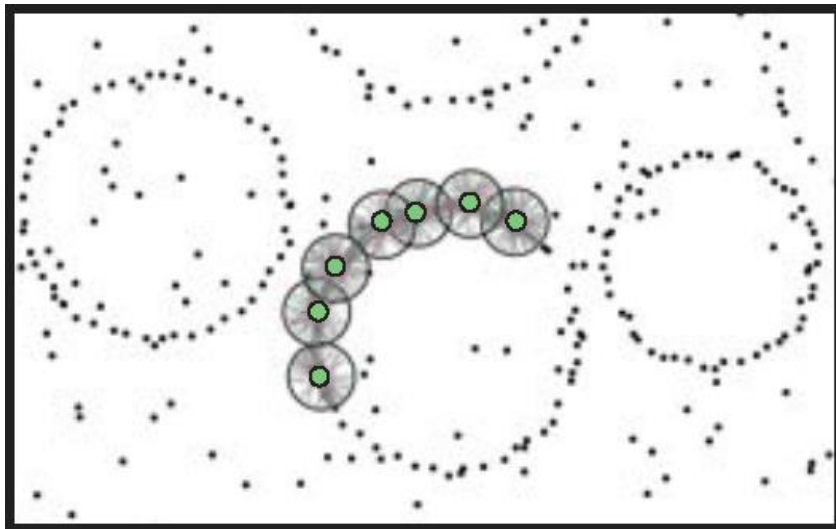
Instead of using centroids, we derive a conceptual representation for each cluster (which also lives in the preference space).

# J-linkage clustering [Toldo and Fusiello, ECCV 08]

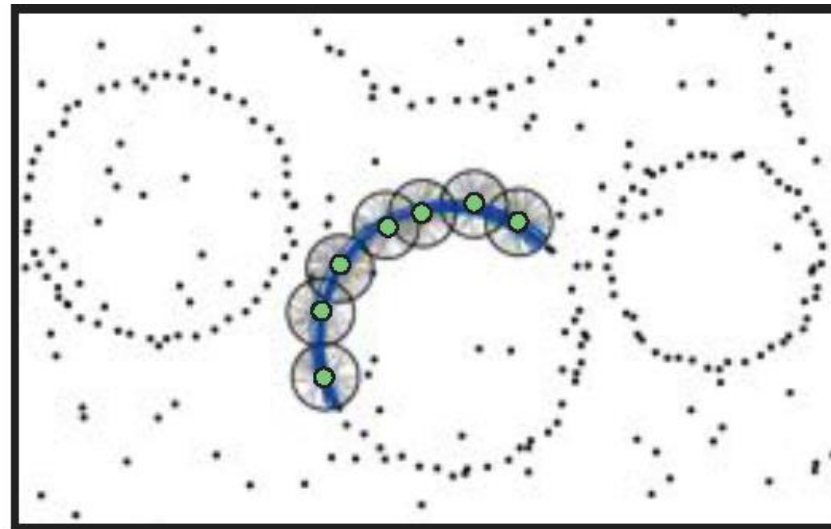
The representation of a cluster  $U \subset X$  in the preference space is the intersection of the PS of its points

$$\underline{U} \subseteq X, \text{PS}(\underline{U}) = \bigcap_{x \in U} \underline{\text{PS}}(x)$$

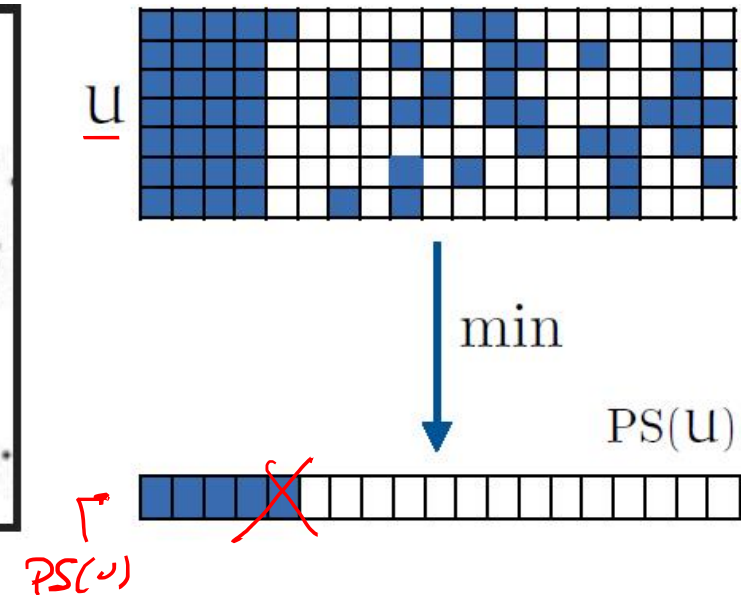
This is the component-wise min of rows in the preference matrix.



$\underline{U} \subseteq X$



$\text{PS}(\underline{U})$



# J-linkage clustering [Toldo and Fusiello, ECCV 08]

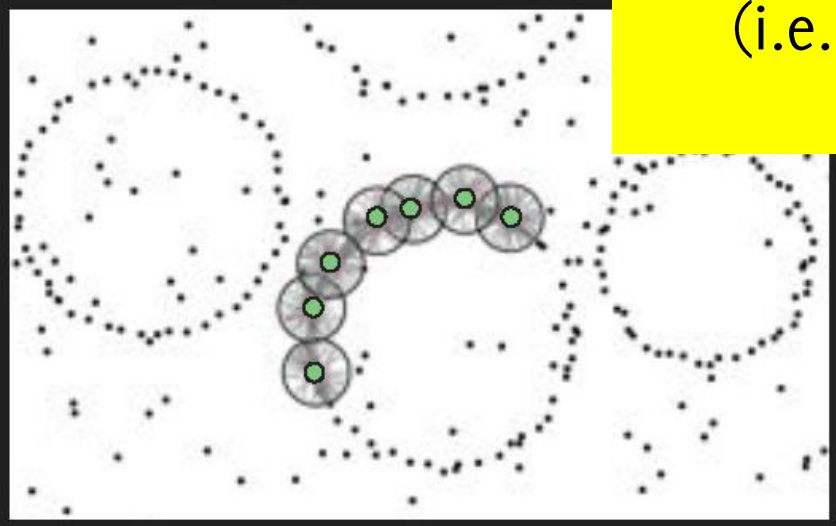
The representation of a cluster  $U \subset X$  in the preference space is the intersection of the PS of its points

$$U \subset X \implies \text{PS}(U) = \bigcap \text{PS}(x)$$

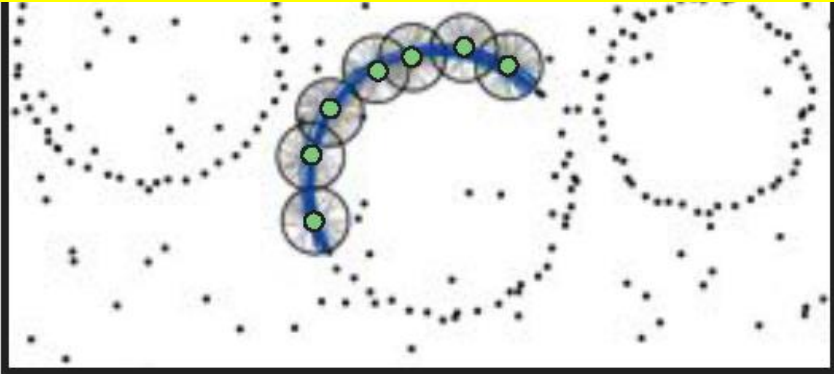
J-linkage iterates a hierarchical clustering scheme based on this distance until all the representatives of the clusters are disjoint (i.e. Jaccard distance = 1, no models in common)

This is the component

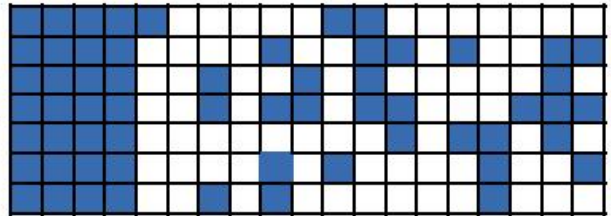
matrix.



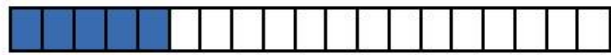
$U \subset X$



$\text{PS}(U)$

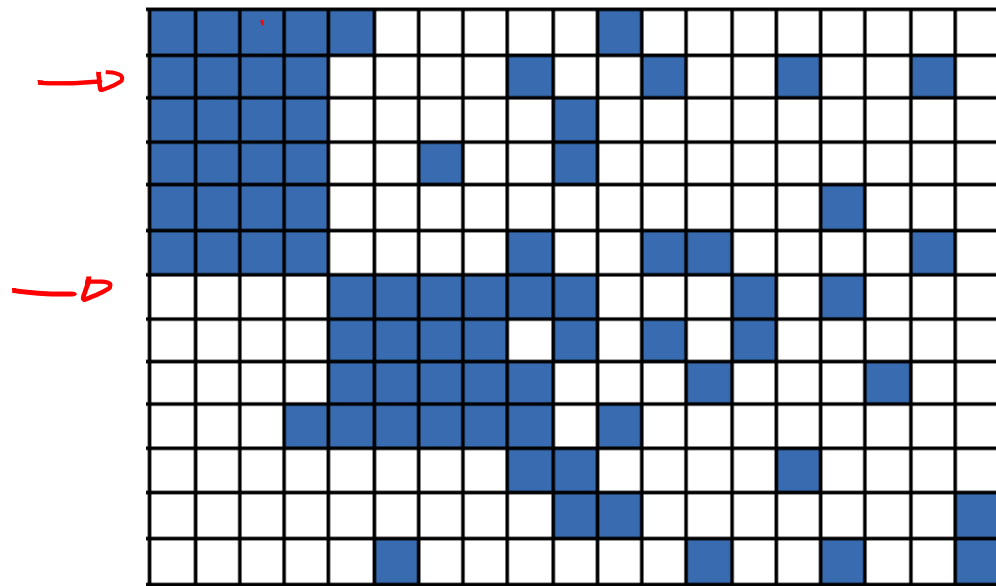


min  
↓



$\text{PS}(U)$

# J-linkage clustering [Toldo and Fusiello, ECCV 08]



Preference matrix

**Input:**  $X$  data,  $\epsilon$  inlier threshold

**Output:** Partition in structures and models

Randomly sample model hypotheses  $H \subset \Theta$ ;

Compute PS;

Put each point in its own cluster  $C_i = \{x_i\}$ ;

Compute  $d_J$  Jaccard distance between PS;

**while**  $\min(d_J) < 1$  **do**

    Find pair  $(C_i, C_j)$  of clusters with the min  $d_J$ ;

    Replace the clusters with their union;

    Compute the PS of  $C_i \cup C_j$ ;

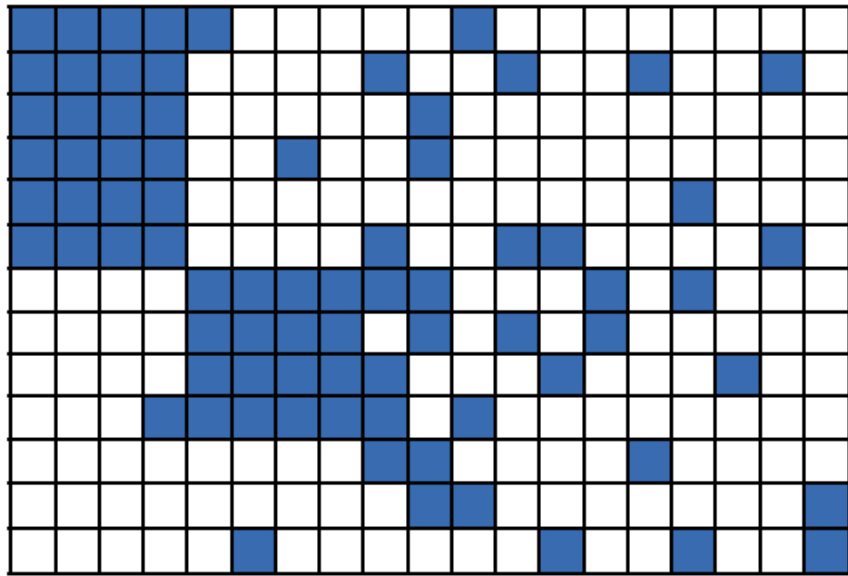
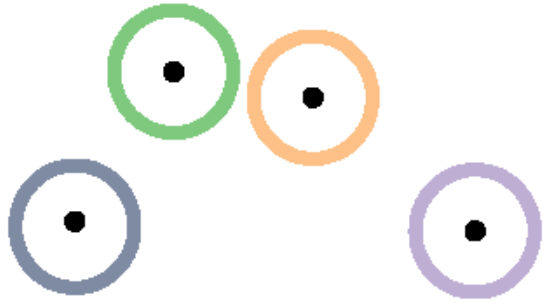
    Update  $d_J$ ;

**end**

Local fit of models to clusters;

# J-linkage clustering [Toldo and Fusiello, ECCV 08]

Put each point in its own cluster



Preference matrix

**Input:**  $X$  data,  $\epsilon$  inlier threshold

**Output:** Partition in structures and models

Randomly sample model hypotheses  $H \subset \Theta$ ;

Compute PS;

Put each point in its own cluster  $C_i = \{x_i\}$ ;

Compute  $d_J$  Jaccard distance between PS;

**while**  $\min(d_J) < 1$  **do**

Find pair  $(C_i, C_j)$  of clusters with the min  $d_J$ ;

Replace the clusters with their union;

Compute the PS of  $C_i \cup C_j$ ;

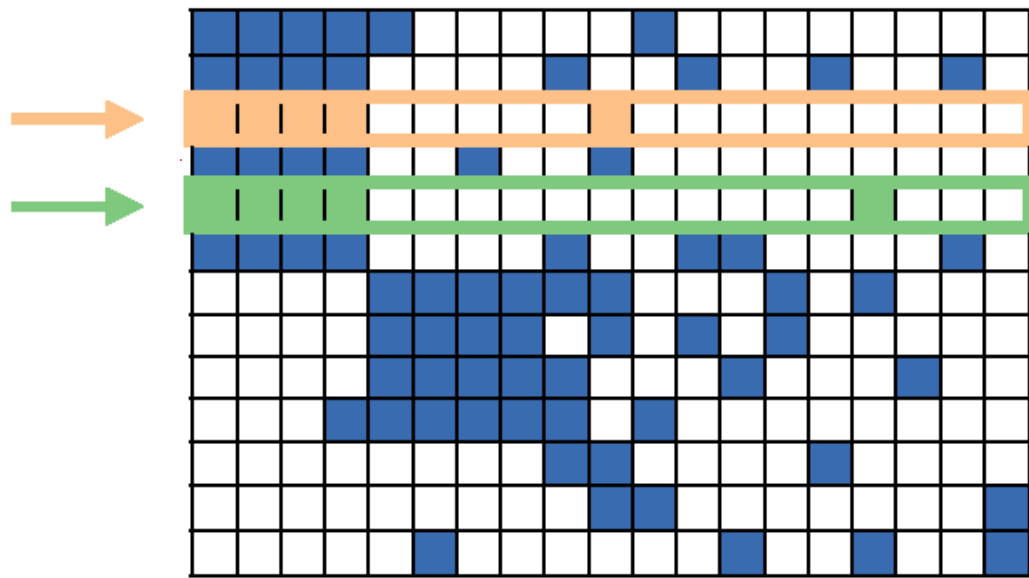
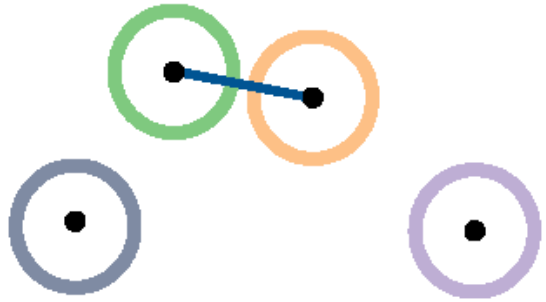
Update  $d_J$ ;

**end**

Local fit of models to clusters;

# J-linkage clustering [Toldo and Fusiello, ECCV 08]

Find the closest points in Preference Space



Preference matrix

**Input:**  $X$  data,  $\epsilon$  inlier threshold

**Output:** Partition in structures and models

Randomly sample model hypotheses  $H \subset \Theta$ ;

Compute PS;

Put each point in its own cluster  $C_i = \{x_i\}$ ;

Compute  $d_J$  Jaccard distance between PS;

**while**  $\min(d_J) < 1$  **do**

Find pair  $(C_i, C_j)$  of clusters with the min  $d_J$

Replace the clusters with their union;

Compute the PS of  $C_i \cup C_j$ ;

Update  $d_J$ ;

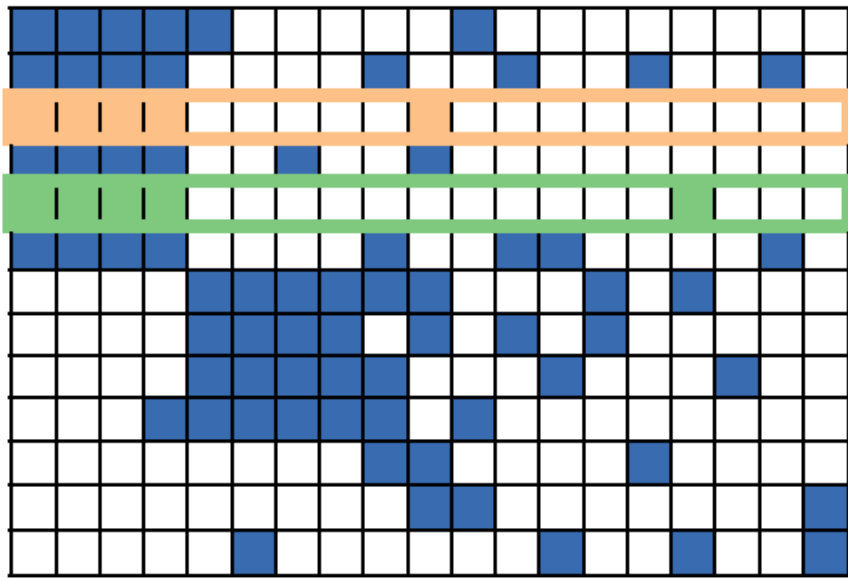
**end**

Local fit of models to clusters;



# J-linkage clustering [Toldo and Fusiello, ECCV 08]

Merge cluster



Preference matrix

**Input:**  $X$  data,  $\epsilon$  inlier threshold

**Output:** Partition in structures and models

Randomly sample model hypotheses  $H \subset \Theta$ ;

Compute PS;

Put each point in its own cluster  $C_i = \{x_i\}$ ;

Compute  $d_J$  Jaccard distance between PS;

**while**  $\min(d_J) < 1$  **do**

Find pair  $(C_i, C_j)$  of clusters with the min  $d_J$ ;

Replace the clusters with their union;

Compute the PS of  $C_i \cup C_j$ ;

Update  $d_J$ ;

**end**

Local fit of models to clusters;

# I-linkage clustering [Toldo and Fusiello, ECCV 08]

Update preferences



**Input:**  $X$  data,  $\epsilon$  inlier threshold

**Output:** Partition in structures and models

Randomly sample model hypotheses  $H \subset \Theta$ ;

Compute PS;

Put each point in its own cluster  $C_i = \{x_i\}$ ;

Compute  $d_J$  Jaccard distance between PS;

**while**  $\min(d_J) < 1$  **do**

Find pair  $(C_i, C_j)$  of clusters with the min  $d_J$ ;

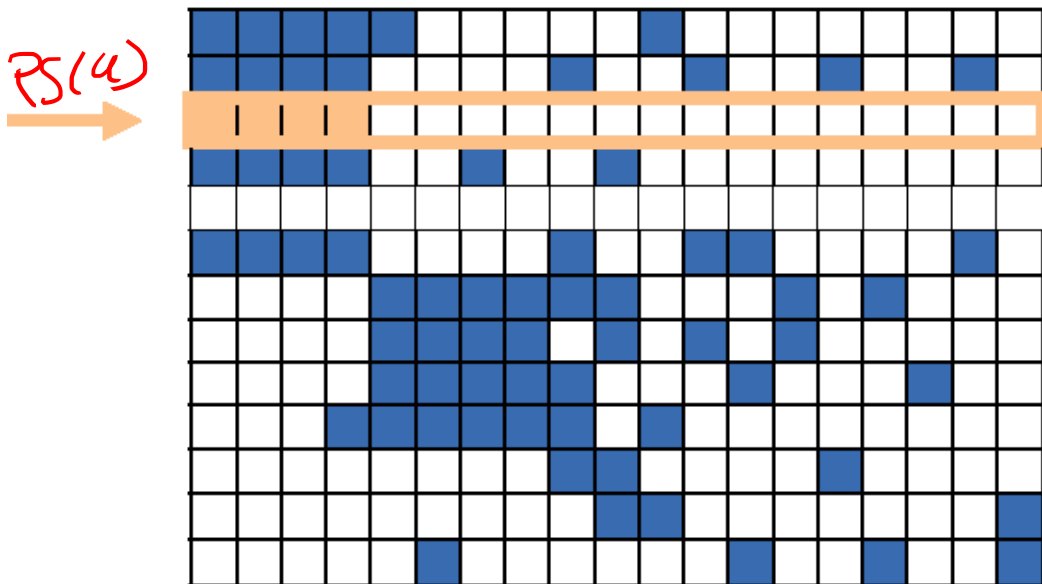
Replace the clusters with their union;

Compute the PS of  $C_i \cup C_j$ ;

Update  $d_J$ ;

**end**

Local fit of models to clusters;

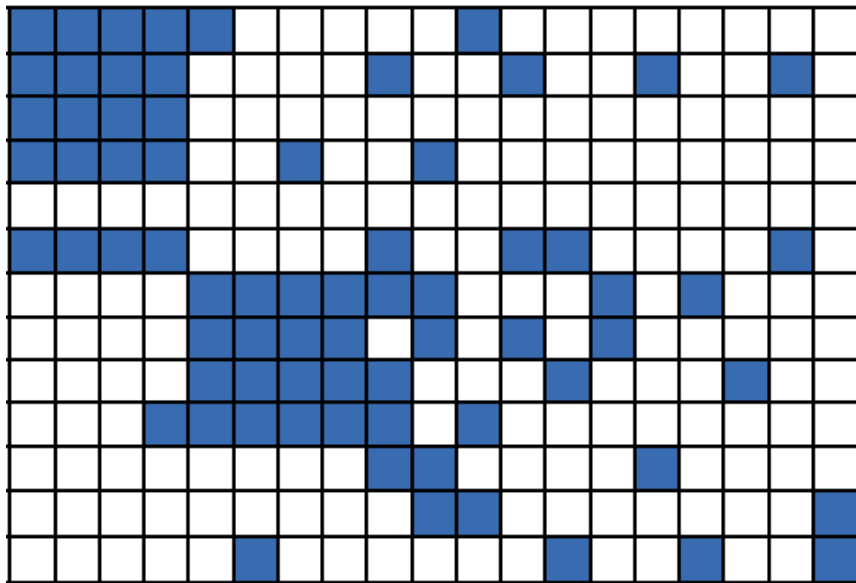
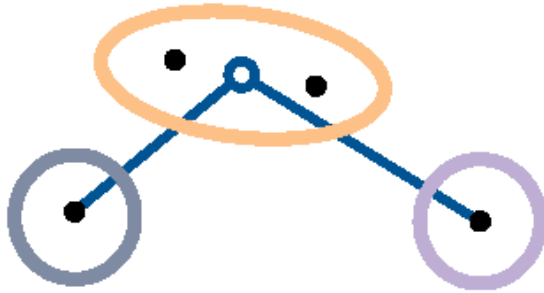


Preference matrix



# J-linkage clustering [Toldo and Fusiello, ECCV 08]

Update distances



Preference matrix

**Input:**  $X$  data,  $\epsilon$  inlier threshold

**Output:** Partition in structures and models

Randomly sample model hypotheses  $H \subset \Theta$ ;

Compute PS;

Put each point in its own cluster  $C_i = \{x_i\}$ ;

Compute  $d_J$  Jaccard distance between PS;

**while**  $\min(d_J) < 1$  **do**

Find pair  $(C_i, C_j)$  of clusters with the min  $d_J$ ;

Replace the clusters with their union;

Compute the PS of  $C_i \cup C_j$ ;

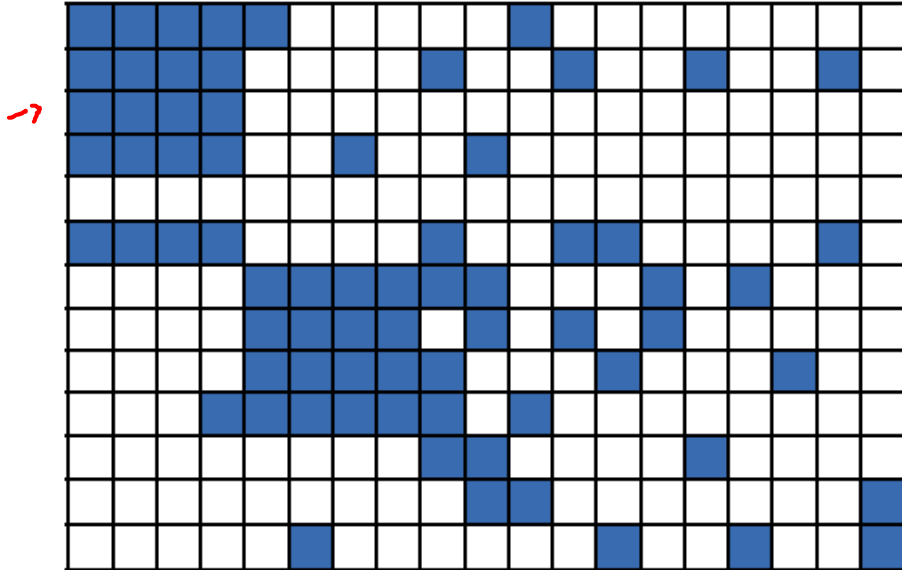
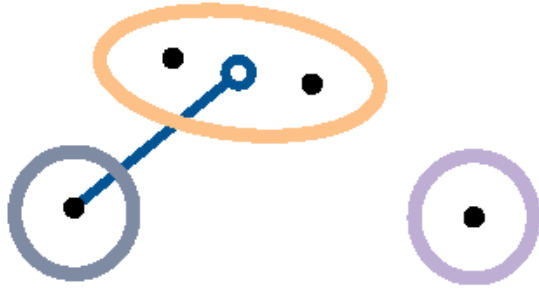
Update  $d_J$ ;

**end**

Local fit of models to clusters;

# J-linkage clustering [Toldo and Fusiello, ECCV 08]

Continue until all PS are disjoint...



Preference matrix

**Input:**  $X$  data,  $\epsilon$  inlier threshold

**Output:** Partition in structures and models

Randomly sample model hypotheses  $H \subset \Theta$ ;

Compute PS;

Put each point in its own cluster  $C_i = \{x_i\}$ ;

Compute  $d_J$  Jaccard distance between PS;

**while**  $\min(d_J) < 1$  **do**

Find pair  $(C_i, C_j)$  of clusters with the min  $d_J$ ;

Replace the clusters with their union;

Compute the PS of  $C_i \cup C_j$ ;

Update  $d_J$ ;

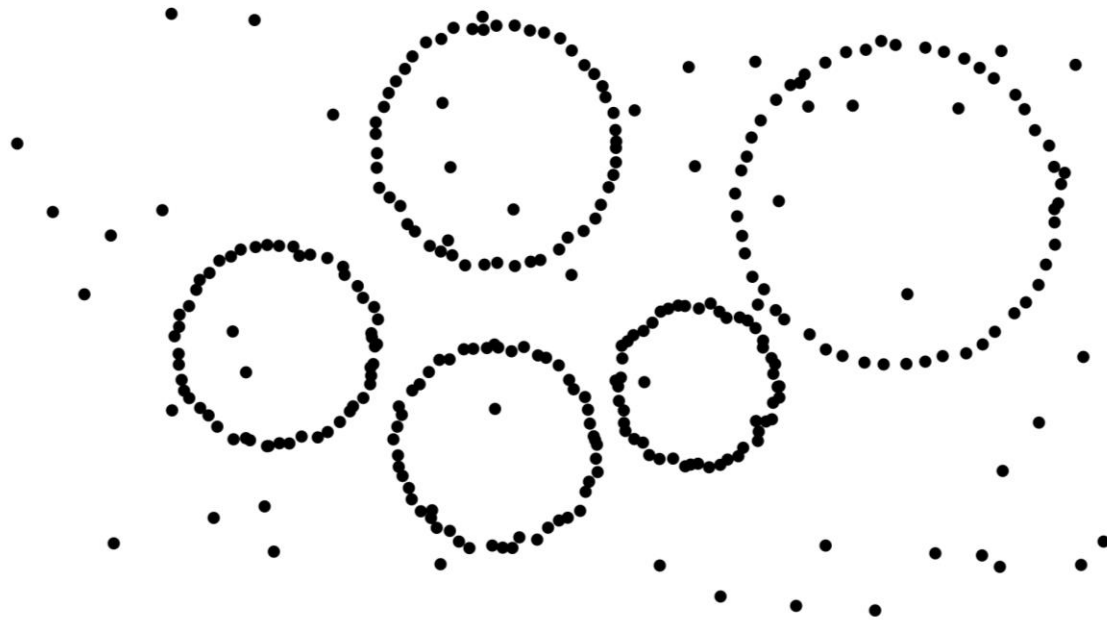
**end**

Local fit of models to clusters;

# J-linkage clustering [Toldo and Fusiello, ECCV 08]

**Input:**  $X$  data,  $\epsilon$  inlier threshold

**Output:** Partition in structures and models



• Randomly sample model hypotheses  $H \subset \Theta$ ;

• Compute PS;

• Put each point in its own cluster  $C_i = \{x_i\}$ ;

• Compute  $d_J$  Jaccard distance between PS;

**while**  $\min(d_J) < 1$  **do**

    Find pair  $(C_i, C_j)$  of clusters with the  $\min d_J$ ;

    Replace the clusters with their union;

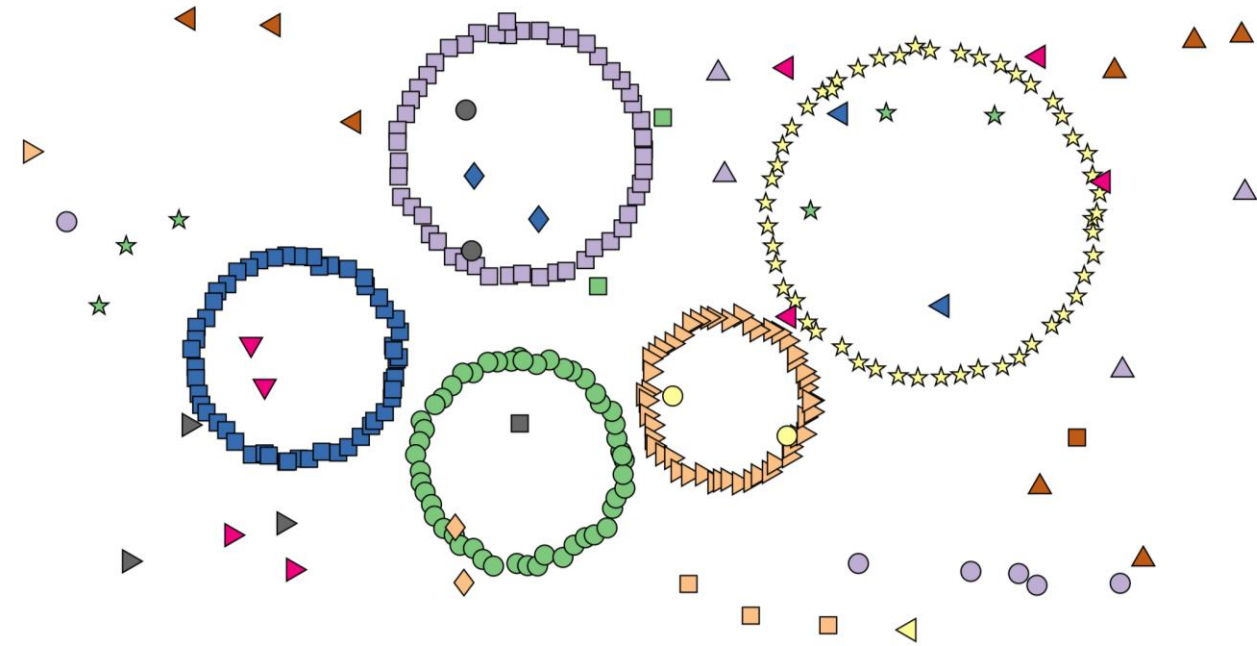
    Compute the PS of  $C_i \cup C_j$ ;

    Update  $d_J$ ;

**end**

Local fit of models to clusters;

# J-linkage clustering [Toldo and Fusiello, ECCV 08]



**Input:**  $X$  data,  $\epsilon$  inlier threshold

**Output:** Partition in structures and models

Randomly sample model hypotheses  $H \subset \Theta$ ;

Compute PS;

Put each point in its own cluster  $C_i = \{x_i\}$ ;

Compute  $d_J$  Jaccard distance between PS;

**while**  $\min(d_J) < 1$  **do**

Find pair  $(C_i, C_j)$  of clusters with the  $\min d_J$ ;

Replace the clusters with their union;

Compute the PS of  $C_i \cup C_j$ ;

Update  $d_J$ ;

**end**

Local fit of models to clusters;

# J-linkage clustering [Toldo and Fusiello, ECCV 08]

## Pro:

- The number of structures is automatically determined
- For each **cluster there exists at least one model** that fits all the points of the cluster
- **Clusters are “maximal”** in the sense that does not exist a model that explain all the points of two distinct clusters

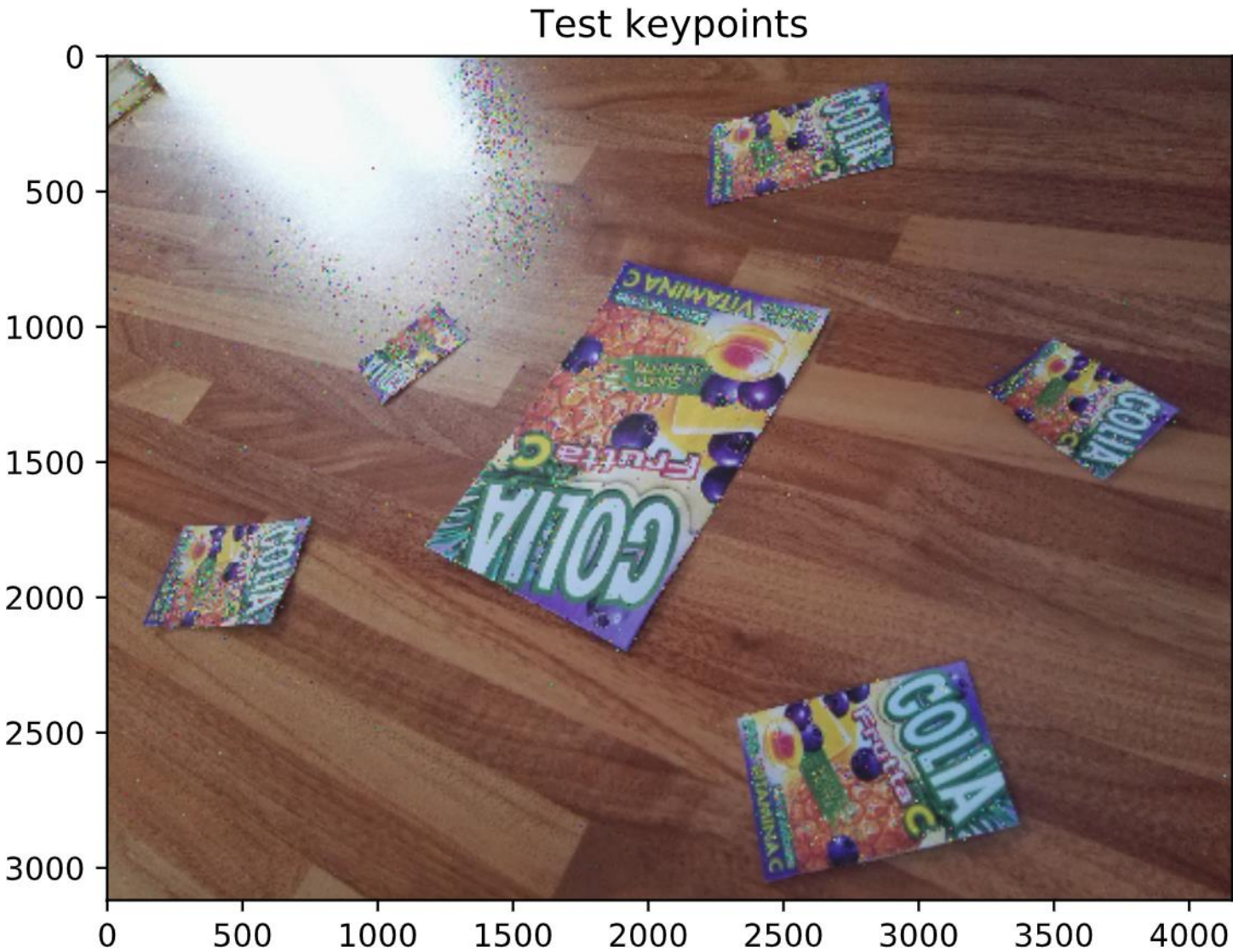
## Cons:

- It still relies on a pre-defined inlier threshold  $\epsilon$
- **Outliers have to be filtered out a posteriori** (they will always end up in small clusters)

# Object Detection by Computer Vision Features



# Object Detection: Keypoint Extraction



# Object Detection: Keypoint Extraction



Image Courtesy of Filippo Leveni



# Keypoint Matching

All matches

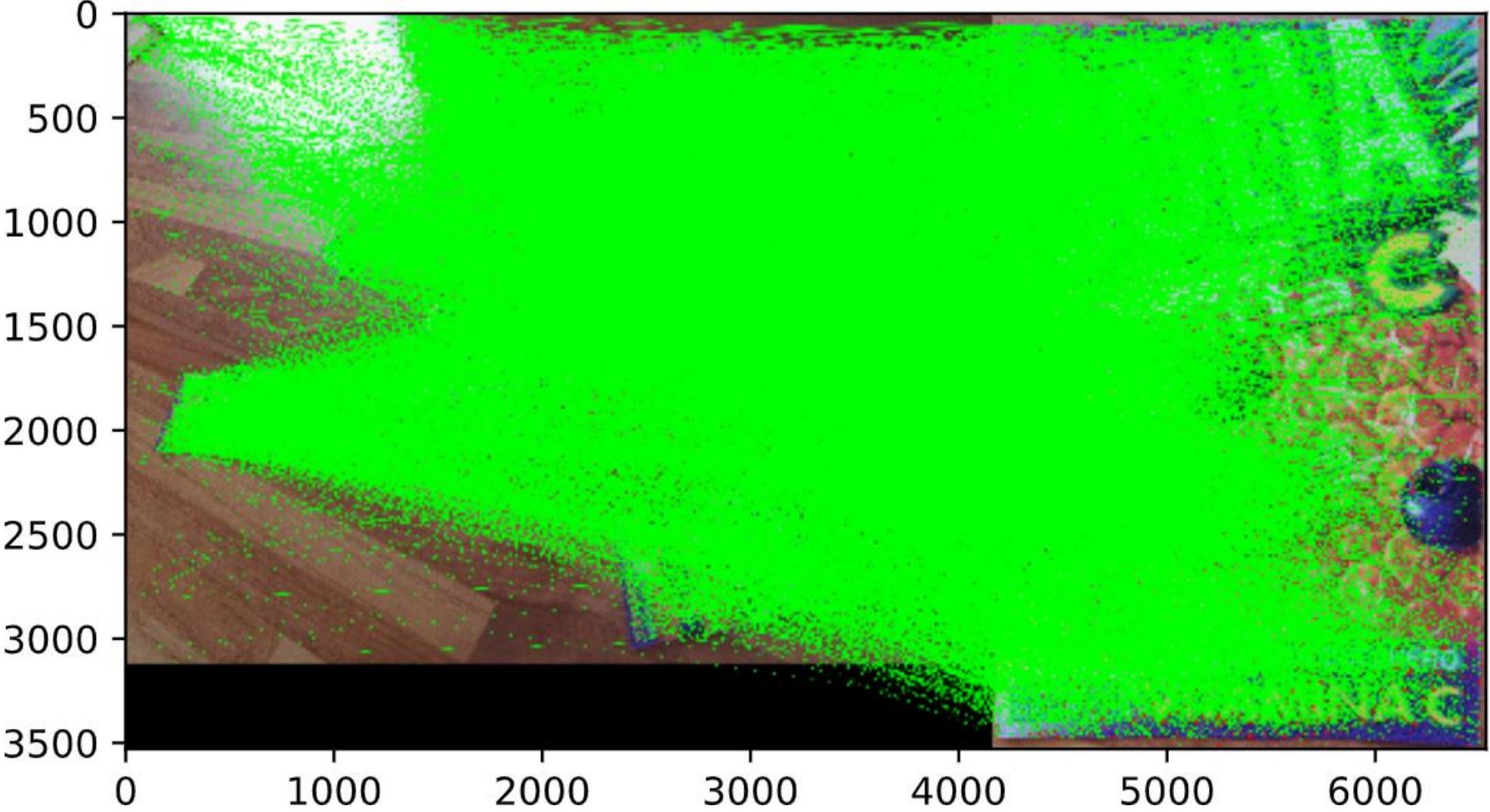


Image Courtesy of Filippo Leveni



# Keypoint Matching: Ratio Test

All matches after ratio test

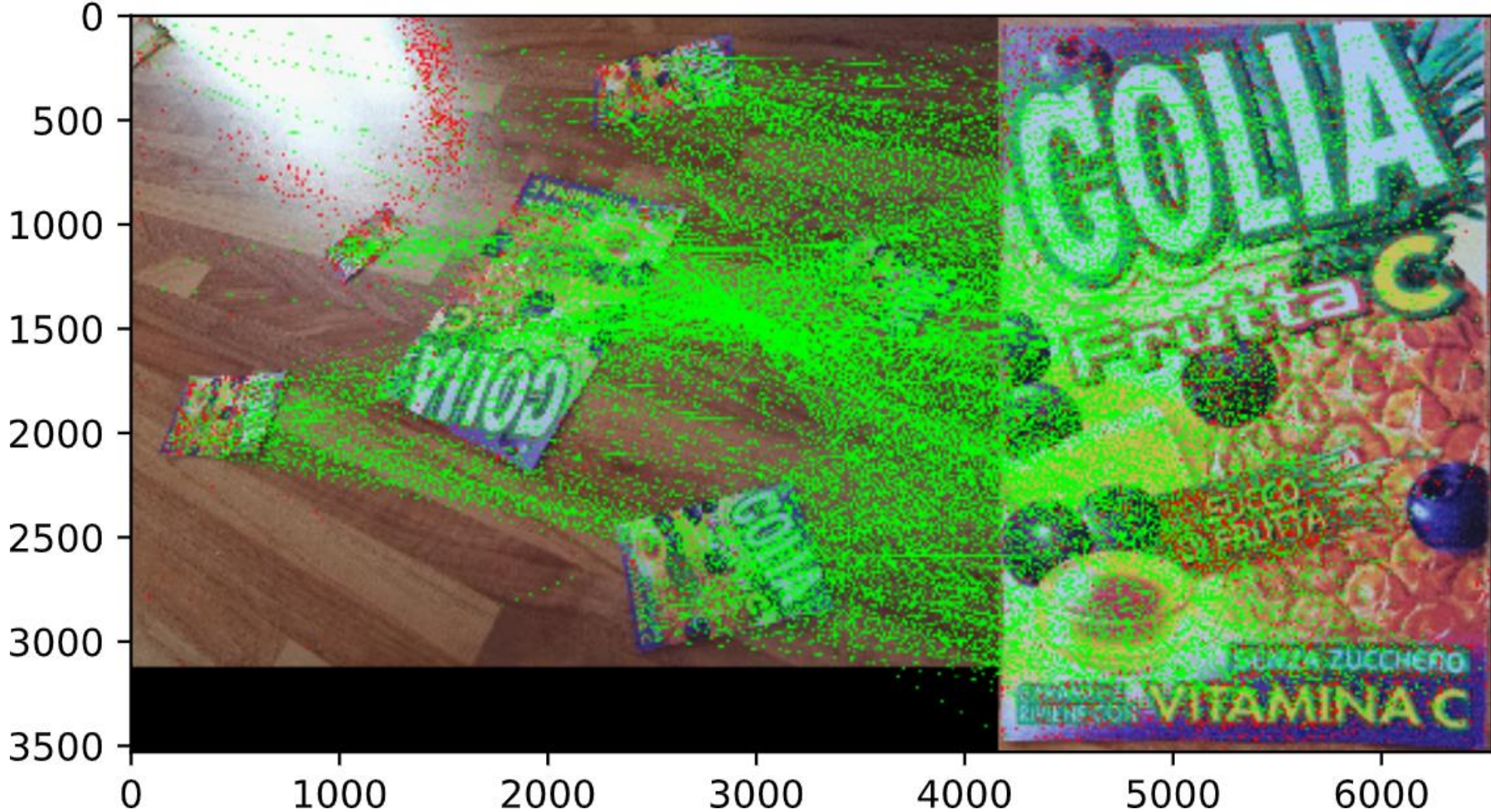


Image Courtesy of Filippo Leveni



# Sequential Ransac Iterations





# Sequential Ransac Image Rectification



Image Courtesy of Filippo Leveni

# Sequential Ransac: Histogram Matching



Image Courtesy of Filippo Leveni



# Sequential Ransac: Image Differences

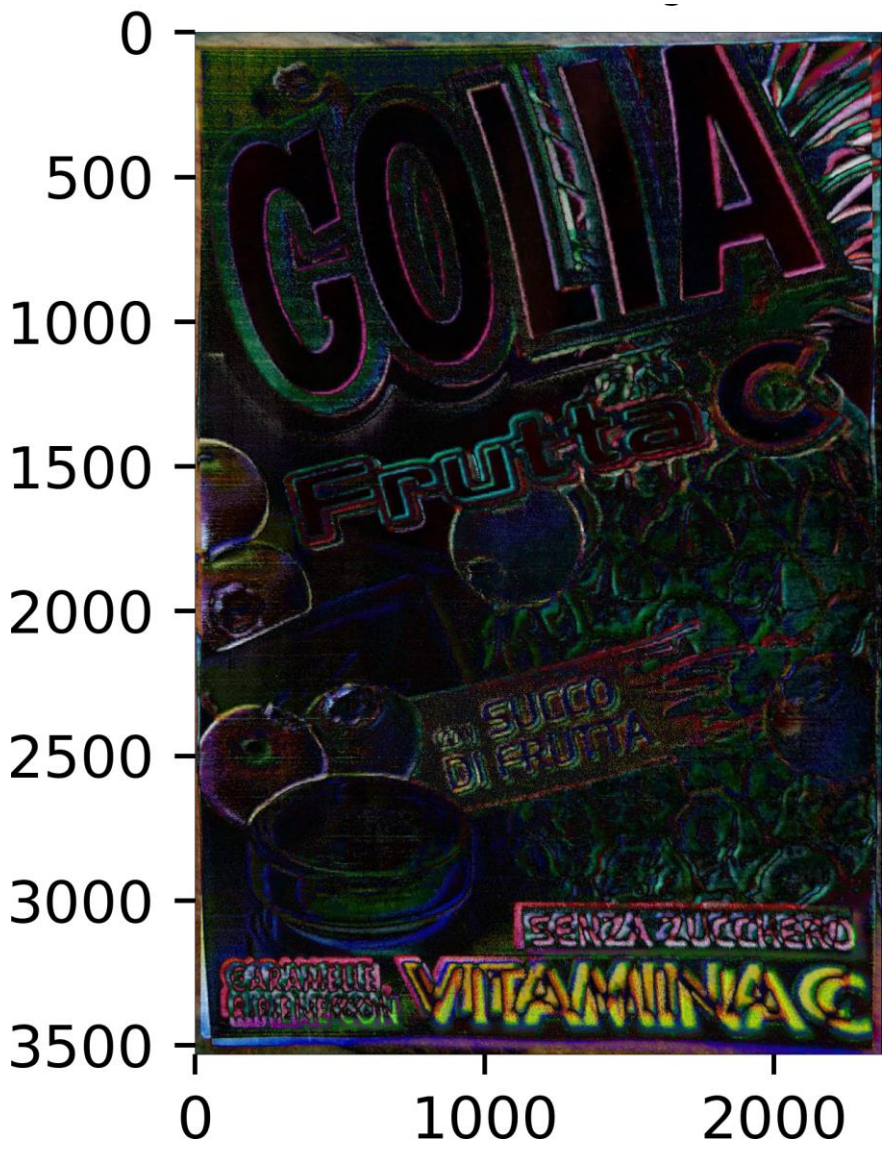


Image Courtesy of Filippo Leveni

# Sequential Ransac





# Sequential Ransac Image Rectification





# Sequential Ransac Image Rectification



# Sequential Ransac Issues: Radial Distortions



Image Courtesy of Filippo Leveni



# Sequential Ransac Issues: Radial Distortions



Image Courtesy of Filippo Leveni

# Sequential Ransac Issues: Radial Distortions



Image Courtesy of Filippo Leveni



# Sequential Ransac Issues: Radial Distortions



Image Courtesy of Filippo Leveni

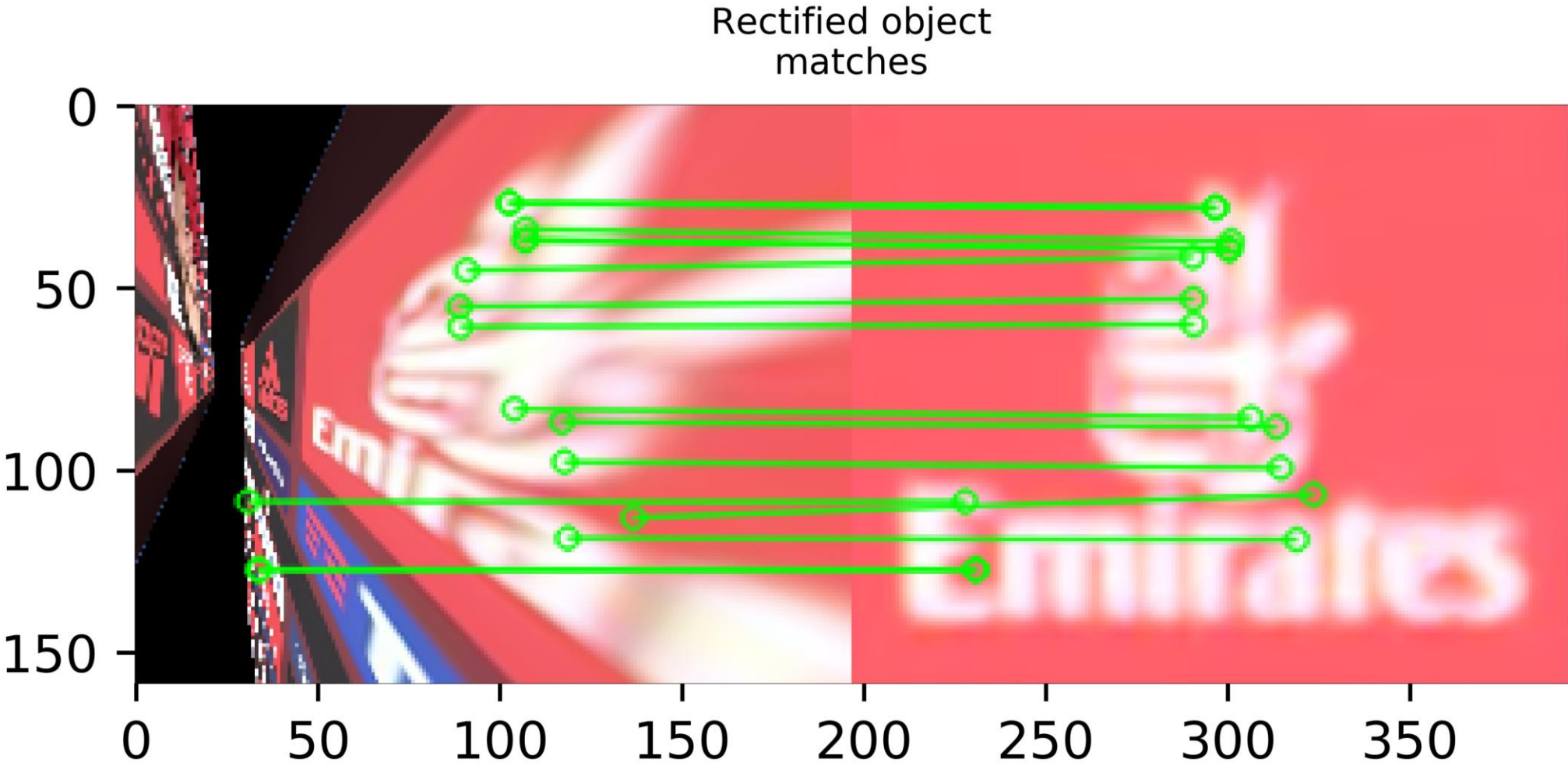
# Sequential Ransac Issues: Outliers



Image Courtesy of Filippo Leveni

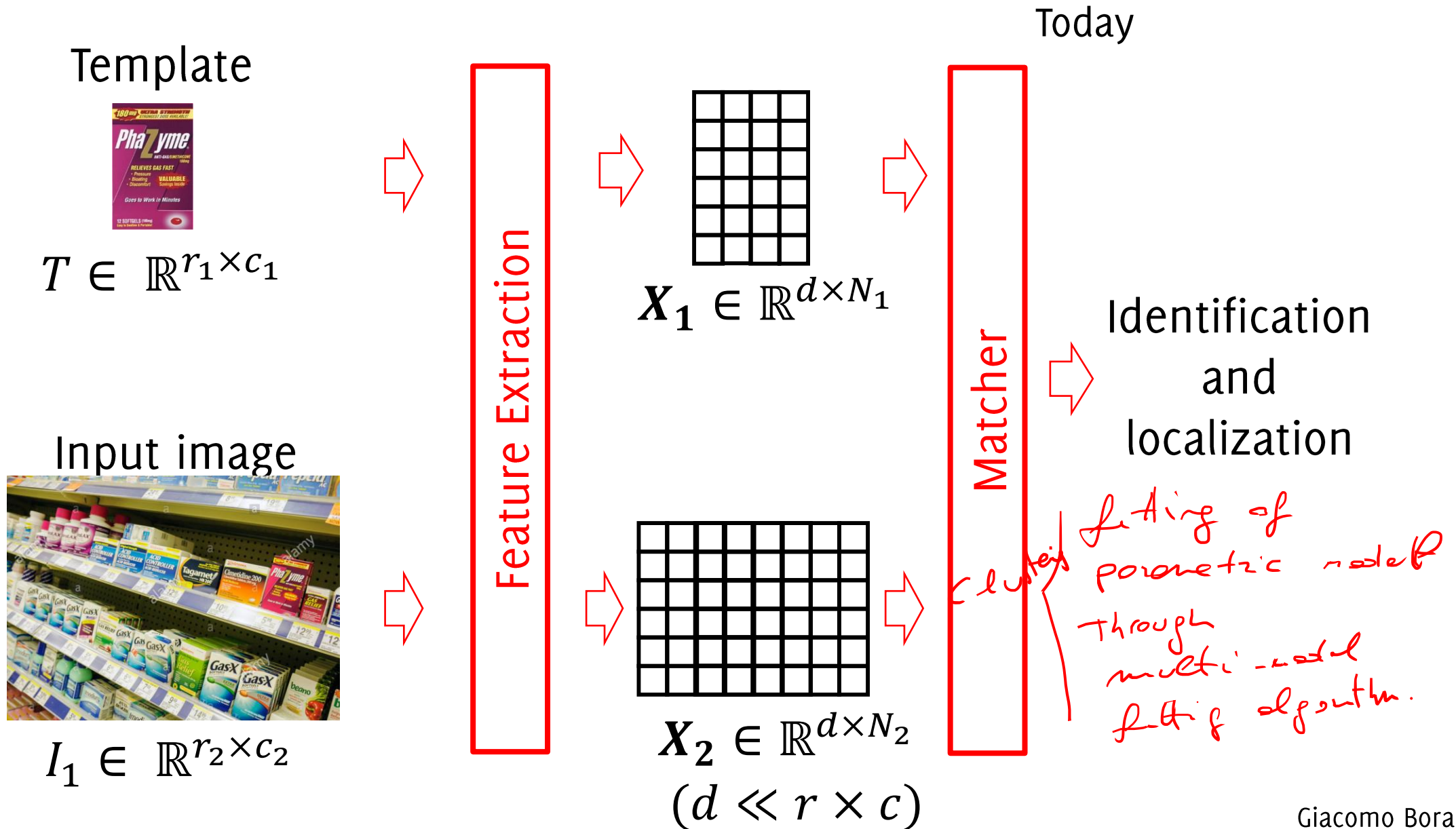


# Sequential Ransac Issues: Outliers





# Object Recognition by Feature Extraction



# Object Recognition by Computer Vision Features:

## Advantages:

- Leverage Geometric Properties of the Scene
- Detection of the object in any position / location / zoom level is possible, as long as deformation are perspective
- It is possible to rectify each detected instance and compare it against the template to spot minor differences
- No need of training data, just a template
- Engineered to be invariant to a set of photometric and geometric transformations and partial occlusions
- Naturally designed for object detection
- Accurate localization, and distortion correction
- Can process very large images efficiently

# Adding Homography Constraint

Each estimated homography can be used to rectify the each detected region and make it pixel-wise comparable with the template



# Object Recognition by Computer Vision Features:

## Cons:

- Not effective out of the set of invariant transformations
- It is often necessary for the template to feature some geometric regularity
- Most of computer vision features ignore color information
- When searching for multiple templates it can be slow



# Tempalte matching limitation (it is not cherry creme)



# Object Recognition by Computer Vision Features

## "Learning" phase:

- Extract features from the template(s)

## Detection phase:

- Extract features from the image
  - Keypoint detection
  - Descriptor Computation
- Match features with the reference template
- Prune matches to achieve object localization

# Project Description



# Goal

Identify yourself a realistic scenario for multi-template detection

- **Multiple templates have to be identified** within the same image
- The image should include multiple instances of each template
- Templates have to be detected by **fitting a parametric transformation**
- **Add a decision criteria** to determine whether matches give rise to a detection. If possible, implement a pixel-wise comparison.
- Include any advanced method to improve your algorithm

You can possibly use multiple images to demonstrate the performance of your algorithm. Templates have not to be selected from the test image.

Ideally these should be acquired in «ideal settings» (as in assignment 2)

# Goal

**Identify yourself a realistic scenario for multi-template detection**

- **Multiple templates have to be identified** within the same image
- The image should include multiple instances of each template
- Templates have to be detected by **fitting a parametric transformation**
- **Add a decision criteria** to determine whether matches give rise to a detection. If possible, implement a pixel-wise comparison.
- Include any advanced method to improve your algorithm

You can p  
your algo  
Ideally th

Extending the solution shown in Homework 2 to  
address these problems have to be consider the bare  
minimum for the project

formance of  
image.  
(ment 2)

# Ideally...

To choose your research idea you have to:

- Identify a situation where the template matching implemented in the Homework 2 fails (or exhibit severe limitations)
- Understand what are the reasons of such failures / limitations
- Find out a nice idea to overcome these limitations
- You are ready to start your project!

# Distinguish among multiple similar templates



# Distinguish multiple similar templates





# Distinguish multiple similar templates





# Detection under blur, shadings





# Detect non-planar templates



# Detect non-planar templates





# A few ideas (just as as a referece...)

Expand the project in the direction you prefer to "make it cooler". Use techniques presented in this course to address problems like:

- Improve RanSaC efficiency by optimizing the algorithm
  - modify stopping criteria of RanSaC
  - assume a first detection is given, estimate a pixel/cm ratio in order to perform a tile-based analysis of each template
- Improve RanSac rffectiveness
  - Improve **outlier reject criteria** during random search inside RanSaC, to discard bad homographies. Don't mind if this becomes extremely slow w.r.t. to openCV functions
  - Try another multi-model fitting algorithm
- Address a different scenario where matches need to be found by different criteria.

# A few ideas (just as as a referece...)

Improve: detection-Decision criteria

- Handling occluded instances
- Handling photometric distortions (e.g. blur, noise shadows)

Expand the template model

- Estimate fundamental matrices to handle multiple instances of 3D objects

Prioritize scanning order over multiple templates. Possible criteria are:

- deep learning to speed up detections by defining which template to search first
- color analysis to define which template to match first
- consider match density

--- To be continued

# Image Classification and Retrieval By Computer Vision Features

Giacomo Boracchi

CVPR USI, May 12 2020



# Feeding Computer Vision Features to a Classifier

Part of these lectures are from  
ICCV 2009 course: *Recognizing and Learning  
Object Categories*

[http://people.csail.mit.edu/torralba/shortCourse  
RLOC/index.html](http://people.csail.mit.edu/torralba/shortCourse/RLOC/index.html)

# The rationale

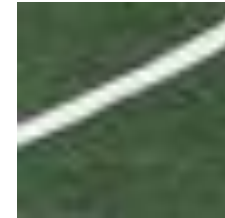
Difficult to extract distinctive features to describe the **whole image**

However... when I show you these:

# The rationale

Difficult to extract distinctive features to describe the **whole image**

However... when I show you these:



# The rationale

You can clearly recognize that these are from a football pitch



Analyze small image regions to infer the image content



# Feature Extraction and Classification

Perform classification by analyzing image features.

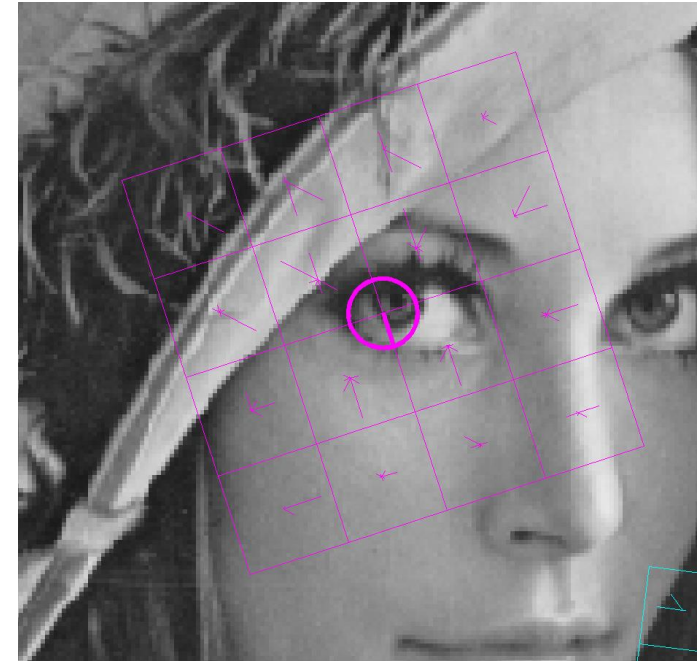
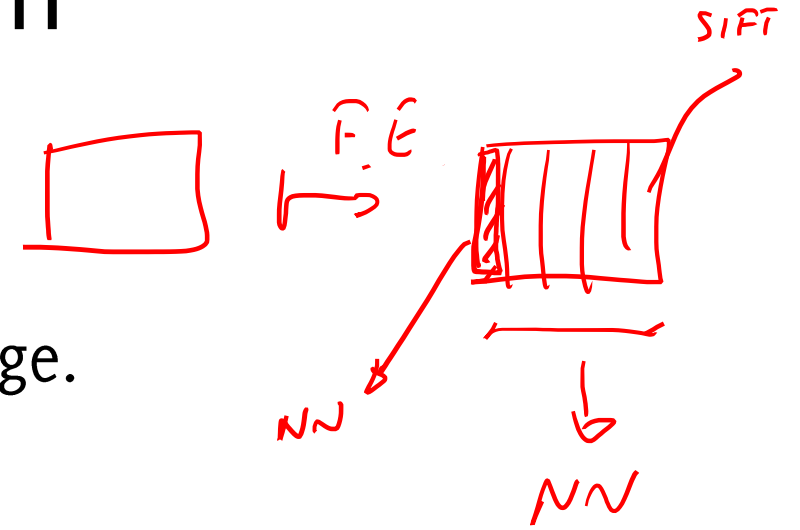
## Advantages:

- Keeps only the **most discriminative regions** in the image.
- **Reduce** the overall **dimensionality**.
- Use **intermediate representations** to classifying images.

## Issues:

- **Different images provide different number of features and in random order**
- **Impossible to label features**

Pixel-wise representation cannot be straightforwardly replaced by feature-based representations



# The Feature Extraction Perspective



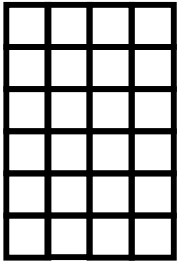
$$I_1 \in \mathbb{R}^{r_1 \times c_1}$$

Input image



$$I_1 \in \mathbb{R}^{r_2 \times c_2}$$

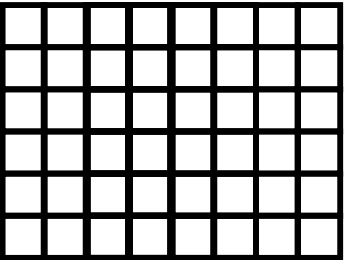
Feature Extraction



$$X_1 \in \mathbb{R}^{d \times N_1}$$

Classifier

“roofs”



$$X_2 \in \mathbb{R}^{d \times N_2}$$

$$(d \ll r \times c)$$

“drugs”

$$y \in \Lambda$$



# The Feature Extraction Perspective



$$I_1 \in \mathbb{R}^{r_1 \times c_1}$$

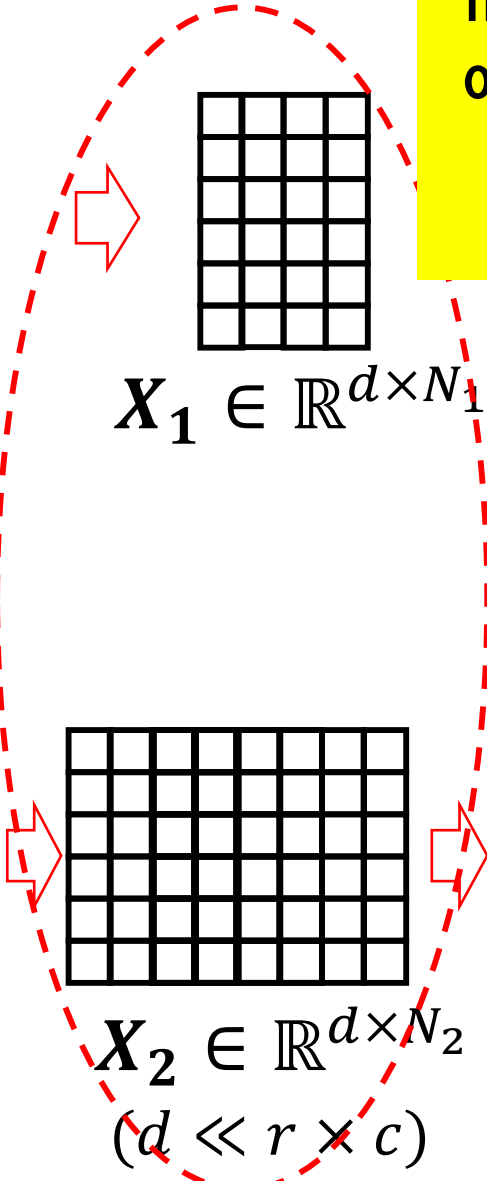
Input image



$$I_1 \in \mathbb{R}^{r_2 \times c_2}$$



Feature Extraction



$$X_1 \in \mathbb{R}^{d \times N_1}$$

$$X_2 \in \mathbb{R}^{d \times N_2}$$

$(d \ll r \times c)$

These are in different numbers and in order, thus overall the classifier should take as input different dimensions

Classifier



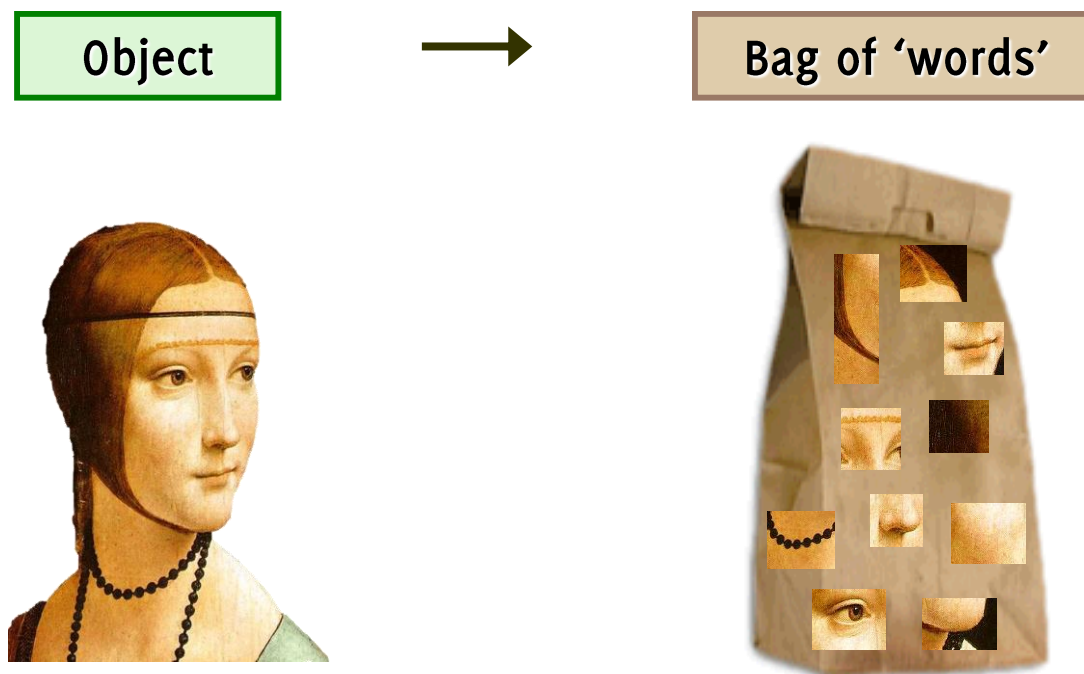
“drugs”

$$y \in \Lambda$$

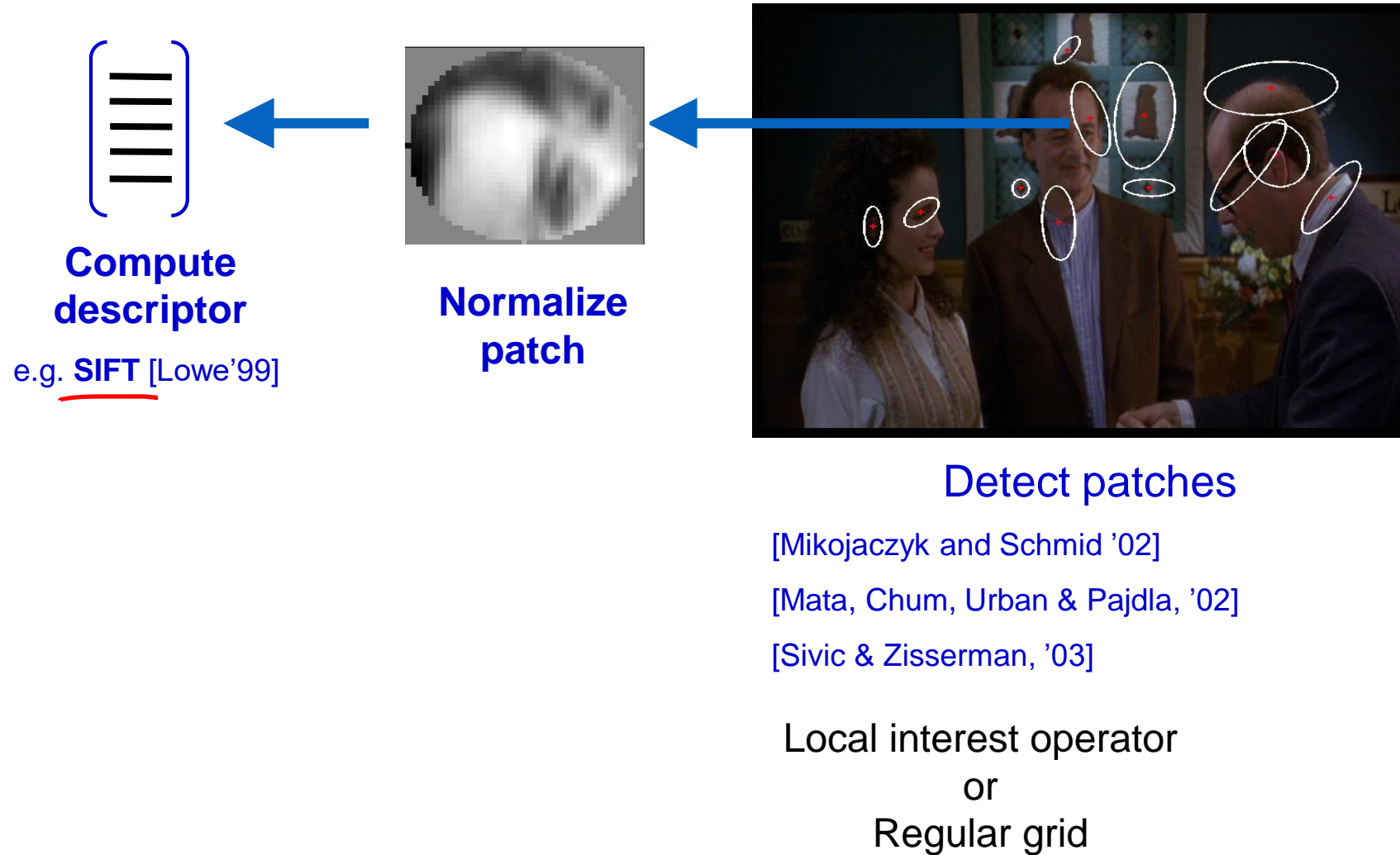
# Bag of Words Model (BoW)

To represent an image using BoW model, an image can be treated as a **document**.

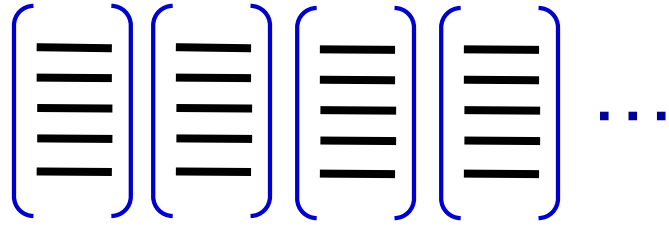
An image is modeled as a **collection of patches / descriptors / features**. These local descriptors are the words, the image is the whole document.



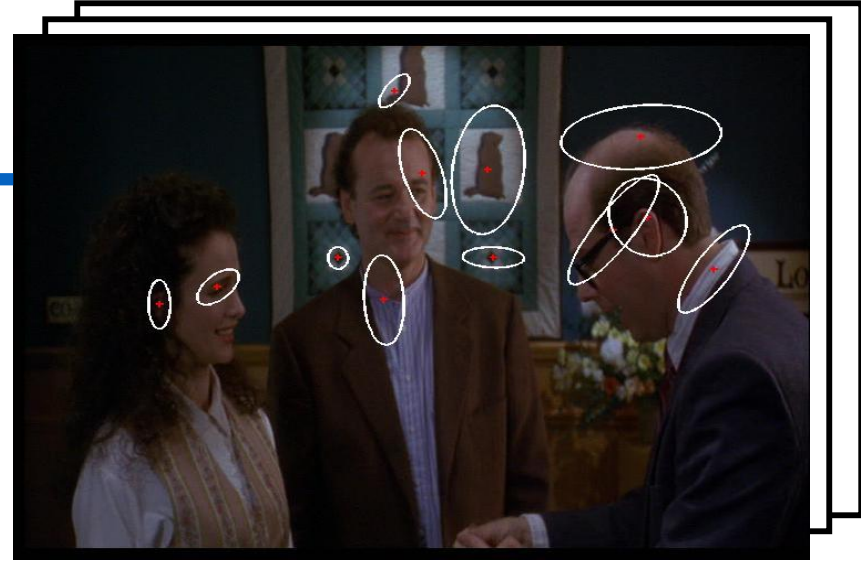
# 1. Feature detection and representation



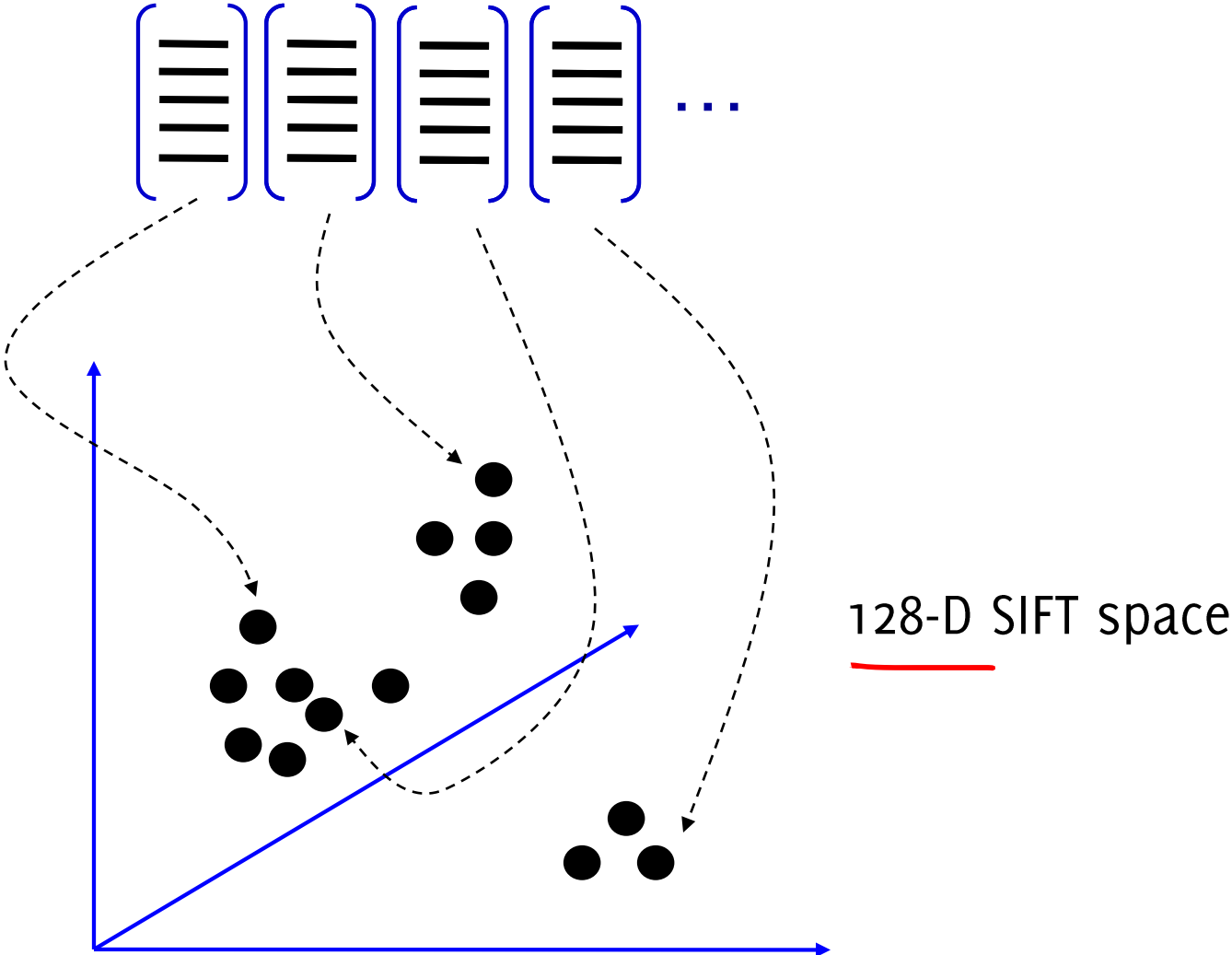
# 1. Feature detection and representation



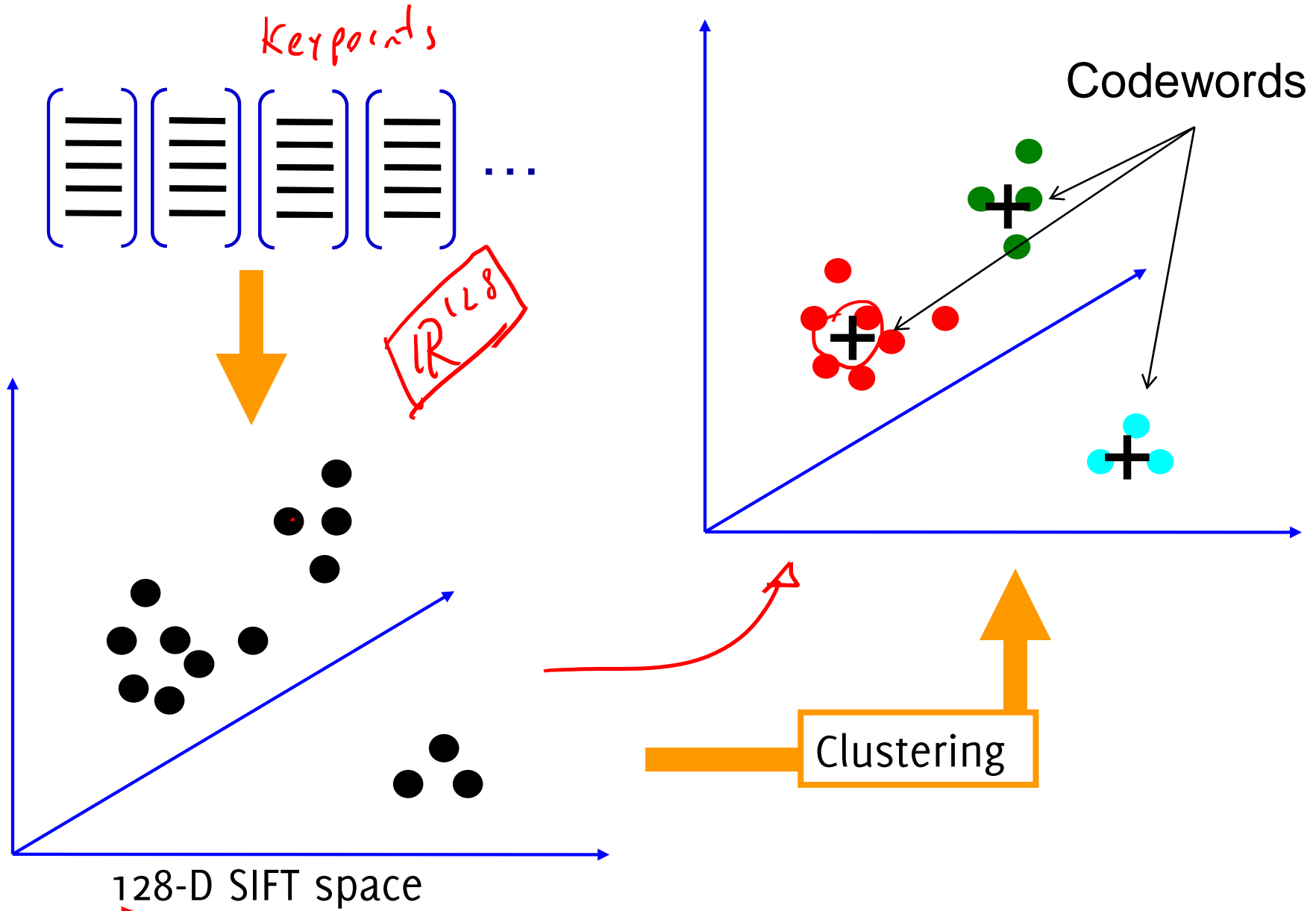
Extract other features  
from the same image and  
from other images of the  
same class



# 2. Codewords dictionary formation

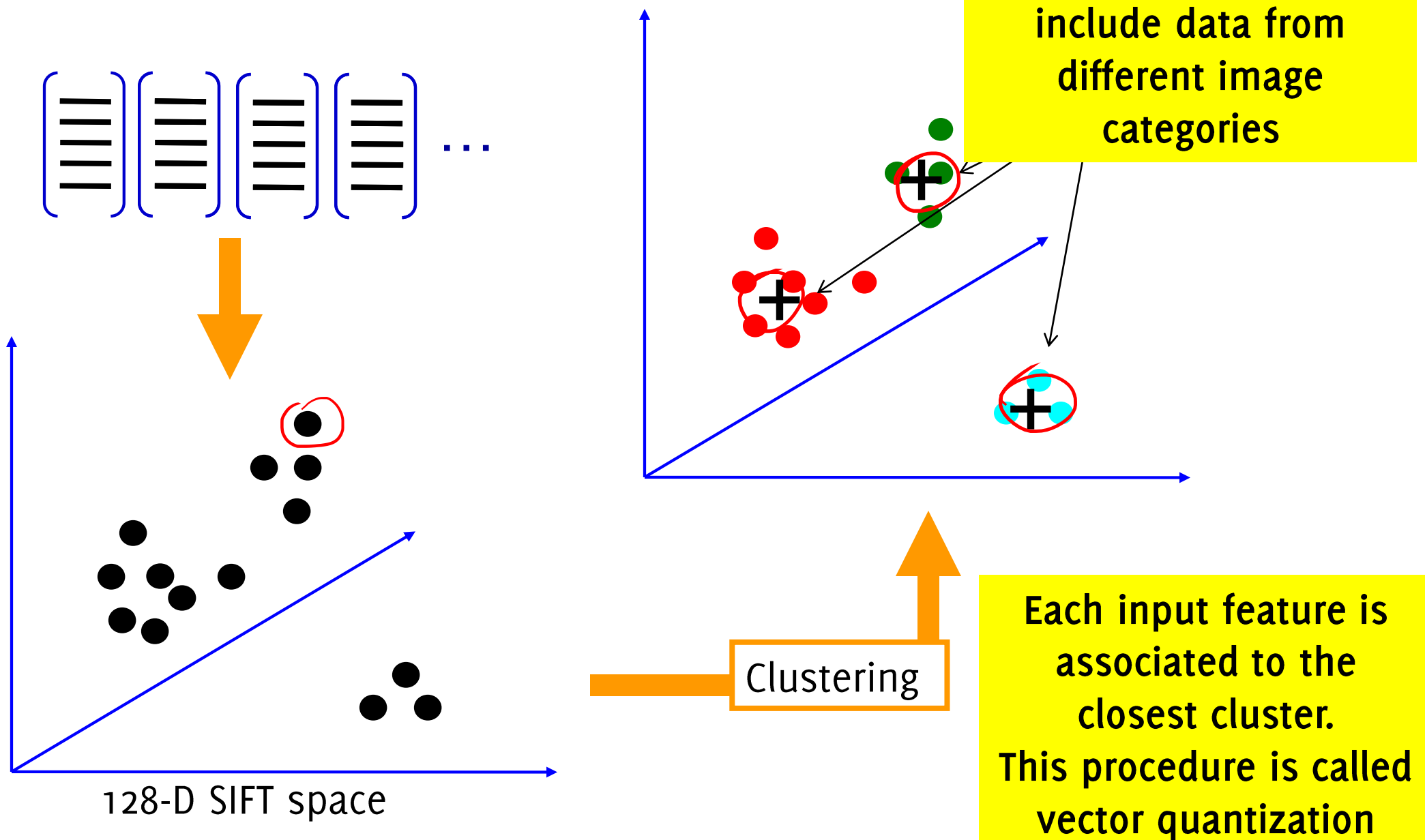


## 2. Codewords dictionary formation





## 2. Codewords dictionary formation



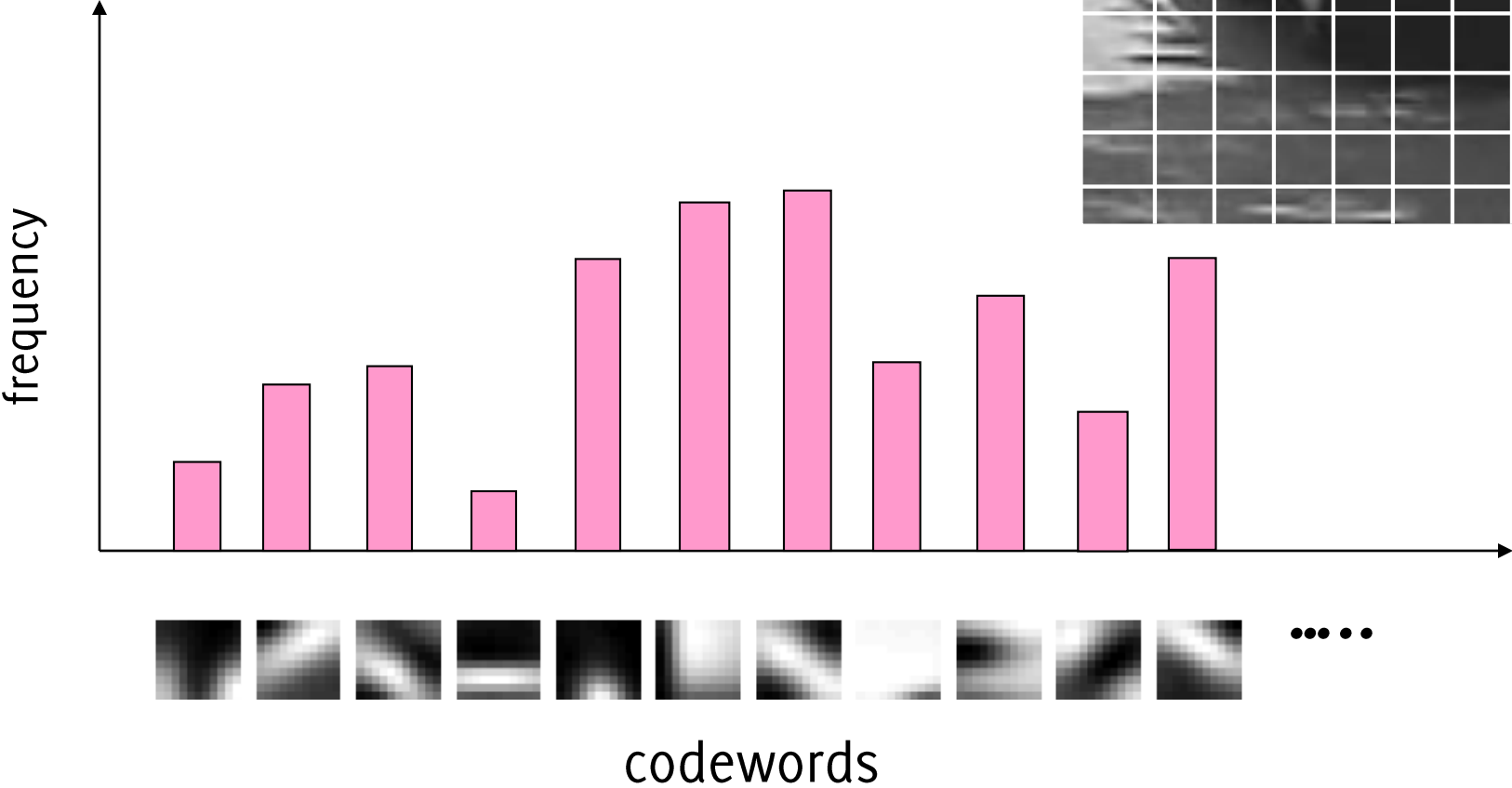
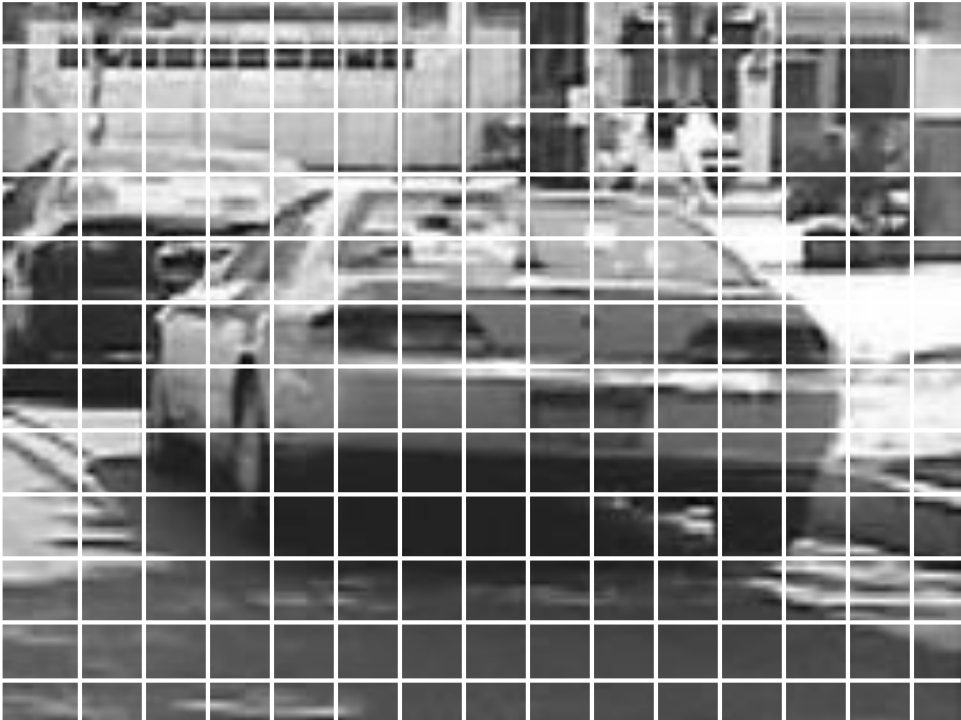
# Image Representation

An image becomes an histogram of features w.r.t. codewords (cluster centers) extracted from salient point locations



# Image Representation

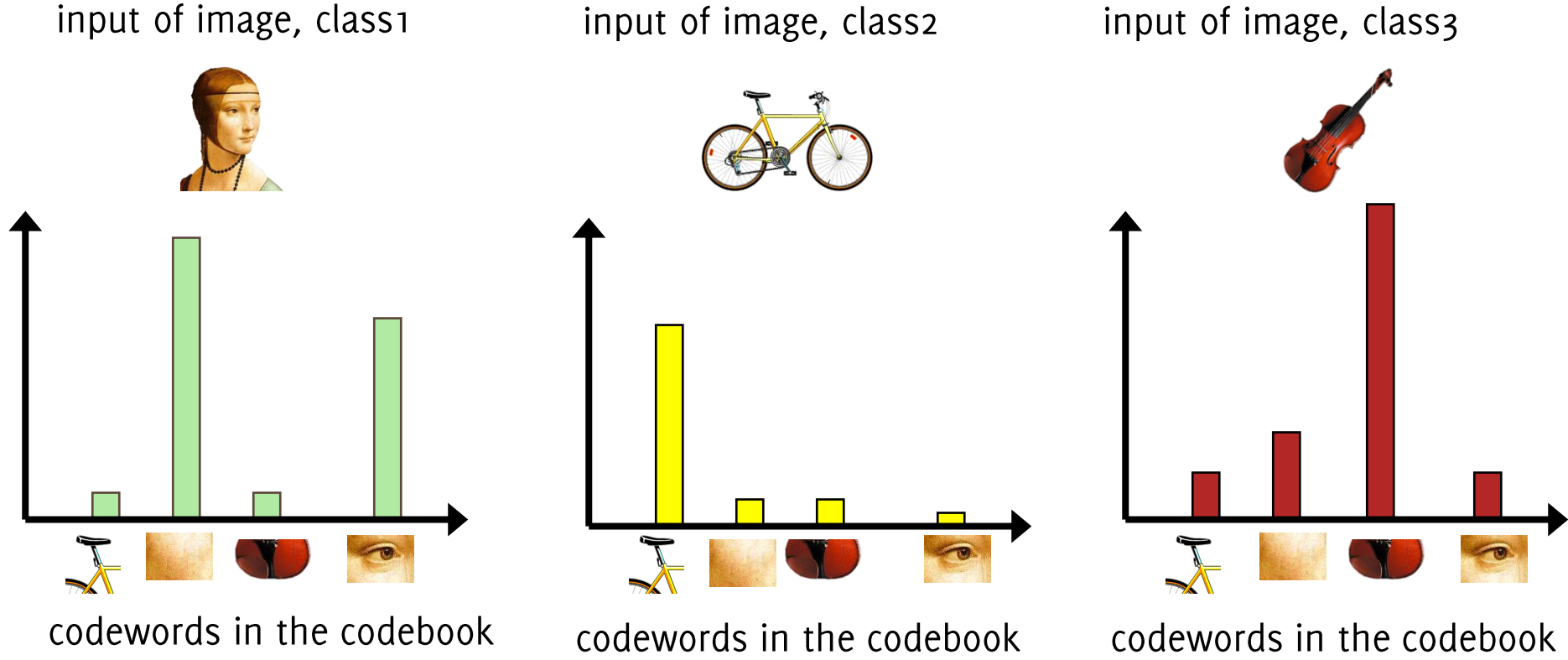
An image becomes an histogram of features w.r.t. codewords (cluster centers) extracted over a grid (as in HOG)



# Classifiers for BOVW Representations

## BOVW: Bag of Visual Word

Learn a model that best represents the distribution of codewords in each category of scenes

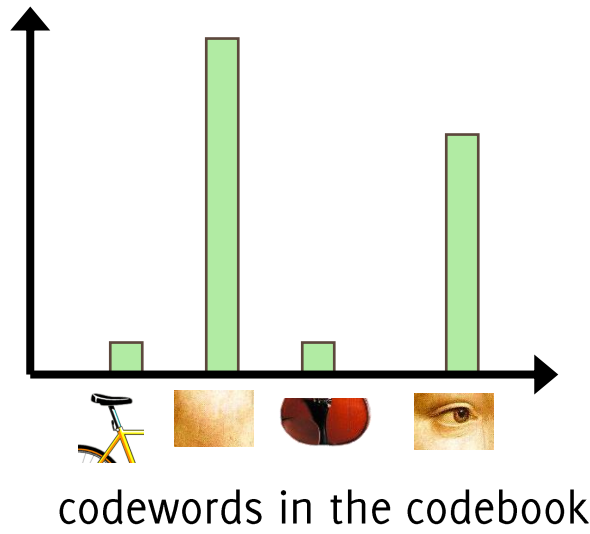


# Classifiers for BOVW Representations

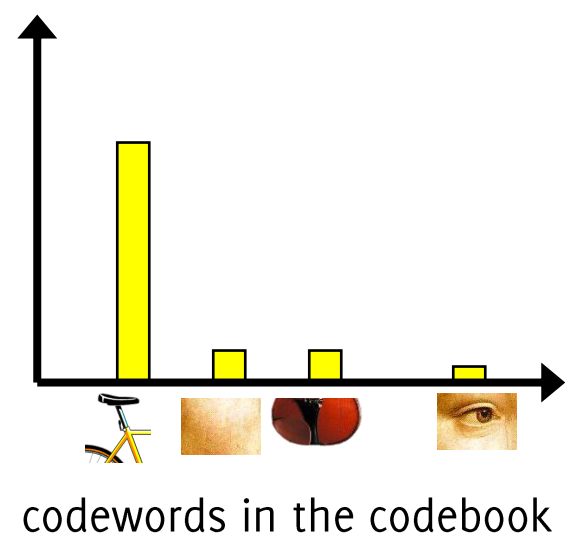
## BOVW: Bag of Visual Word

Learn a model that best represents the distribution of codewords in each category of scenes

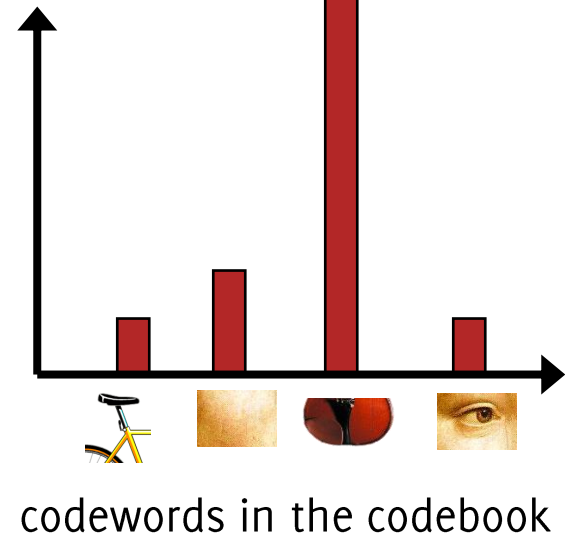
Distribution of codewords, for class1 (faces)



Distribution of codewords, for class2 (bicycles)



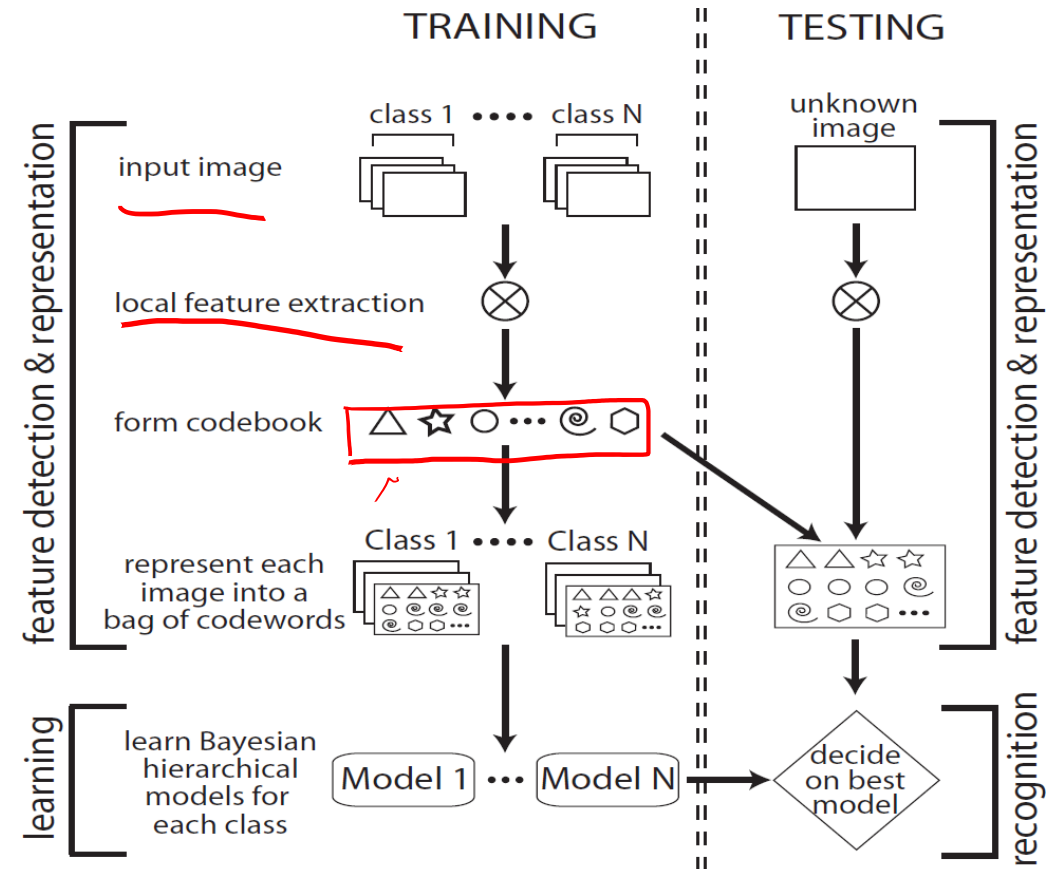
Distribution of codewords, for class3 (violin)



# Learning the BOVW model

Given a bunch of images in each class,

- Extract features (e.g. SIFT)
- Learn codewords that are representative of several similar patches (e.g. k-means, VQ)
- The codebook contains all the codewords
- Learn a model that best represents the distribution of codewords in each class



**Figure 2.** Flow chart of the algorithm.



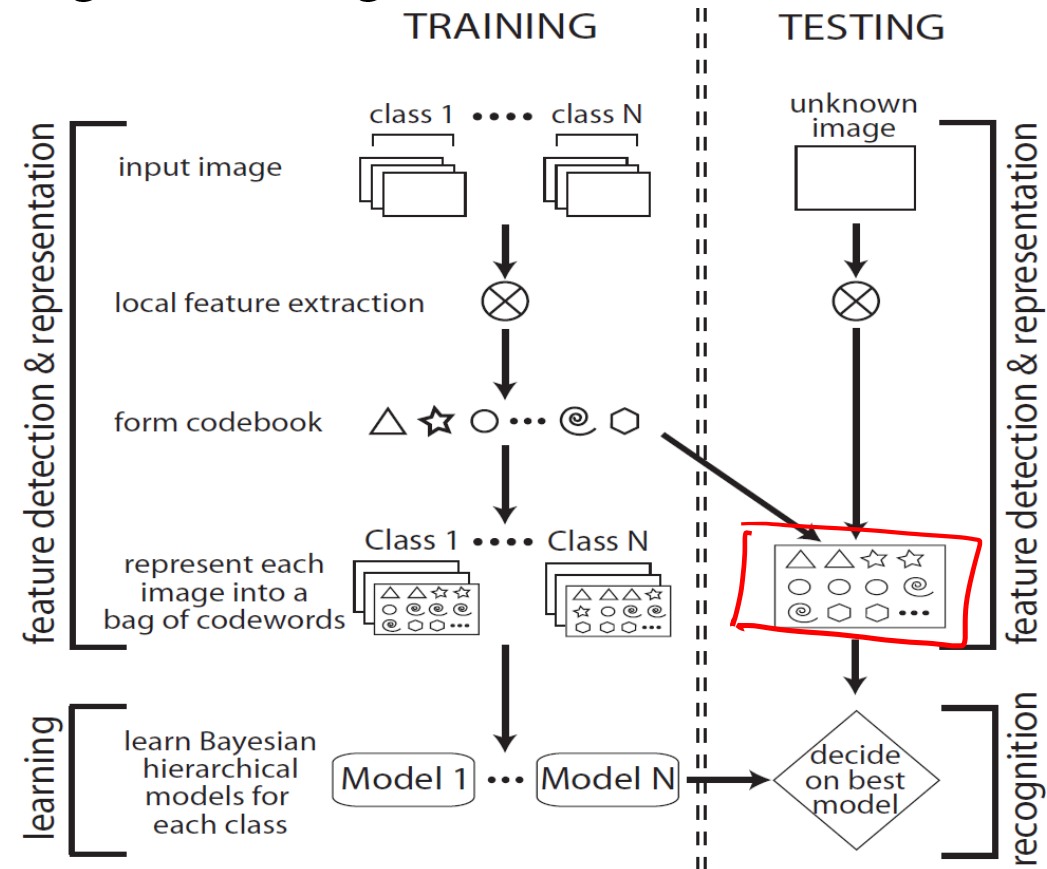


# Learning the BOVW model

These histograms are used as features describing the image.

A classifier is learned by:

- Training a generative model on histograms (e.g., a Naïve Bayes classifier or a Hierarchical Bayesian model that describes classes and themes)
- Training a discriminative model (e.g. an SVM or a Neural Network)

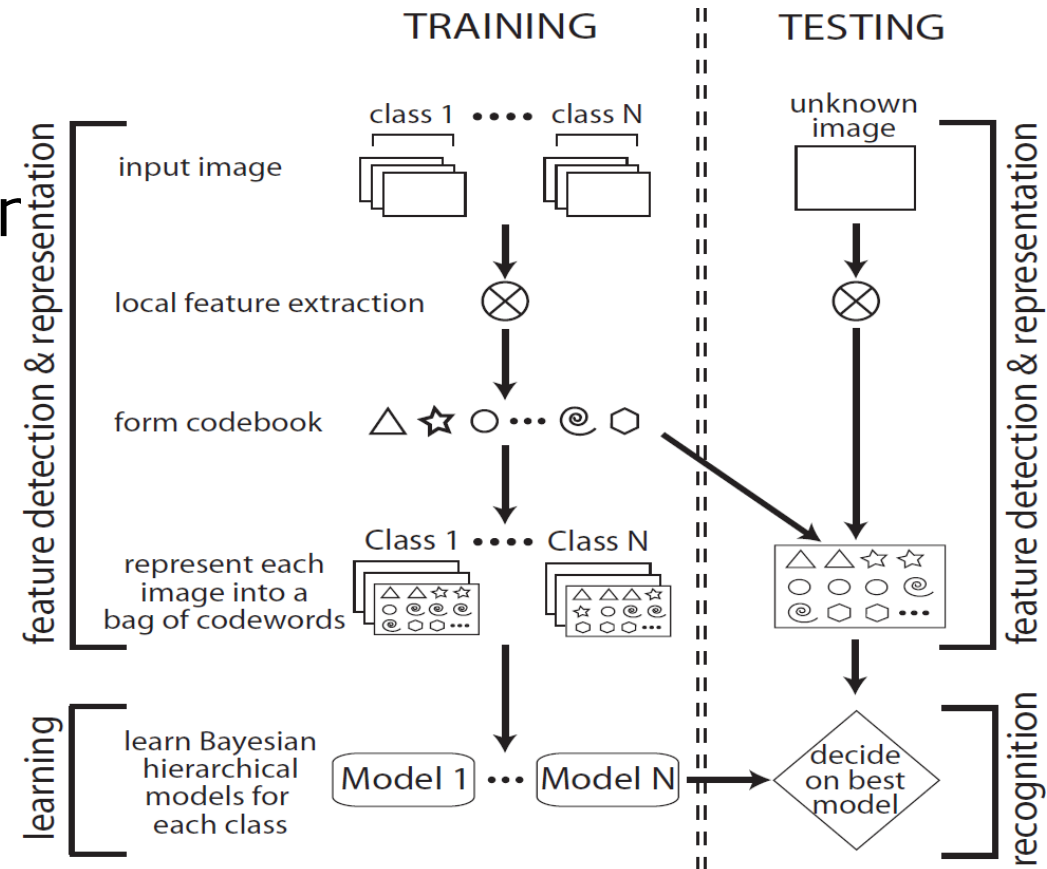


**Figure 2.** Flow chart of the algorithm.

# Classifying using the BOVW model

When classifying a test image:

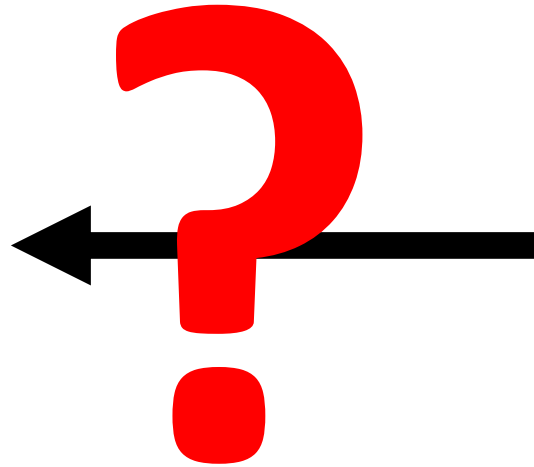
1. extract features
2. match each feature the closest codeword in (e.g. by clustering)
3. find the model that fits best the corresponding histogram of codewords of the image.



**Figure 2.** Flow chart of the algorithm.

# Limitations of BoVW model

One of the notorious disadvantages of BoW is that it ignores the spatial relationships among the patches, which are very important in image representation.



# Limitations of BoVW model

One of the notorious disadvantages of BoVW is that it ignores the spatial relationships among the patches, which are very important in image representation.

Possible solutions:

- Take into **account the relative position of codewords** in generative models.
- For discriminative models, perform pyramid matching by **partitioning the image into increasingly fine sub-regions** and compute histograms of local features inside each sub-region.

# Image Retrieval

# Example of Image Retrieval

Query Image



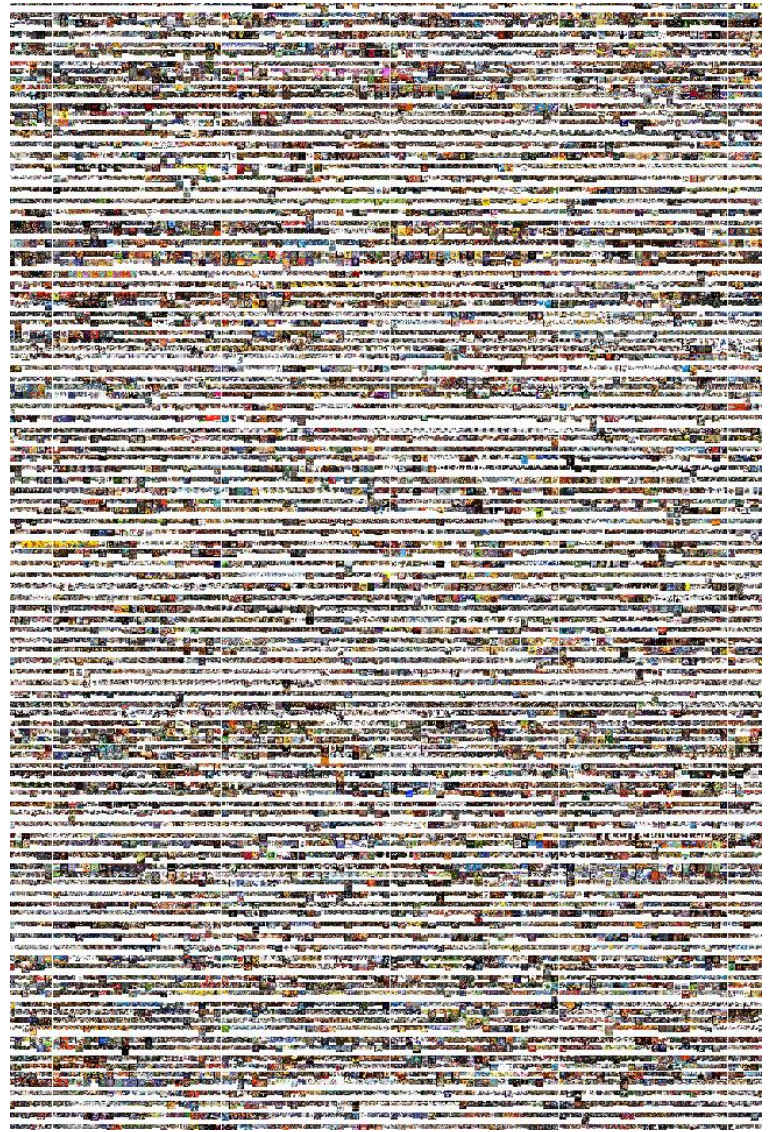


# Example of Image Retrieval

Query Image



Dataset of images

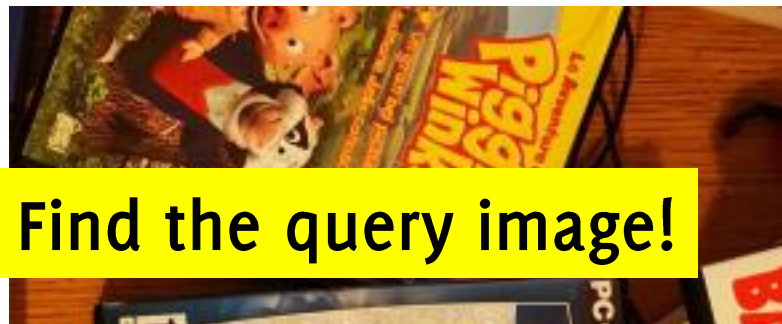




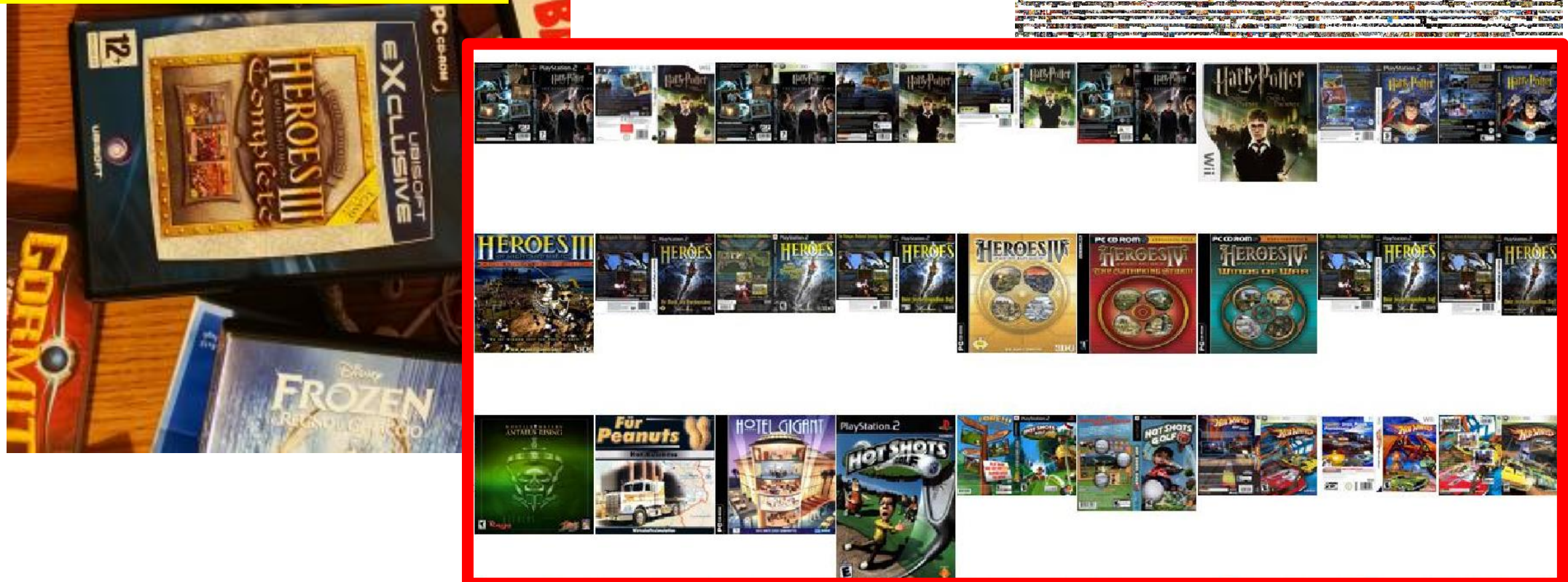
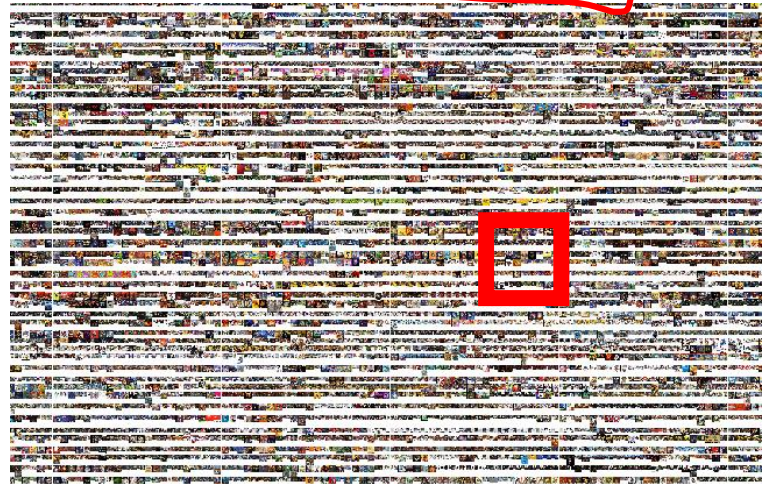
# Example of Image Retrieval

Dataset of images

Query Image



100%

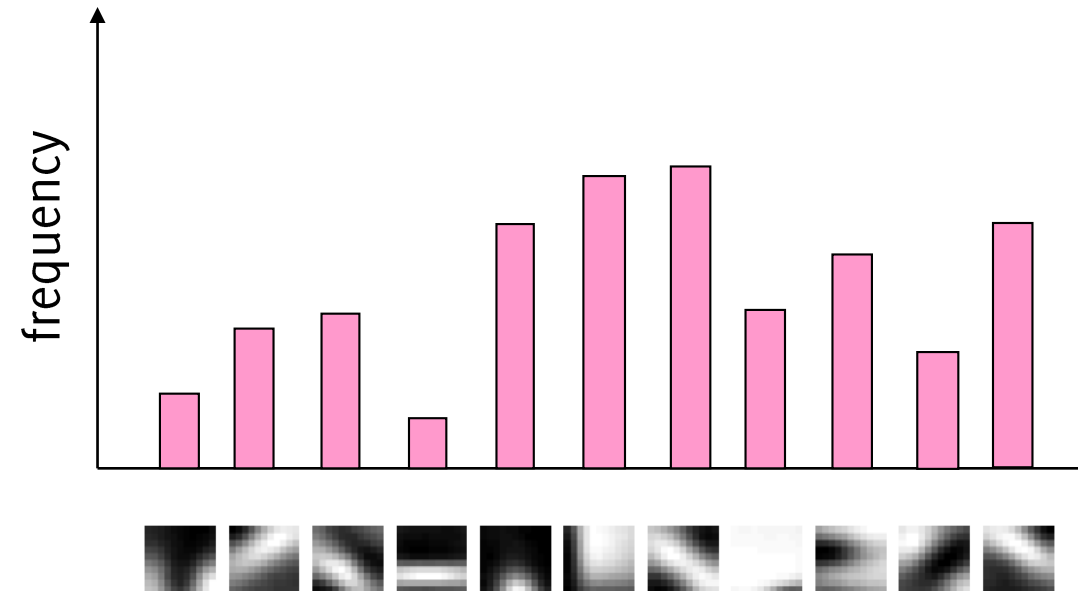


# Representation

As in text retrieval, each image  $I$  is assigned to a Bag of Visual Word descriptor, thus a vector of frequencies of each codeword

$$I \rightarrow \boxed{f_I} \in \mathbb{R}^d$$

Being  $d$  the number of codewords (i.e. the dictionary size)

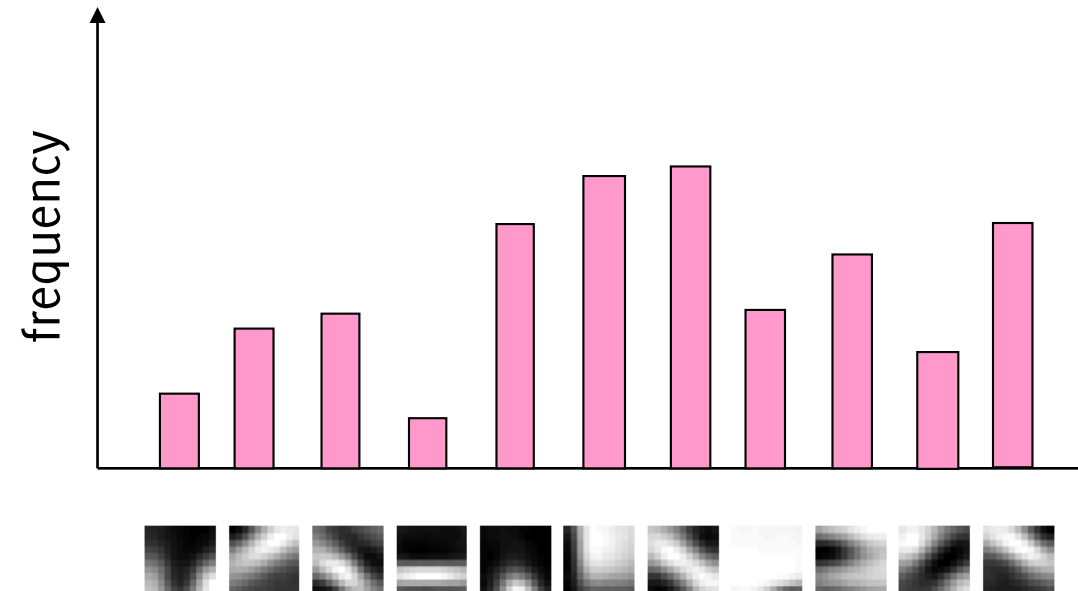


# Representation weighting

## *Term-frequency – inverse document frequency*

In text retrieval it is usual to apply a weighting to the components of the representation vector, rather than using the frequency vector directly for indexing

"the" "only" "am"  
"epipolar" ←



# Representation weighting

## *Term-frequency – inverse document frequency*

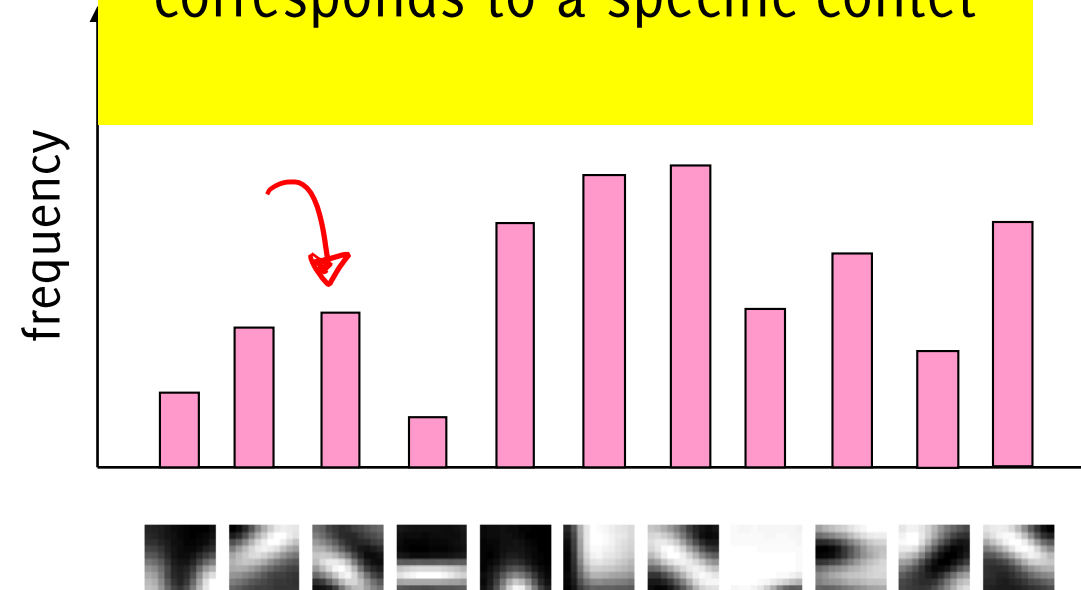
The weight to term  $t_i$  in a representation is

$$w_i = \frac{n_{id}}{n_d} \log \frac{N}{n_i}, \quad i = 1, \dots, d$$

Where  $n_{id}$  number of occurrence of term  $t_i$  in the document,  $n_d$  the number of terms in the document,  $N$  the number of terms in the dataset,  $n_i$  the occurrence of  $t_i$  in the dataset

Larger weights to word that are often occurring in the document and that are rare in the dataset (thus that are very specific)

e.g. words like «the», «for» are not informative, others like «liver» or «epipolar» more clearly corresponds to a specific content



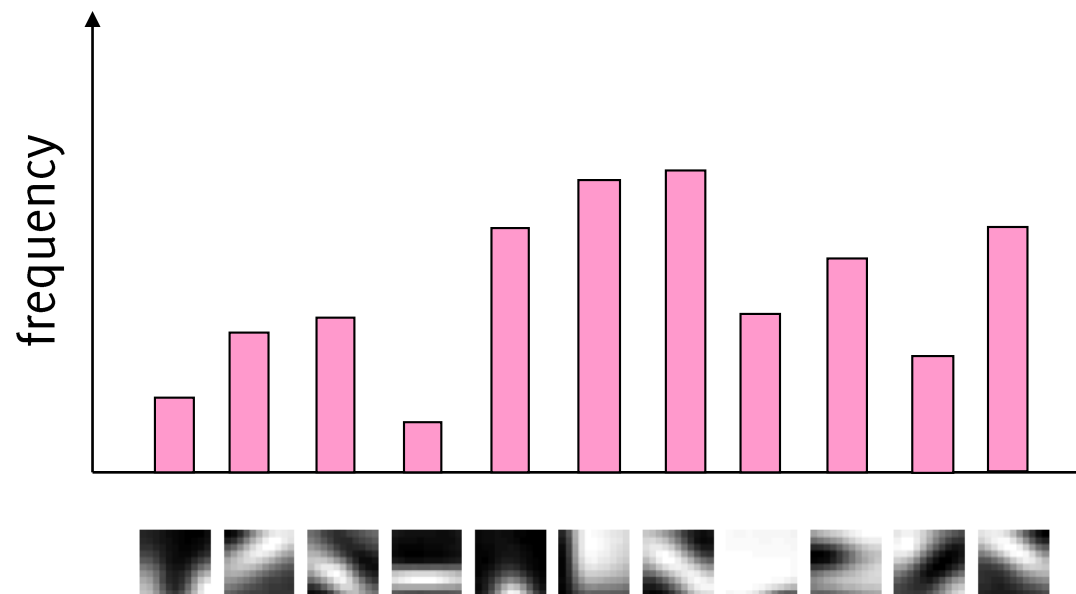
# Representation weighting

Each image is associated to a weighted frequency vector

$$I \rightarrow w.* f_I$$

And the same is done for images in a database

Larger weights to word that are often occurring in the document and that are rare in the dataset  
(thus that are very specific)





# Large Scale Image Search

## Building the database:

- Extract features from the database images
- Learn a vocabulary using k-means (typical k: 100,000)
- Compute weights for each visual codeword
- Create an inverted file mapping words to images (this is a database structure for efficient searches)

# Large Scale Image Search

Searching for a query image:

- Extract features from the image  $I$
- Compute weights for each visual codeword
- For each image  $J$  in the dataset, compute cosine similarity between  $I$  and  $J$ . Chose the image yielding the minimum distance

$$\langle w_I^* f_I, w_T^* f_T \rangle$$

# Example of Visual Word Matches

*query use*

