Visuelle Perzeption für Mensch-Maschine Schnittstellen

Vorlesung, WS 2013 / 2014

Prof. Dr. Rainer Stiefelhagen
Dr. Saquib Sarfraz

Institut für Anthropomatik
Karlsruher Institut für Technologie - KIT

http://cvhci.anthropomatik.kit.edu/
rainer.stiefelhagen@kit.edu

Basics: Pattern Recognition

WS 2013/14

Institute for Anthropomatics
Computer Vision for Human-Computer Interaction

Today

- Why pattern recognition and what is it?
- Dimensionality Reduction
  - Principle Component Analysis
- Classification
  - Bayes Decision Theory
  - Gaussian Mixture Models (GMM)
  - Linear Discriminant Functions

Overview

- Learn common patterns based on either
  - a priori knowledge or
  - on statistical information

- Why statistics?
  - Manual definition of patterns often tedious or not obvious
  - Important for the adaptability to different tasks/domains
  - Finally: Try to mimic human learning / better understand human learning

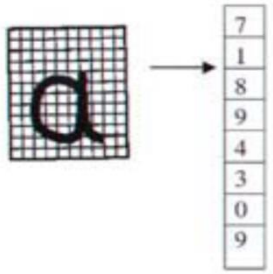
Pattern Recognition

- Supervised
  - Data samples with associated class labels
- Unsupervised
  - Data samples WITHOUT any labels

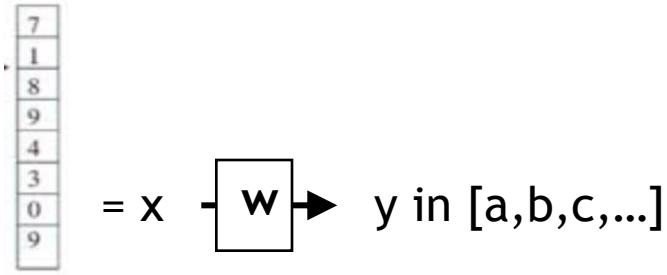
- Semi-Supervised / Weakly-Supervised Learning
  - Not a topic in this lecture

Pattern Recognition Stages

1. Feature Extraction
   - Which part of the data is most important

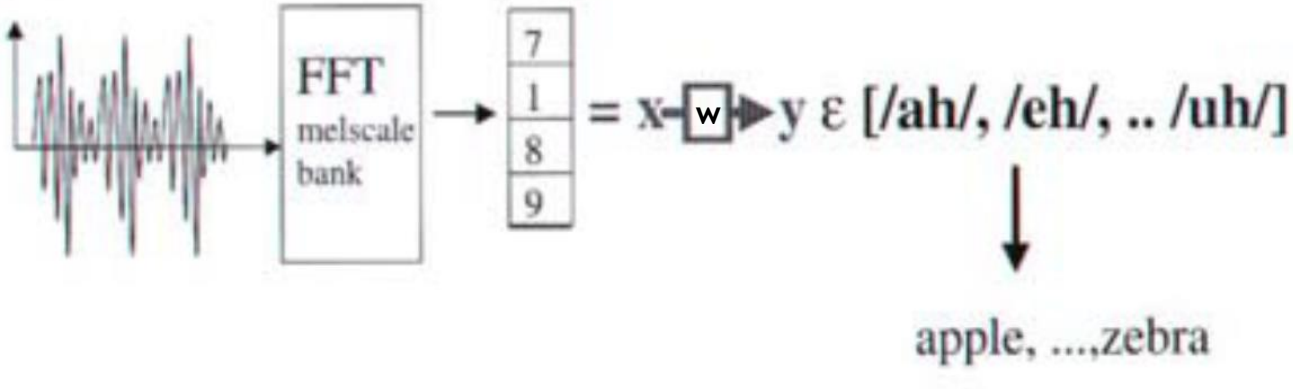


2. Classification/Learning
   - How can we map the extracted features to our desired output (supervised learning)?



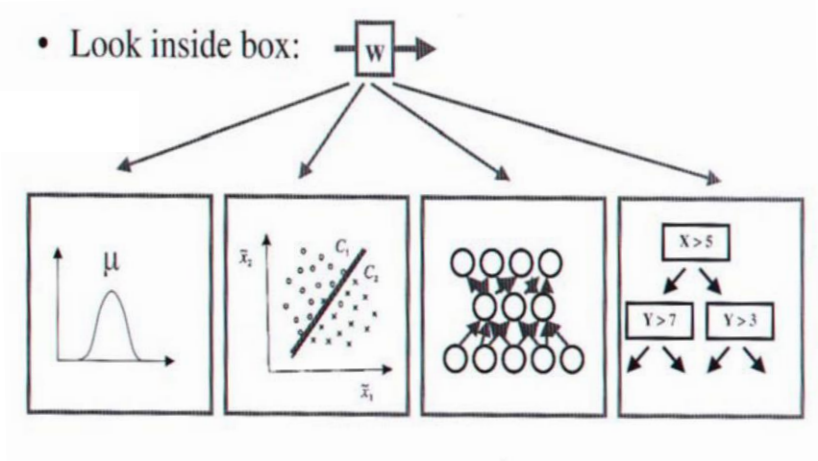
Examples

- Speech recognition



- Computer Vision
  - You will see many examples in the course of this semester …

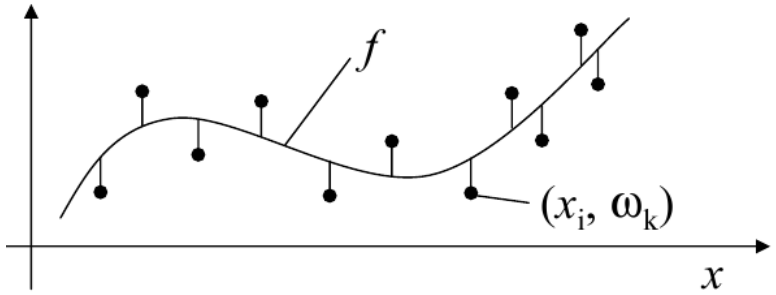
What's in the box?



- Parametric/Non-parametric distributions
- Naïve Bayes Classifier
- Support Vector Machines
- Neural Networks
- Decision Trees
- Boosting
- ….

Classification vs. Regression

- Classification
  - Decide to which class a feature vector belongs
  - Typically a small number of classes
  - Many basic methods are restricted to binary classification
- Regression
  - Function approximation
  - Multi-class problems can be addressed by defining a desired function value e.g. y_i=[0,..1,0]

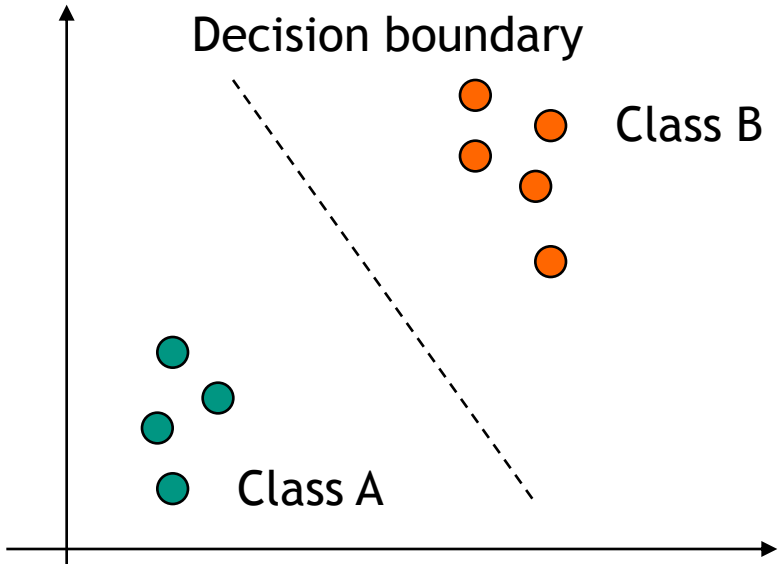


Problems and Considerations

- Features
  - How do I encode domain knowledge?
  - Allow invariance (e.g. rotation in the case of images)
    - see e.g. previous lecture
  - Which part of the data can be discarded as it represents redundant information?
  - How can we reduce the dimensionality?
    - i.e. how can we make the problem as simple as possible, but as complex as necessary

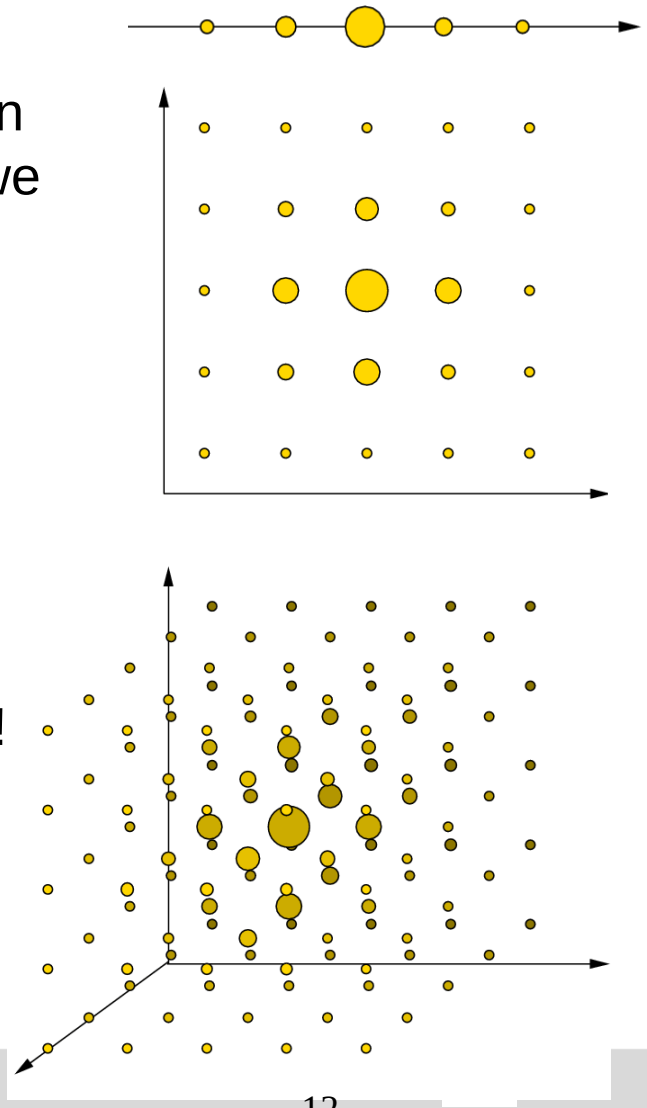
Curse of Dimensionality

- Generally: the more dimensions, the more difficult for a learning algorithm to extract patterns
- More dimensions = more degrees of freedom



Sample densities

- Consider interval sampled with 5 samples
- To sample with the same distance between samples (in direction of coordinate axes) we need:
  - 1D: 5
  - 2D: $5^2 = 25$
  - 3D: $5^3 = 125$
  - …
  - 100D: $5^{100} = 8 \cdot 10^{69}$
- Samples needed for same sample density increase exponentially with dimensionality!



Dimensionality Reduction

- Reduce the dimensionality of the data, while retaining relevant information

- Popular techniques:
  - Principal Component Analysis (PCA)
  - Linear Discriminant Analysis (LDA)
  - …

PCA – Short Version

PCA – Details -

17

First Steps: Sample Variance

- Given a set of samples $(a_1,...,a_n)$

  with mean a'

- An unbiased variance estimator is defined as

$$\mathrm{var}(a) = \frac{1}{n-1}\sum (a_i - a')^2 = \frac{1}{n-1}\sum z_i^{\,2}$$

- with $(z_1,...,z_n)$ the sequence with zero mean
- written in vectors $\frac{1}{n-1} zz^T$

First Steps: Covariance

- Given two sets of samples $(a_1,...,a_n)$ $(b_1,...,b_n)$

  with means a', b'

- A covariance estimator is defined as

$$\operatorname{cov}(a,b) = \frac{1}{n-1}\sum(a_i - a')(b_i - b') = \frac{1}{n-1}\sum c_i d_i$$

- with $(c_1,...,c_n),(d_1,...,d_n)$ sequences with zero mean
- written in vectors $\frac{1}{n-1} c d^T$

First Steps: Covariance Matrix

- Given n sets samples $v_1, \ldots, v_n$

  with means v_i' and $v_i = (a_1, \ldots, a_d)$

  d being the dimensionality

- A multi-dimensional covariance estimator is defined as

$$\operatorname{cov}(V)_{i,j} = \frac{1}{n-1}(v_i - v_i')(v_j - v_j')^T$$

- Written in vectors $\frac{1}{n-1} V V^T$

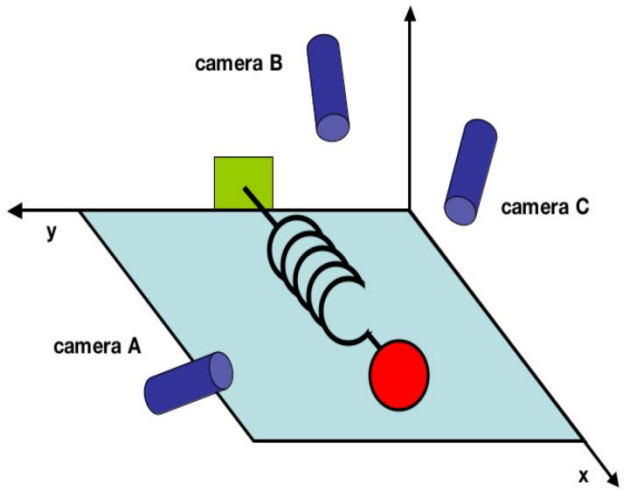
Variance Properties

- Properties:
  - var(a)=cov(a,a)
  - cov(a,b)=0 iff a,b are completely uncorrelated
  - cov(V) is a square-symmetric matrix (containing variances on the diagonal and covariances on the off-diagonals)

$$\begin{pmatrix} \mathrm{var}(v_1) & \cdots & \mathrm{cov}(v_1, v_d) \\ \vdots & \ddots & \vdots \\ \mathrm{cov}(v_d, v_1) & \cdots & \mathrm{var}(v_d) \end{pmatrix}$$

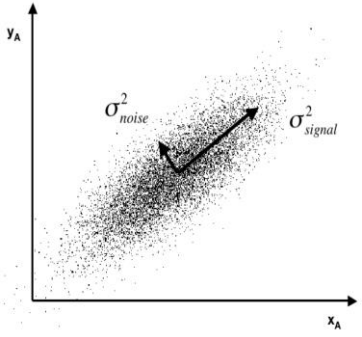
PCA: Toy example

- Toy example:
  - Try to understand the motion of a spring
  - Data from three camera sensors (x,y-position for each camera, i.e. 6-dim. data)

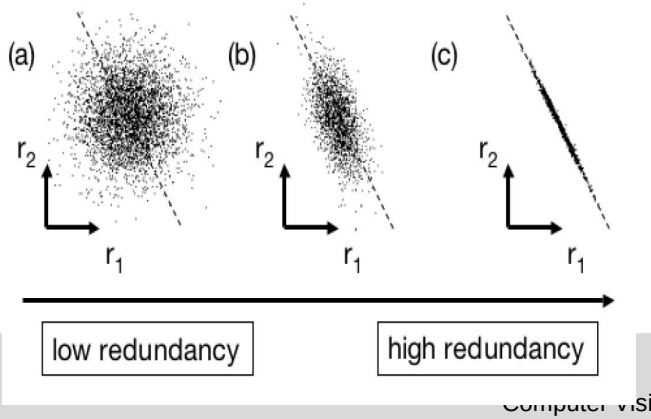


Noise and Redundancy

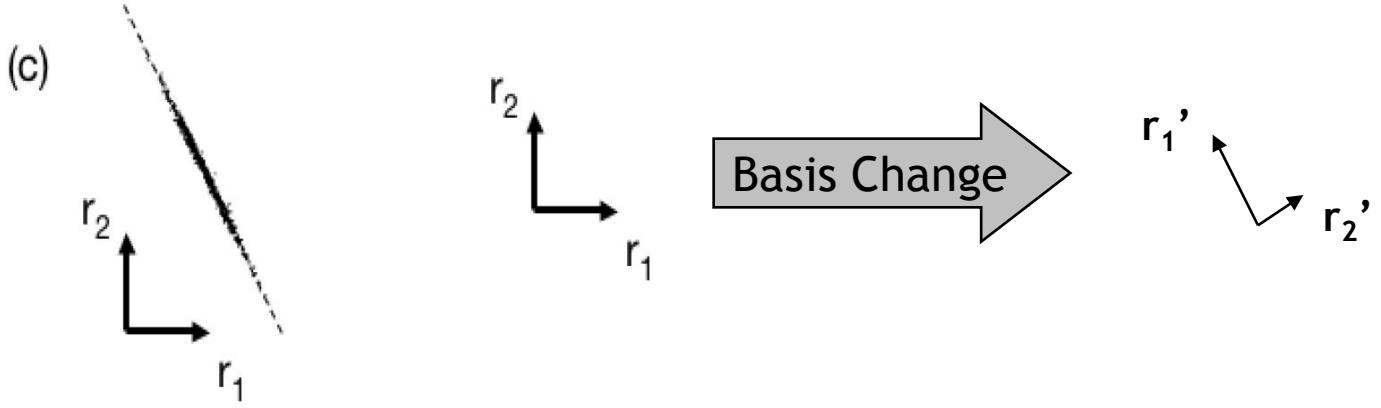
- Data contains noise



- Data is redundant
  - 1-dimensional motion vs. 6-dimensional sam
  - Sensor data from the three cameras are highly correlated



Change of basis



- By changing the basis, we may represent the data almost perfectly with a lower number of dimensions
- Remember that the data has a mean of zero

Change of Basis

- Basis change can be written as matrix multiplication
- Given the new basis vectors $p_1, \ldots, p_n$ we can transform data samples x_i in the following manner

$$PX = \begin{bmatrix} p_1 \\ \vdots \\ p_m \end{bmatrix} \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}$$

$$Y = \begin{bmatrix} p_1 \cdot x_1 & \cdots & p_1 \cdot x_n \\ \vdots & \ddots & \vdots \\ p_m \cdot x_1 & \cdots & p_m \cdot x_n \end{bmatrix}$$

- i.e. we are projecting x_i onto the new basis vect $y_i = \begin{bmatrix} p_1 \cdot x_i \\ \vdots \\ p_m \cdot x_i \end{bmatrix}$

Reducing the co-variances

- Remember the covariance matrix:

$$\begin{pmatrix} \mathrm{var}(v_1) & \cdots & \mathrm{cov}(v_1, v_n) \\ \vdots & \ddots & \vdots \\ \mathrm{cov}(v_n, v_1) & \cdots & \mathrm{var}(v_n) \end{pmatrix}$$

- We have seen that a basis change may reduce the correlation between the different dimensions
- Goal of PCA
  - Make covariance matrix as diagonal as possible

PCA Assumptions

- Basis (principal components) is orthogonal
- Linearity
  - Change of basis is a linear operation
  - For non-linear problems: kernel PCA
- Mean and variance are sufficient statistics
  - This holds if data is distributed according to a Gaussian distribution
- Large variances have important dynamics

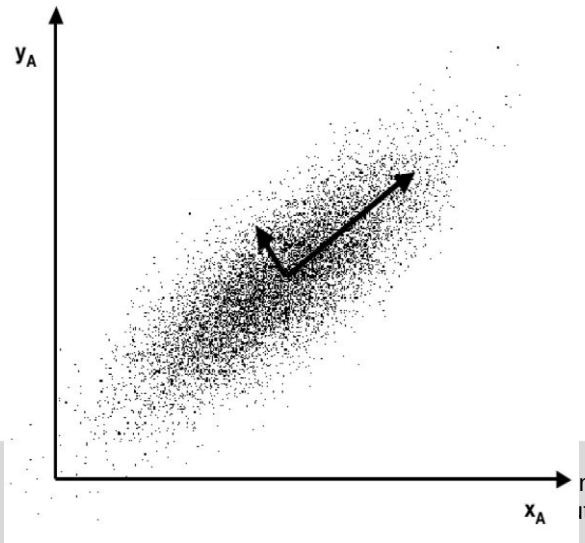
27

Finally solving it

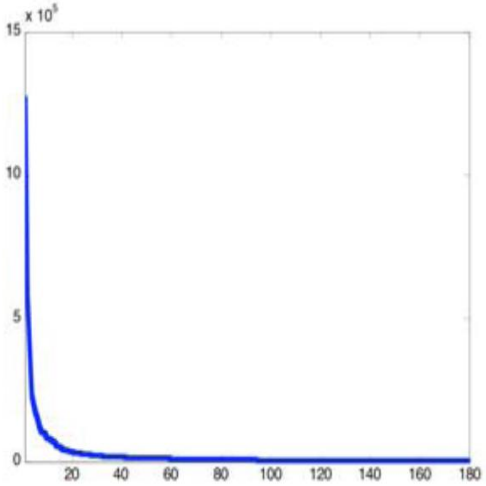
- Theorem:

  The covariance matrix is diagonalized by an orthogonal matrix of its eigenvectors

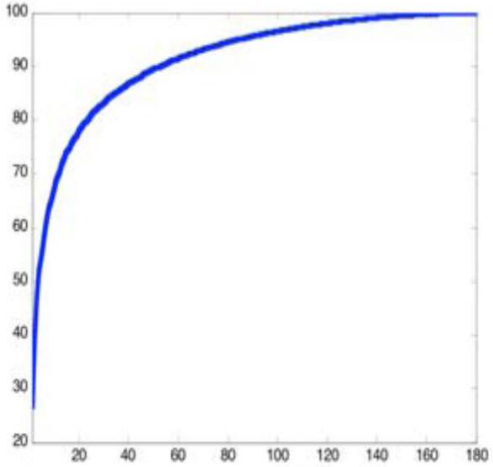
- That is the principal components of the underlying data are the eigenvectors
- The higher the eigenvalue the more variance is captured along the dimension



Eigenspectrum and Energy



eigenspectrum



energy: $en_k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{N} \lambda_i}$

Computing the eigen values and vectors

- Eigenvectors can be computed by solving the linear equation system

$$Ax = \lambda x$$

- This can be done with the well-known Gauss-Algorithm

$$(A - \lambda I)x = 0$$

- In practice, we most often use a singular value decomposition (SVD)

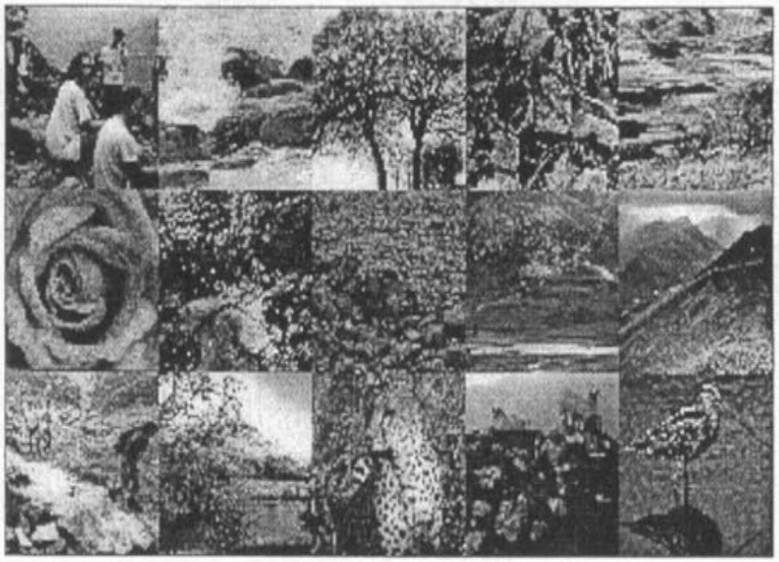
PCA in practice

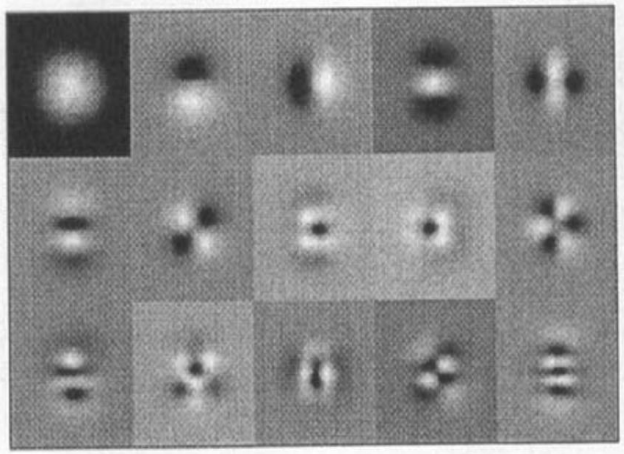
- The covariance of a set of images can be computed by writing the image data as columns into a matrix and multiplying it with it's transposed matrix

$$Cov = \frac{1}{N-1} AA^T$$

- If the data samples are high dimensional, the covariance matrix can get extremely large
  - Let the sample x_i be images with a resolution of 1600x1200 then the covariance would be a 1920000x1920000 matrix
  - Computationally, computing the eigenvectors would be very costly

Principal Components of natural images





Hancock et al:

33

Example: Face Images

Principal components are called “eigenfaces” and they span the “face space”

→ See Lecture on face recognition !

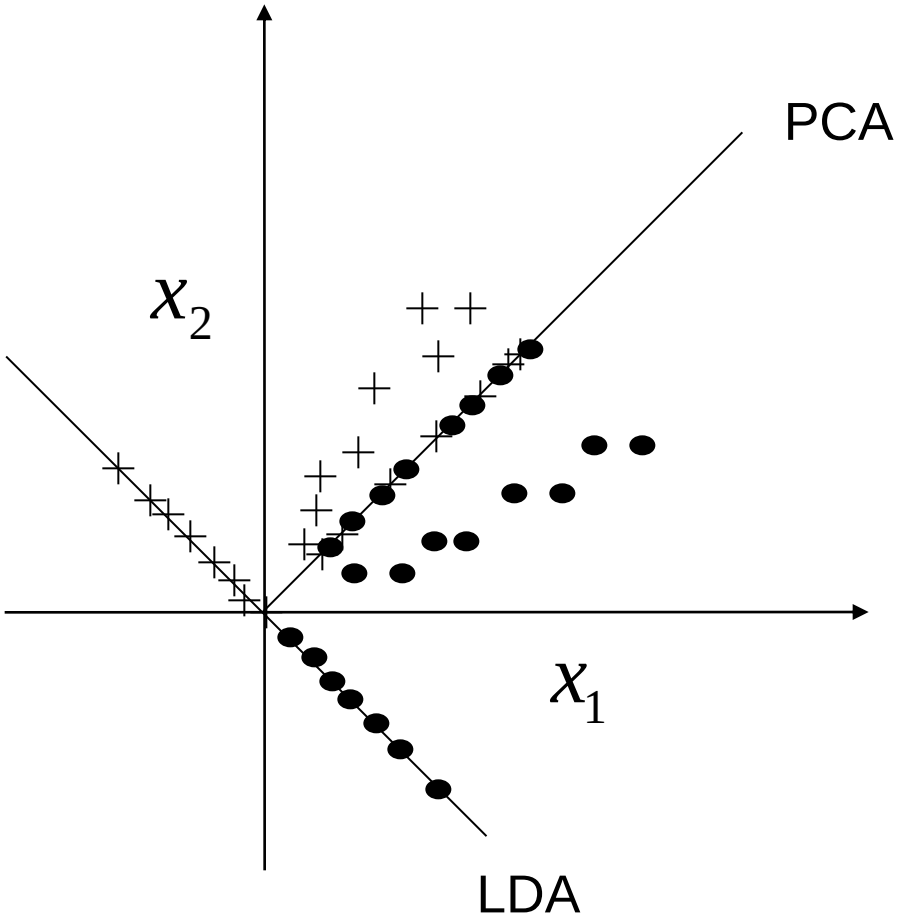
34

Summary PCA

- can be used to reduce the number of dimensions
- Find the principal dimensions
  - Principal dimensions are assumed to have a high variance
  - discard non-informative dimensions (redundancy and noise)
- Extensions:
  - Robust PCA (deal with occlusion)
  - Incremental PCA
  - Kernel PCA (non-linear)

Dimensionality reduction

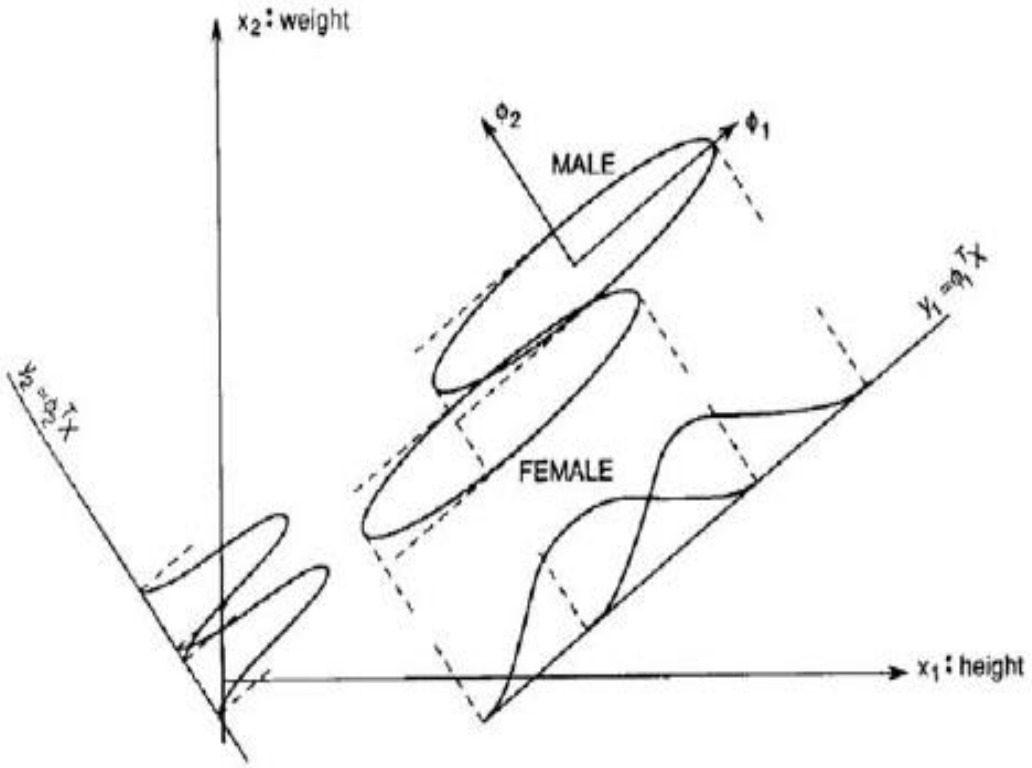
- LDA: Maximize class separability



(→ see Duda & Hart)

Motivation: Fisher linear discriminant

- more on



Bayesian Classification

- We are given a feature vector **x** and want to know which class ω_i is most likely, given **x**
- Use Bayes' rule:

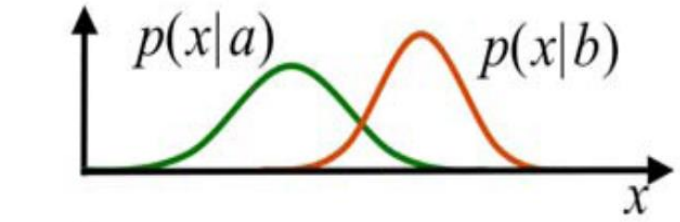
$$P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i) P(\omega_i)}{p(\mathbf{x})}$$

$$\text{with } p(\mathbf{x}) = \sum_i p(\mathbf{x} \mid \omega_i) P(\omega_i)$$

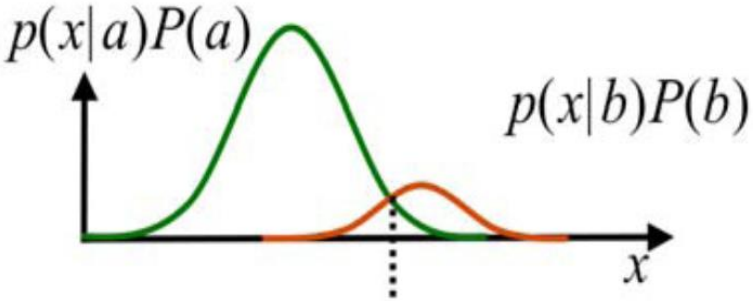
$$\Rightarrow \quad \text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{normalization factor}}$$

- Decide for the class ω_i with maximum posterior probability
- If we know $p(\mathbf{x}|\omega_i)$ and $P(\omega_i)$, then we can just compute the probabilities
- It can be shown that this is the optimal solution
- Problem: $p(\mathbf{x}|\omega_i)$ (and to a lesser degree $P(\omega_i)$) is usually unknown and often hard to estimate from data
- Priors describe what we know about the classes before observing anything
  - Can be used to model prior knowledge
  - Sometimes easy to estimate (counting)

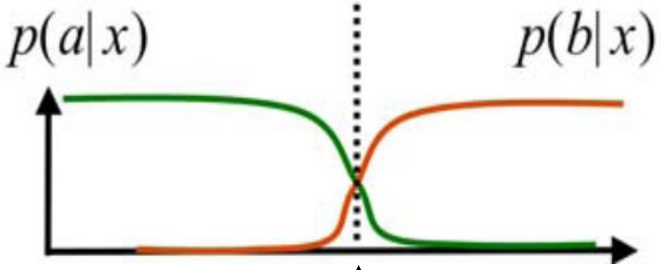
Example



Likelihood



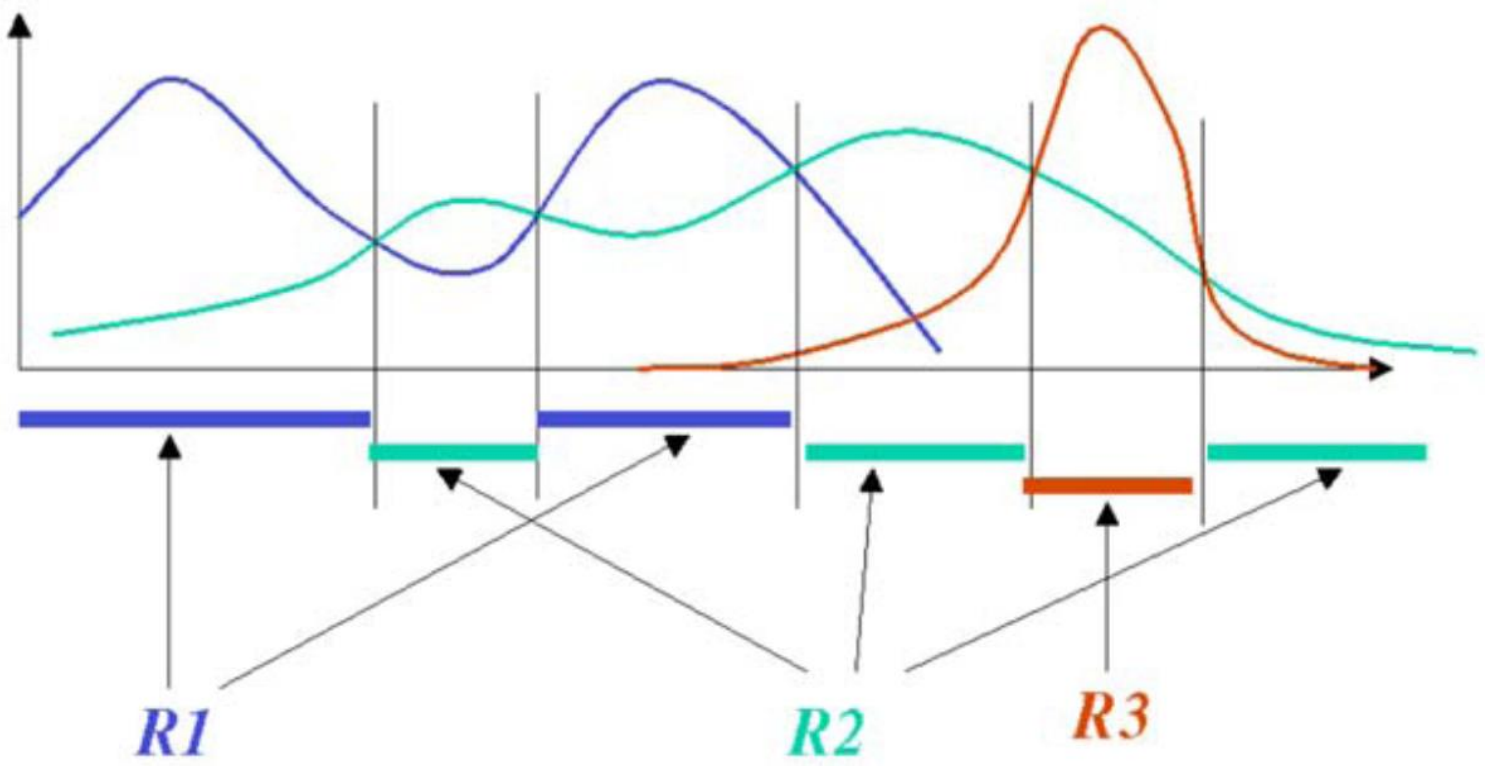
$Likelihood \times Prior$



$$Posterior = \frac{Likelihood \times Prior}{Normalization\ factor}$$

Decision boundary

Example with more classes



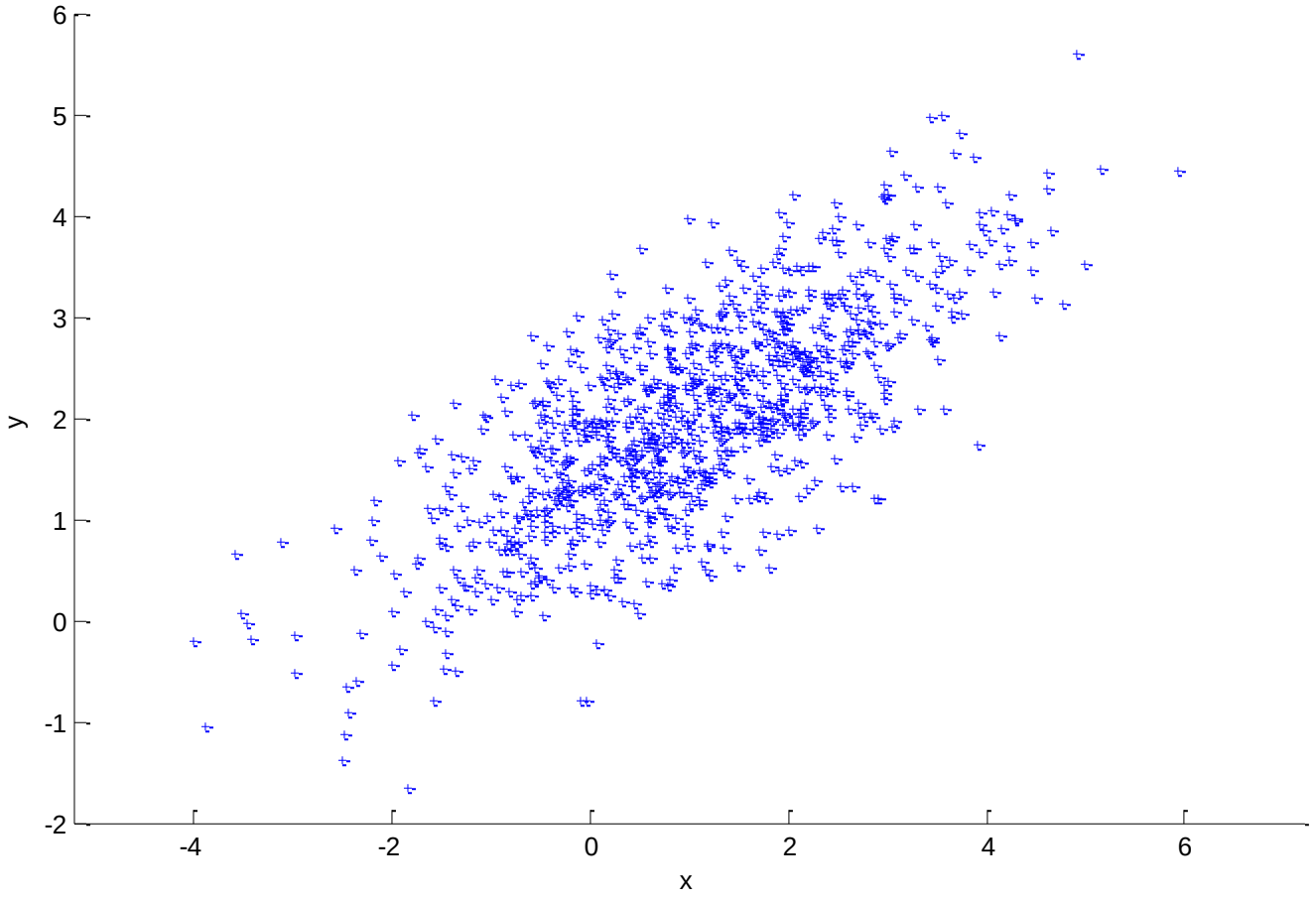
Gaussian classification

- Assumption: $p(\mathbf{x}|\omega_i) \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}) =$ $\frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$
- This makes estimation easier:
  - Only mu and sigma have to be estimated
  - To reduce parameters, the covariance matrix can be restricted
    - Diagonal matrix ➔ Dimensions uncorrelated
    - Multiple of unit matrix ➔ Dimensions uncorrelated with same variance

- Problem: if the assumption(s) do not hold, the model does not represent reality well (→ performance will decrease)

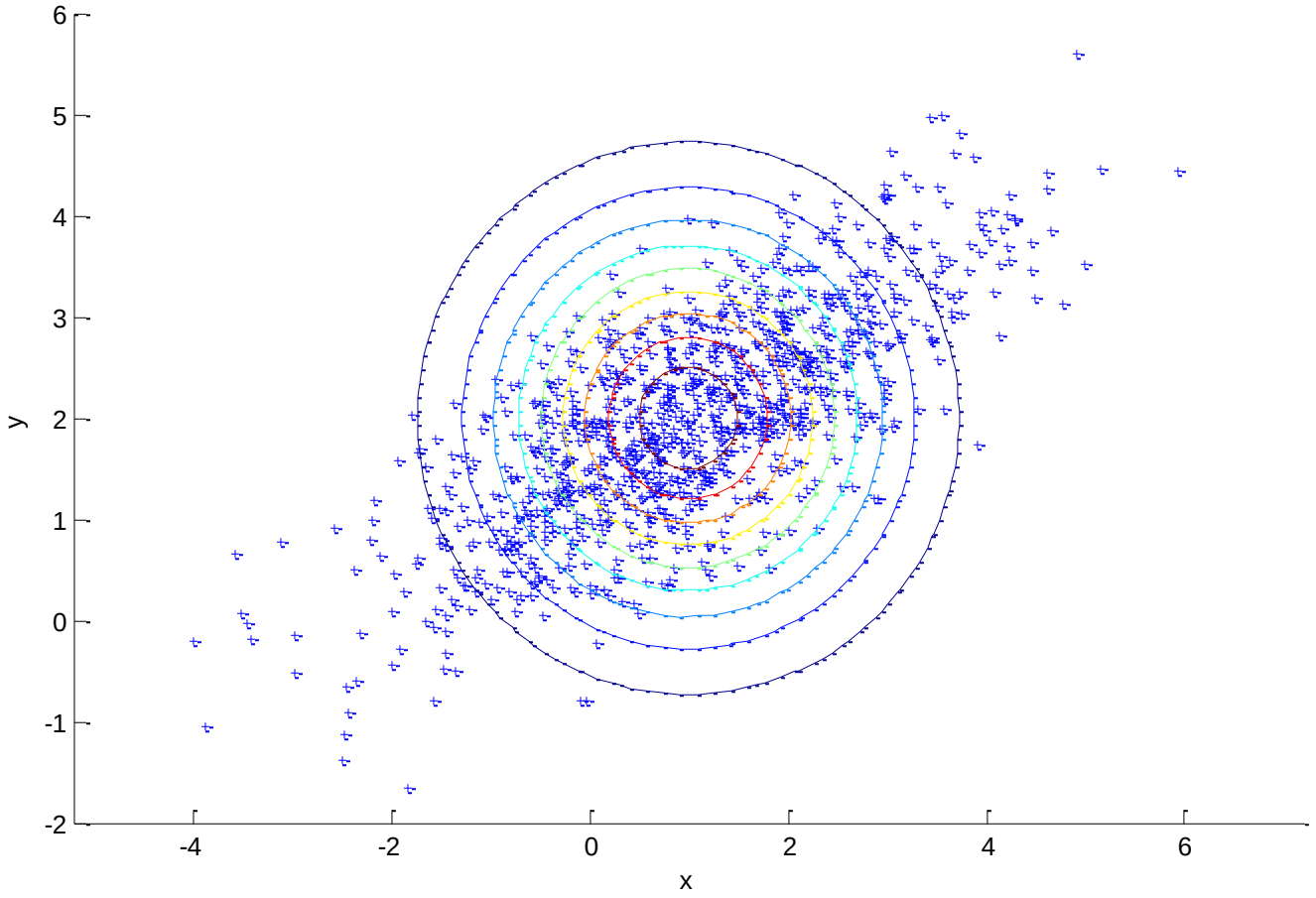
- Estimation of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ with Maximum Likelihood (ML)
  - Use parameters, that best explain the data (highest likelihood):
    $l(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = p(\text{data} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = p(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$
    $= p(\mathbf{x}_0 \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \cdot p(\mathbf{x}_1 \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \cdot \ldots \cdot p(\mathbf{x}_n \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$
  - $\log(l(\boldsymbol{\mu}, \boldsymbol{\Sigma})) = \log(p(\mathbf{x}_0 \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})) + \ldots + \log(p(\mathbf{x}_n \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}))$
  - Maximize $\log(l(\boldsymbol{\mu}, \boldsymbol{\Sigma}))$ over $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$

Gaussian Example



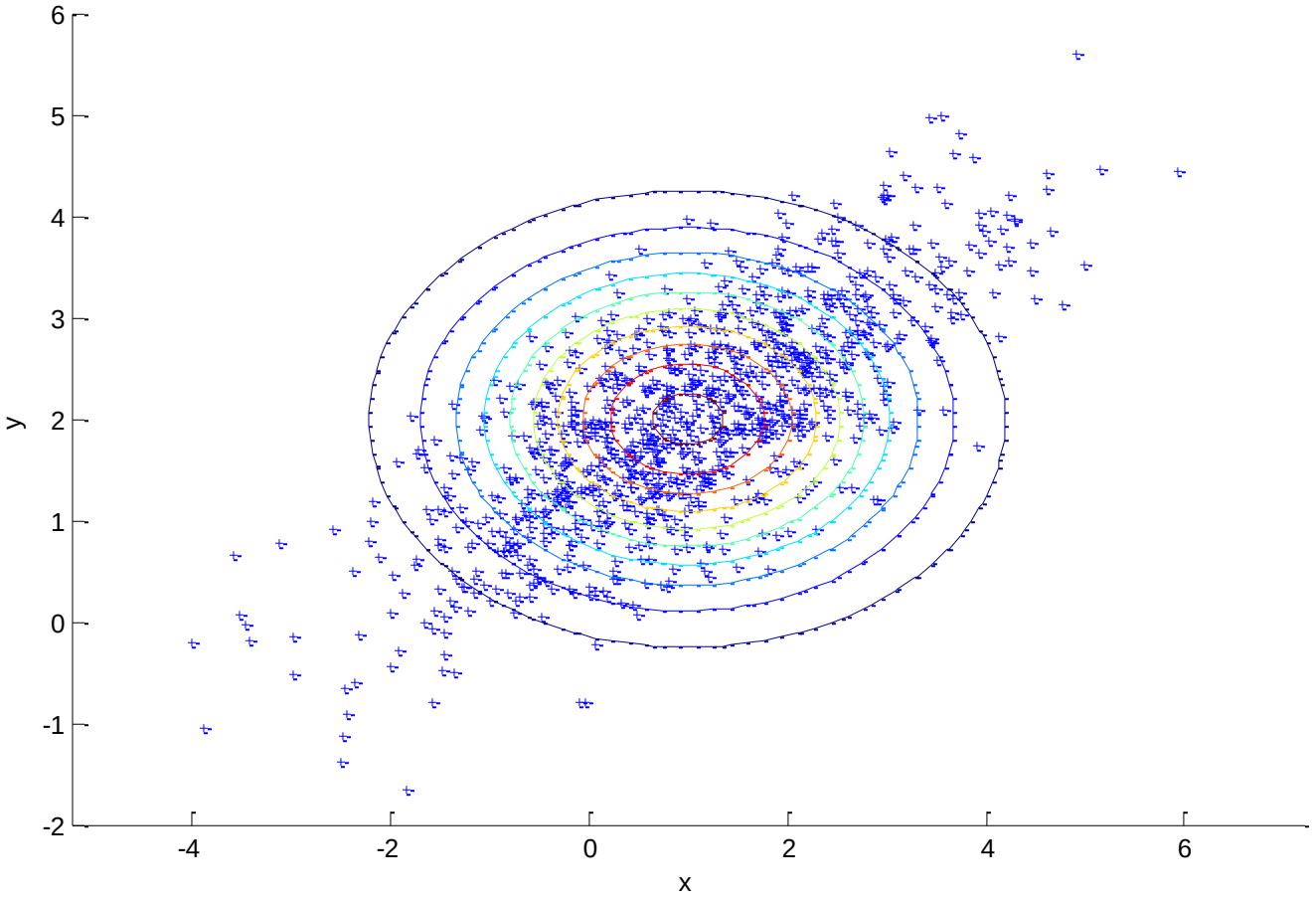
Random samples drawn from a Gaussian distribution

Gaussian Example



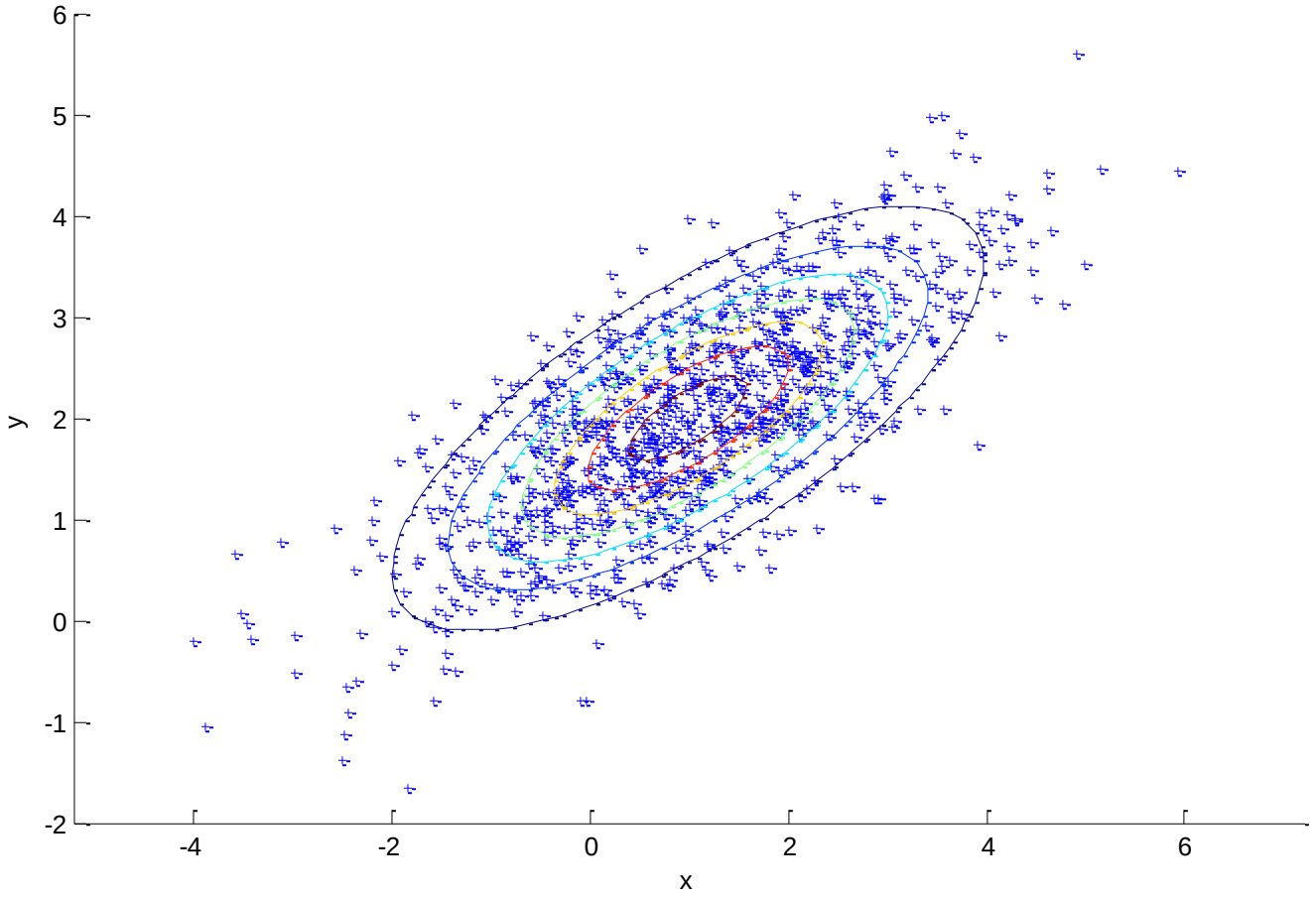
Estimation of Gaussian with multiple of unit covariance matri

Gaussian Example



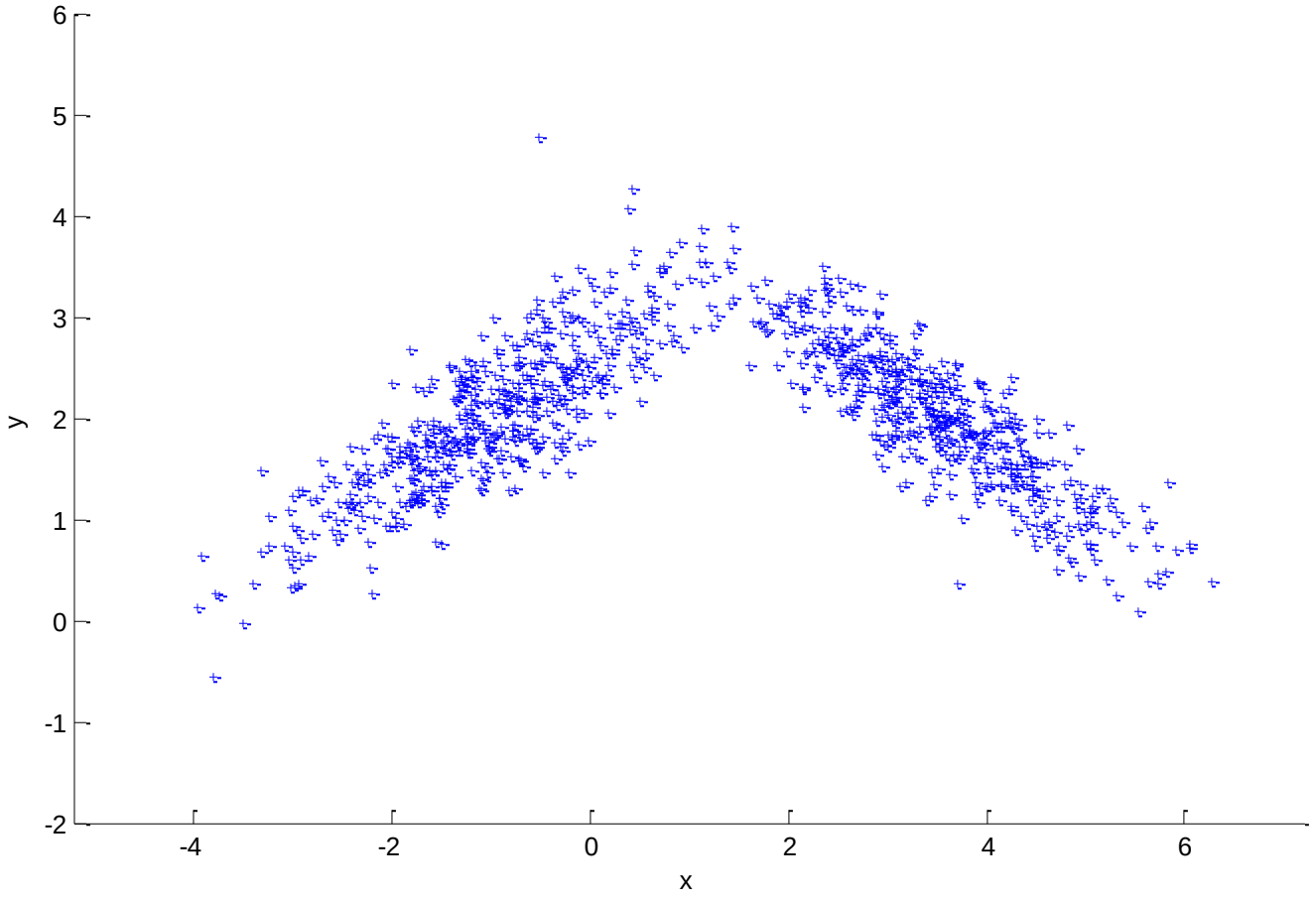
Estimation of Gaussian with diagonal covariance matrix

Gaussian Example



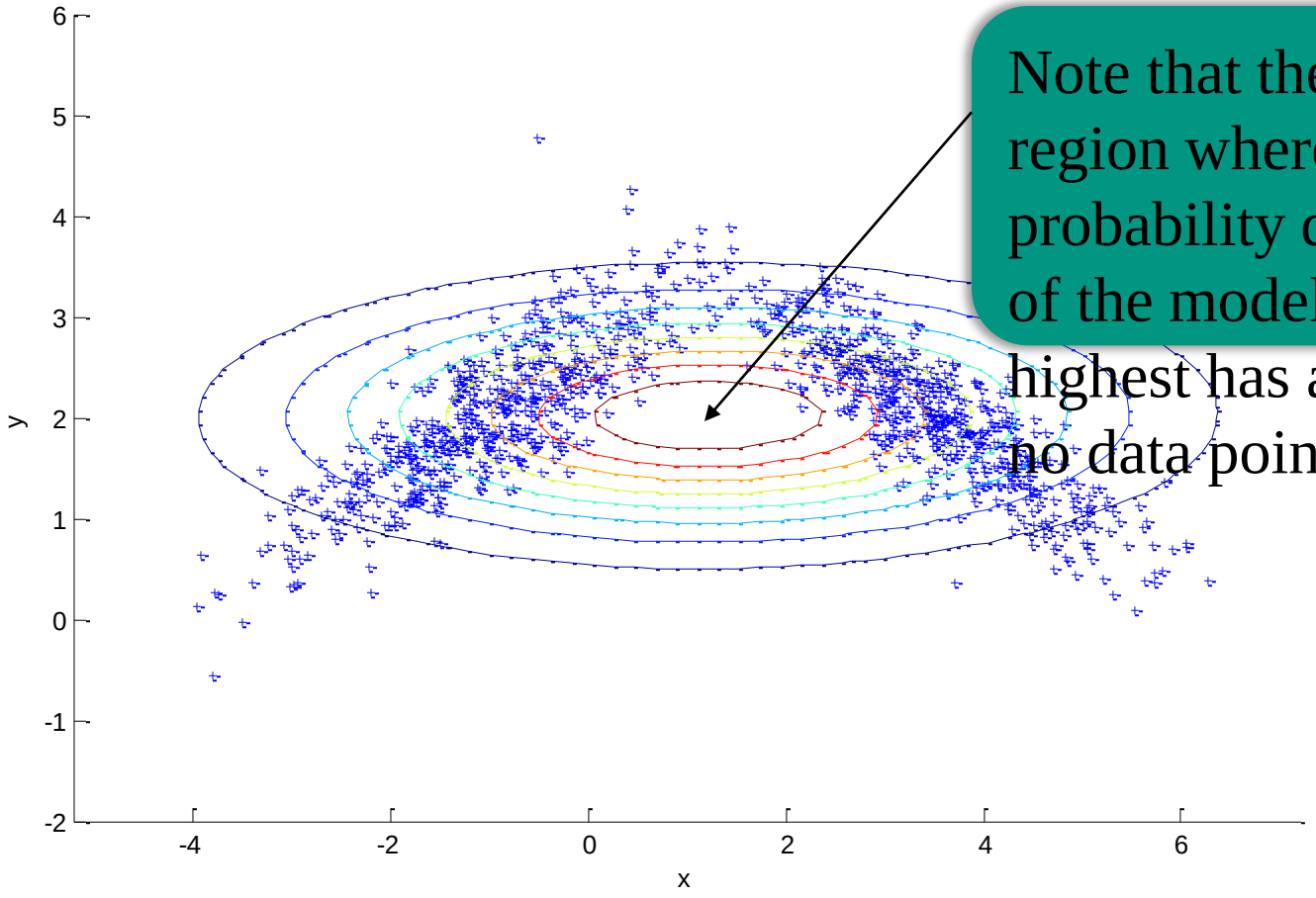
Estimation of Gaussian with full covariance matrix

Gaussian Example



Random samples drawn from non-Gaussian distribution

Gaussian Example



Note that the region where the probability density of the model is highest has almost no data points!

Estimation of Gaussian with full covariance matrix

Gaussian Mixture Models (GMMs)

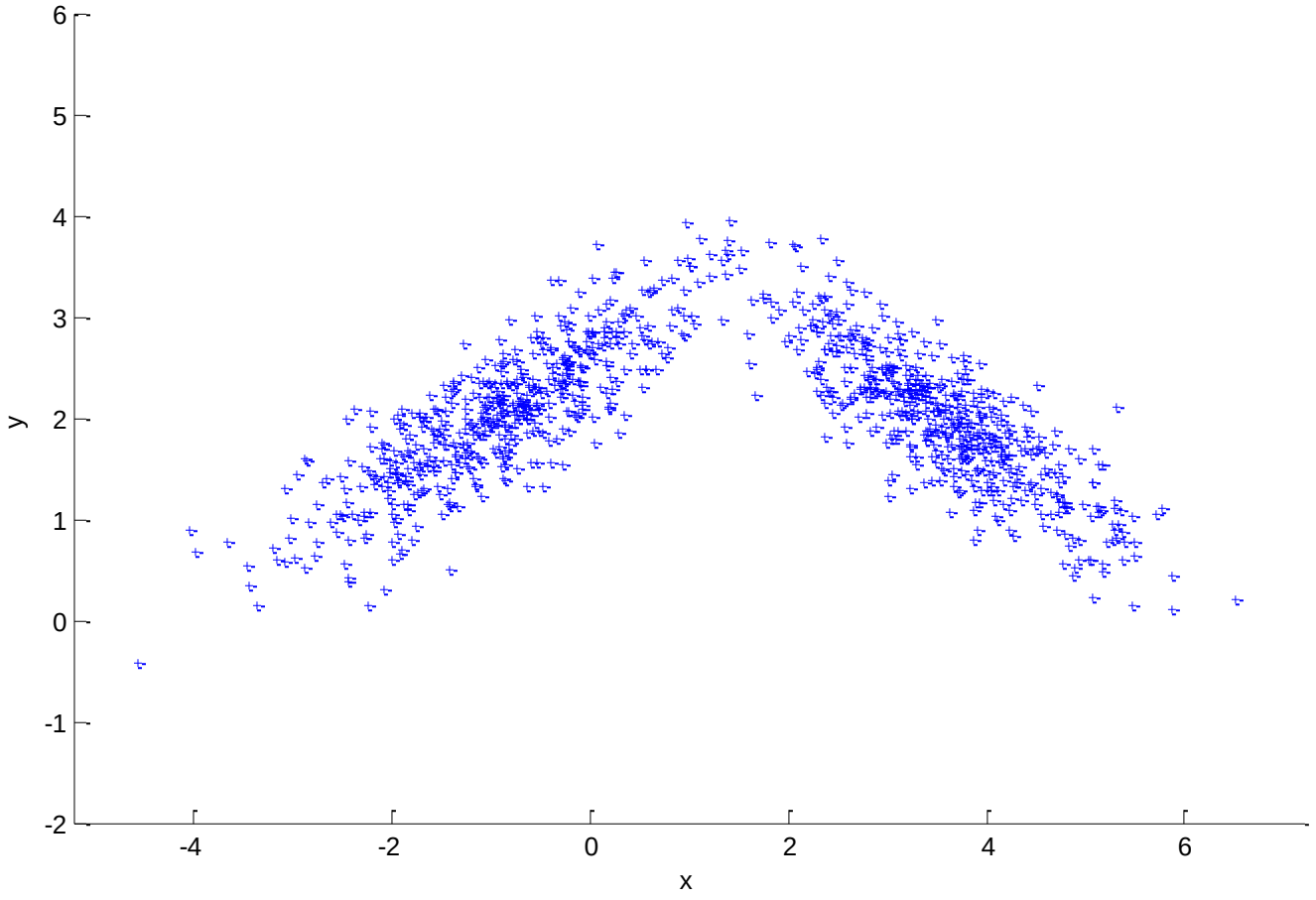
- Approximate true density function using a weighted sum of several Gaussians

$$p(\mathbf{x}) = \sum_i w_i \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$$

$$\text{with } \sum_i w_i = 1$$

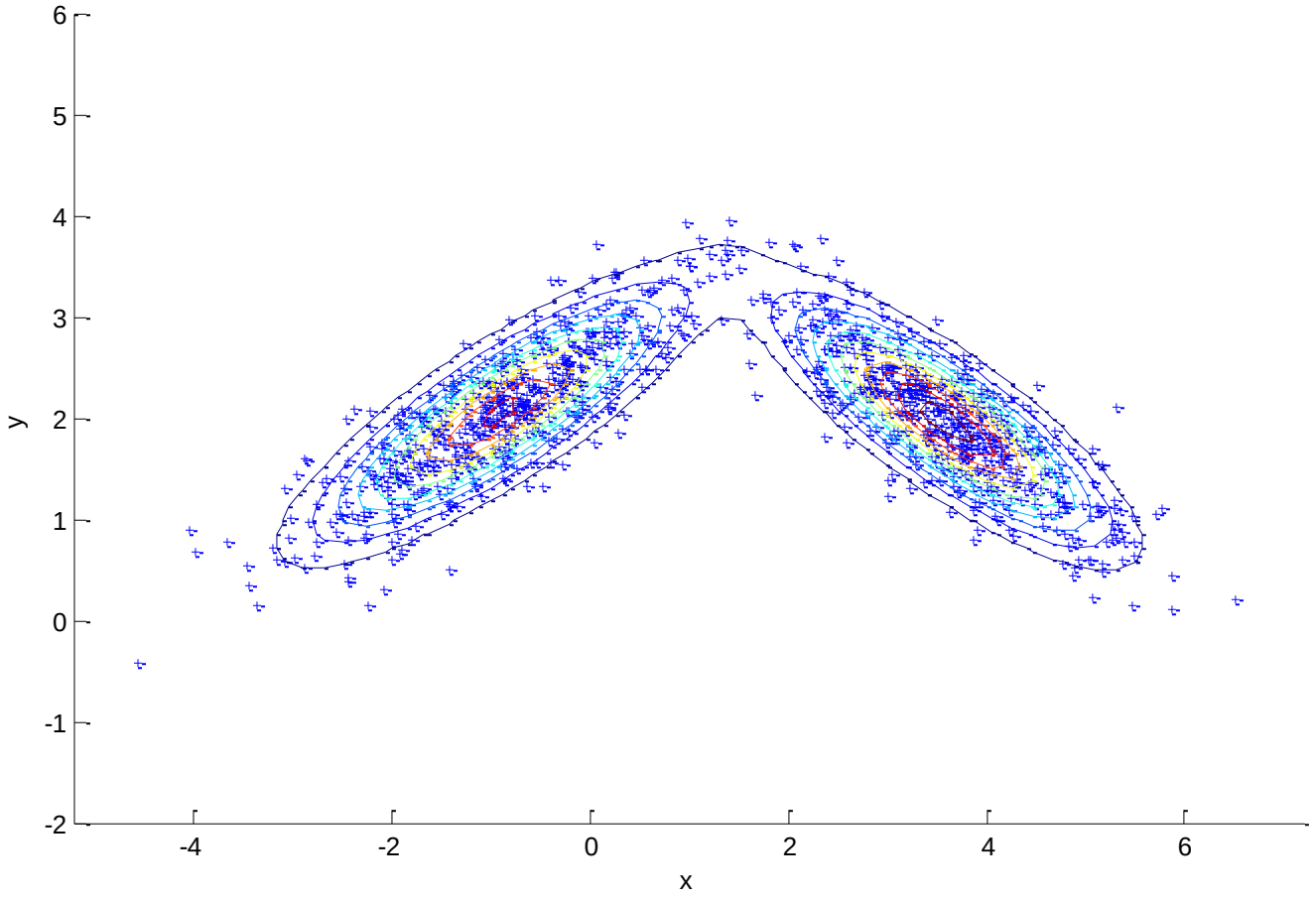
- Any density can be approximated this way with arbitrary precision
  - But might need many Gaussians
  - ➔ Difficult to estimate many parameters
    - Restrict covariance matrices as before
    - Only one shared covariance matrix for all Gaussians
- Now need to estimate parameters of the Gauss functions as well as the weights
  - Expectation Maximization (EM) Algorithm

GMM example



Random samples drawn from non-Gaussian distribution

GMM example



Estimated GMM with full covariance matrices

Expectation Maximization (EM)

- If we knew to which Gaussian each data point belongs, this would be simple
  ➔ We don't exactly know, but we can estimate
- EM Algorithm
  - Initialize parameters of GMM randomly
  - Repeat until convergence
    - Expectation (E) step:
      Compute the probability p_{ij} that data point i belongs to Gaussian j
      - Take the value of each Gaussian at point i and normalize so they sum up to one
    - Maximization (M) step:
      Compute new GMM parameters using soft assignments p_{ij}
      - Maximum Likelihood with data weighted according to p_{ij}

GMM applets

- http://www.socr.ucla.edu/Applets.dir/MixtureEM.html
- http://www.the-wabe.com/notebook/em-applet.html

Some taxonomy: parametric vs. non-parametric?

- Gaussian and GMMs are called parametric classifiers
  - They assume a specific form of probability distribution with some parameters
  - Then only the parameters need to be estimated
- There are also methods which do not assume a specific form of probability distribution
  - They are called non-parametric
  - Examples: Parzen windows, k-nearest neighbors
- Properties of parametric classifiers
  - + Need less training data because less parameters to estimate
  - - Only work well if model fits data
- Properties of non-parametric classifiers
  - + Work well for all types of distributions
  - - Need more data to correctly estimate distribution

Some taxonomy: generative vs. discriminative

- A method that models $P(\omega_i)$ and $p(\mathbf{x}|\omega_i)$ explicitly is called a generative model
  - $p(\mathbf{x}|\omega_i)$ allows to generate new samples of class ω_i
- The other common approach is called discriminative models
  - They directly model $P(\omega_i|\mathbf{x})$ or just output a decision ω_i given an input pattern $\mathbf{x}$
- Often, discriminative models are easier to train because they solve a simpler problem

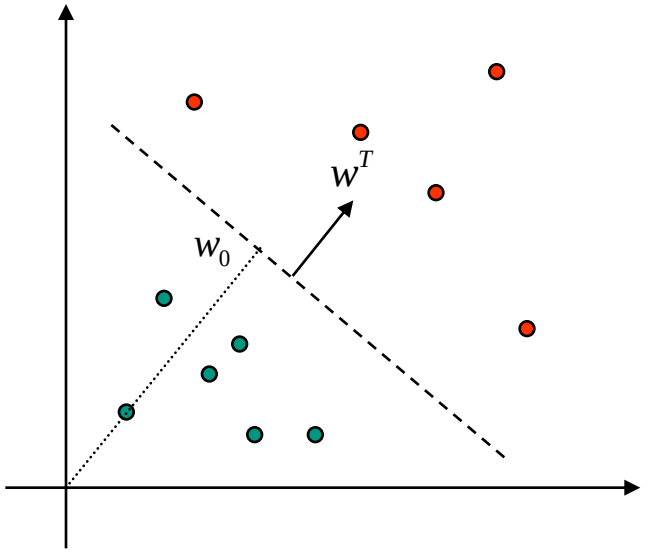
Linear Discriminant Functions

- Separate two classes with a linear hyper plane

$$y(x) = w^T x + w_0$$

- With w^T the normal vector of the hyper plane

- Examples:
  - Perceptron
  - Linear SVM



57

Perceptron Algorithmus[Rosenblatt ’58]

- Preconditions:
  - Data set is linear separable
- Goal
  - Find a separating hyper plane
- Idea
  - Iteratively improve solution
  - Only update solution if the data sample of interest is classified incorrectly

Perceptron Algorithm

1. Initialize $w = 0$

2. Classify a new data sample with the following inner product $y(x) = \text{sign}(w^T x)$

3. If correct, then goto step 2
   else and $w = w - y(x)x$

4. If no errors left, then done
   else goto step 2

Why does this work?

- Given misclassified sample x with y(x)=1
- Then: w = w - x
- If we now classify the sample again:
  - $y(x) = (w-x)^T x = w^T x – x^T x$
  - $x^T x$ is positive and therefore the next prediction for this data sample will be closer to the correct value

- Convergence:
  - Does not converge if data is not separable

Wait

- We did not consider w_0
- What happens if separating hyper plane does not pass through (0,0)
- Small trick:
  - Choose x to be n+1 dimensional with n+1 dimension always being 1, then

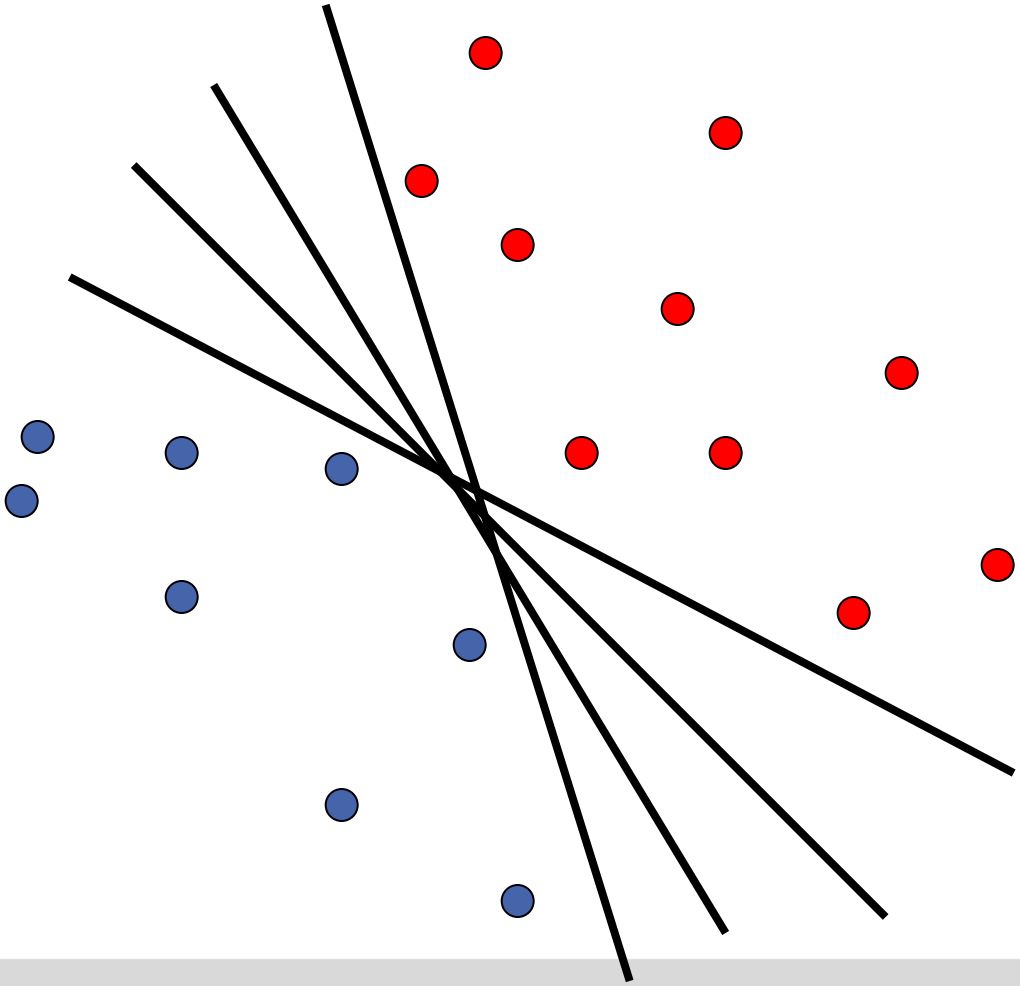
$$y(x) = w^T x + w_0$$

  - Becomes

$$y(x) = w^T x$$

Which hyperplane is best?

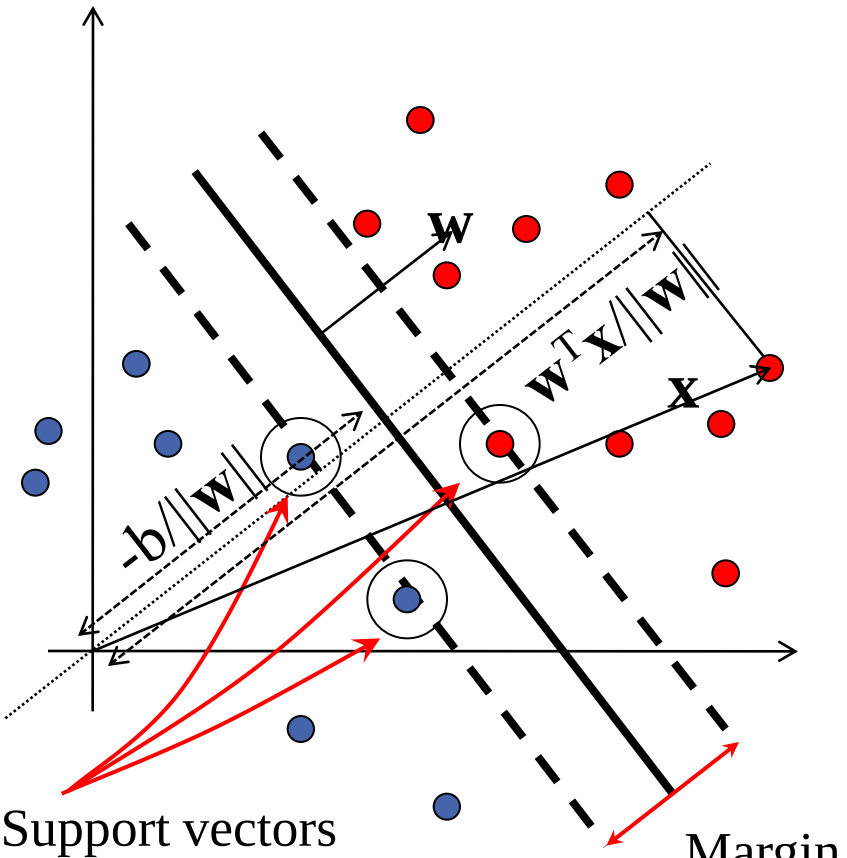
- Perceptron algorithm finds any hyperplane that separates the data



Which hyperplane is best?

Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



$\mathbf{w}^T\mathbf{x}$ / $\|\mathbf{w}\|$ → $\mathbf{x}$ projected in direction $\mathbf{w}$

On which side of the hyperplane is $\mathbf{x}$?
$\mathbf{w}^T\mathbf{x}$ / $\|\mathbf{w}\|$ <> -b / $\|\mathbf{w}\|$ ➔ $\mathbf{w}^T\mathbf{x}$ <> -b

$y_i = 1$ → $\mathbf{w}^T\mathbf{x}_i > b$ $\quad y_i = -1$ → $\mathbf{w}^T\mathbf{x}_i < b$
➔ $y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0$

Length of $\mathbf{w}$ does not change anything
Fix it by requiring:
$\min y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$

Distance to hyperplane?
$|\mathbf{w}^T\mathbf{x} + b| / \|\mathbf{w}\|$

Minimal distance to hyperplane?
$1 / \|\mathbf{w}\|$

Margin?
$2 / \|\mathbf{w}\|$

65

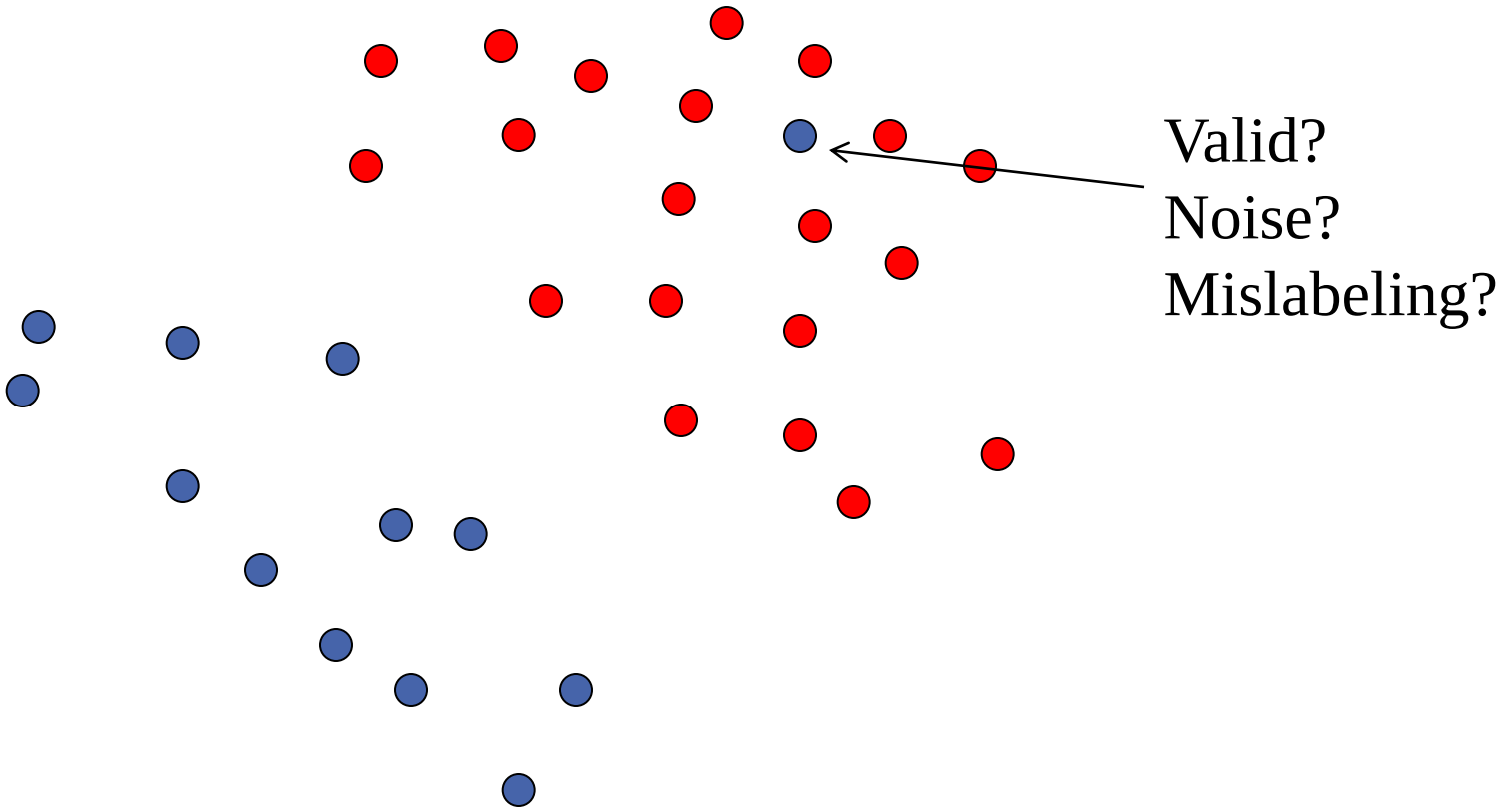
Finding the hyperplane with maximum margin

- Pose as constrained optimization problem:
  - Minimize $||\mathbf{w}||^2$ (i.e. maximize margin 2 / $||\mathbf{w}||$)
  - Subject to $y_i(\mathbf{w}^T\mathbf{x}_i + b) >= 1$
    - Samples are on the right side of the hyperplane (> 0)
    - Samples are outside the margin (>= 1)

- This is known as a quadratic optimization problem
  - Has only one global minimum
    - Very nice, no problems with local minima!
  - Efficient algorithms for solving it are known
    - Optimization using Lagrange multipliers

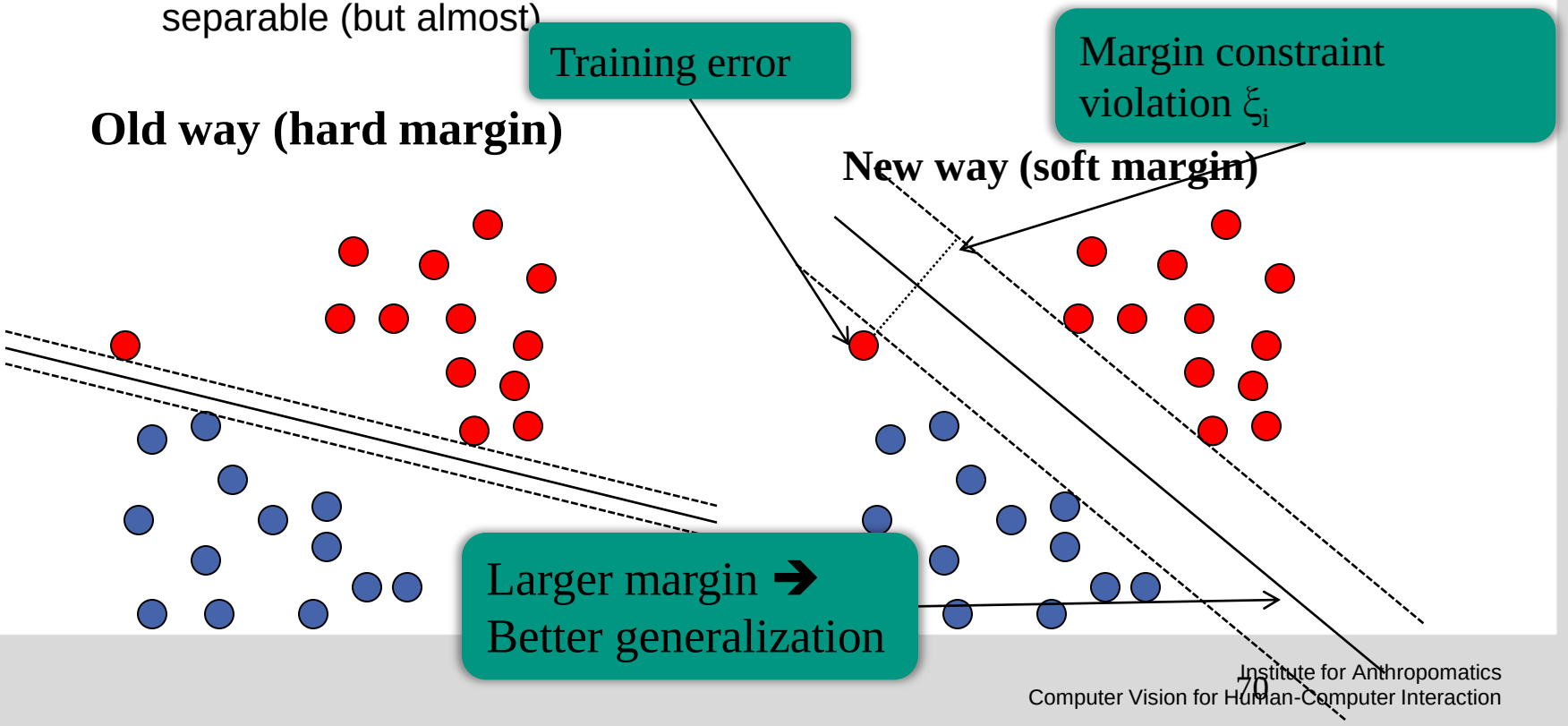
Non-separable data

- What to do with data that is not linearly separable?



Large margin vs. training error

- Noise in data or labels can make training data non-separable
  - Or seriously reduce the size of the margin
- What we want:
  - A way to maximize margin while ignoring (some) outliers, that would lead to small margin
  - This gives us an SVM algorithm that also works for data that is not linearly separable (but almost)



70

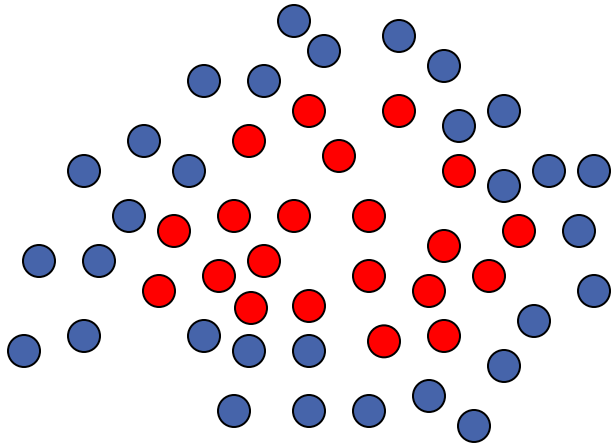
Soft-margin SVM formulation

- Reminder: hard-margin SVM
  - Minimize $||\mathbf{w}||^2$ (i.e. maximize margin 2 / $||\mathbf{w}||$)
  - Subject to $y_i(\mathbf{w}^T\mathbf{x}_i + b) >= 1$
- Soft-margin SVM
  - Add sum of constraint violations to objective function:
    Minimize $||\mathbf{w}||^2 + C \cdot \Sigma\, \xi_i$
  - Allow constraints to be violated:
    Subject to $y_i(\mathbf{w}^T\mathbf{x}_i + b) >= 1 - \xi_i$ (with $\xi_i >= 0$)
  - Parameter C controls the trade-off between low training error and large margin
- ξ_i has simple interpretation
  - $\xi_i = 0$ ➔ sample classified correctly outside margin
  - $0 < \xi_i < 1$ ➔ sample classified correctly inside margin
  - $\xi_i > 1$ ➔ sample classified incorrectly
  - $\Sigma\, \xi_i$ is upper bound on number of training errors

71

Really non-separable data

- What about data that is really non-separable (not because of noise)?

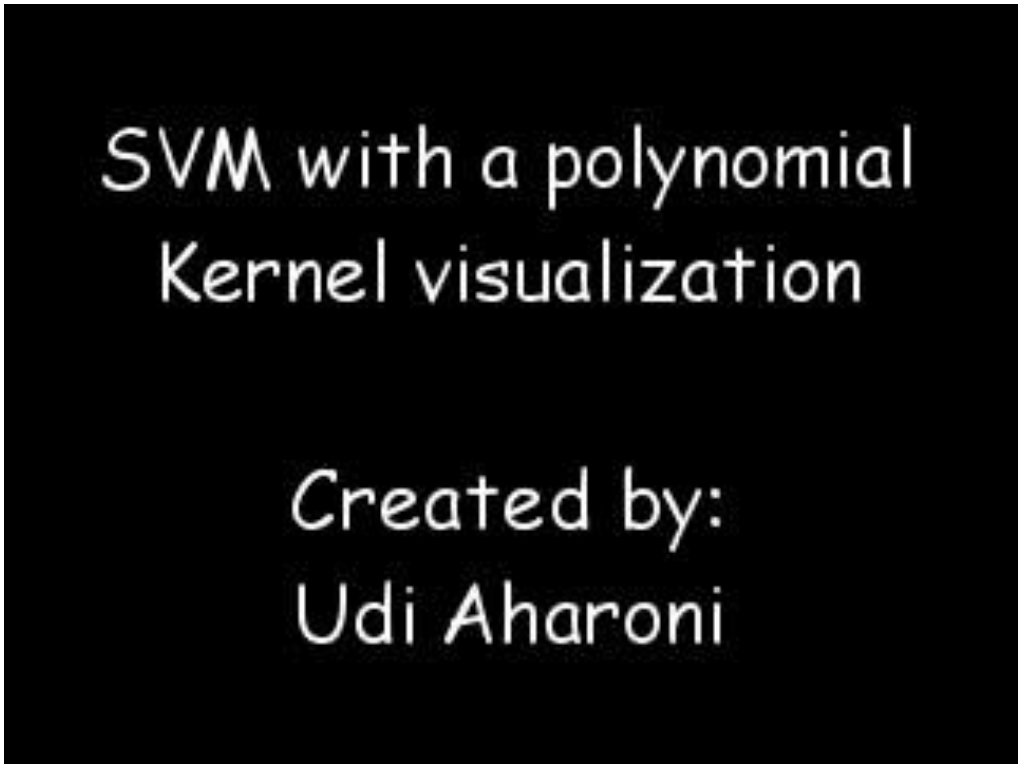


No hyperplane will be able to perform well On this kind of data!

Non-linear SVMs: Going to higher dimensions

- Idea: Transform data to high-dimensional space where it **is** linearly separable

$$\Phi : \mathbf{R}^d \mapsto \mathcal{H}.$$



http://www.youtube.com/watch?v=3liCbRZPrZA

Kernel trick

- Transforming feature vectors into a high dimensional space can be difficult to compute
  - For infinite-dimensional spaces it is impossible

$$\Phi : \mathbf{R}^d \mapsto \mathcal{H}.$$

- Kernel trick:
  - Re-write algorithm so that the feature vectors only appear in dot products *(using Lagrange multipliers / dual optimization problem)*

$$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

  - Use kernel function to directly compute dot product in high-dimensional space from low-dimensional input vectors

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

  - High-dimensional space becomes implicit property of the kernel function
  - *Nothing needs to be computed in high-dimensional space !*
    - Even infinite-dimensional spaces are possible

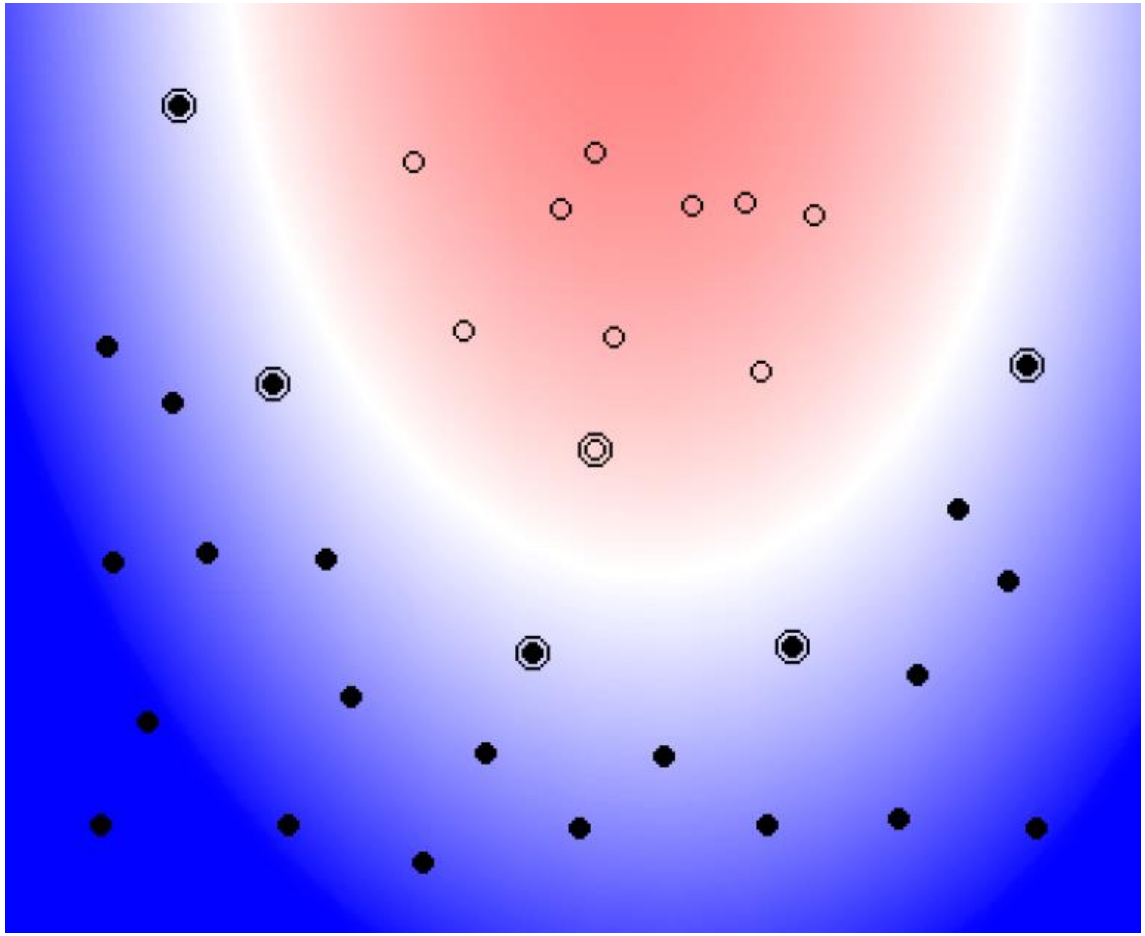
(for mathematical details: See e.g. Burges 1998, or Machine-Learning lecture)

Common SVM kernels

- Some kernel functions are widely used
  - Linear kernel: $k(x, x') = x^T x'$
  - Polynomial kernel: $k(x, x') = (x^T x' + 1)^d$
  - Gaussian (RBF) kernel: $k(x, x') = \exp(-\|\mathbf{x}-\mathbf{x}'\|^2/2\sigma^2)$
  - Sigmoid kernel: $k(x, x') = \tanh(\mathbf{x}^T\mathbf{x}' + c)$
- Kernels for non-vectorial data are also possible
  - Graphs, sets, texts, etc…
- Kernel defines a similarity function for the input vectors
  - Often problem-specific kernels can improve performance

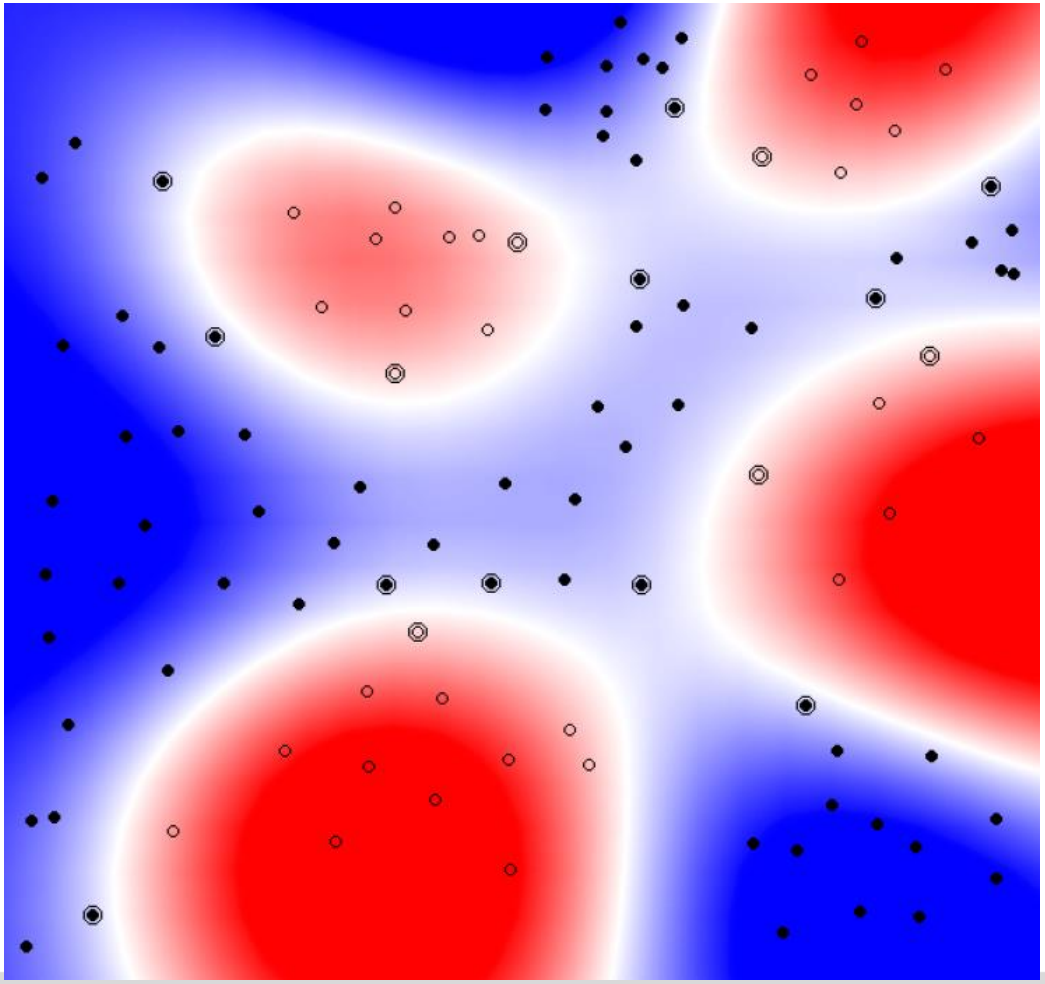
Kernel SVM example

Polynomial kernel (degree 2): $(\mathbf{x}^T\mathbf{x}' + 1)^2$



Kernel SVM example

RBF-Kernel: $\exp(-\|\mathbf{x}-\mathbf{x}'\|^2 / 2\sigma^2)$



Model selection

- SVM learning algorithm has parameter C, Kernels usually have parameter(s)
  - These need to be optimized to achieve best performance
- Simple idea: Training several SVMs with different parameters and taking the one with the best performance on the training set
  - But training error is not a good performance measure
  - What we want is good generalization capability (low test error)
- → n-fold cross-validation (CV)
  - Split training set into n folds
  - For each fold i, train SVM using all folds except fold I
- Perform CV for a number of SVM parameters and use the ones with best performance

Linear SVMs

- Don’t completely forget about linear SVMs!
- If the input space is already high-dimensional, linear SVMs can often perform well too

- Linear SVMs have some advantages
  - Speed: Only one scalar product for classification
    - Kernel SVM needs #(support vectors) kernel evaluations
  - Memory: Only one vector **w** needs to be stored
    - Kernel SVM needs to store all support vectors
  - Training: Training is much faster
    - Specialized primal optimization algorithms
  - Model selection: Only one parameter to optimize
    - Kernel SVMs need to optimize C and kernel parameters

Multi-class SVMs

- SVM is originally a two-class classifier
- Generic methods to get a multi-class classifier
  - One-vs-all: One SVM per class
  - One-vs-one: Pairwise SVMs
  - Problems:
    - Classification sometimes ambiguous
    - Need to train and evaluate multiple classifiers
- Today real multi-class SVMs are available
  - Weston & Watkins 1998
  - Crammer & Singer 2001

Some practical tips for using SVMs

- Normalization of feature vectors very important
  - Normalize each feature separately
    - Zero mean, unit variance or [-1; 1] are popular
  - Normalize feature vector to unit norm
    - $\mathbf{x}' = \mathbf{x} / ||\mathbf{x}||$
- Otherwise performance will suffer
  - Training will be much slower and numerically unstable
  - Classification performance will be suboptimal
- Model selection, i.e. optimization of C and kernel parameters also very important
  - Generally grid-search using cross-validation is used

SVM software

- LibSVM [Chang et al. 2001]
  - http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- SVMlight [Joachims 1999]
  - http://svmlight.joachims.org/
- Applets
  - http://www.smartlab.dibe.unige.it/Files/sw/Applet%20SVM/svmapplet.html
  - http://svm.dcs.rhbnc.ac.uk/pagesnew/GPat.shtml

Instance-Based Learning

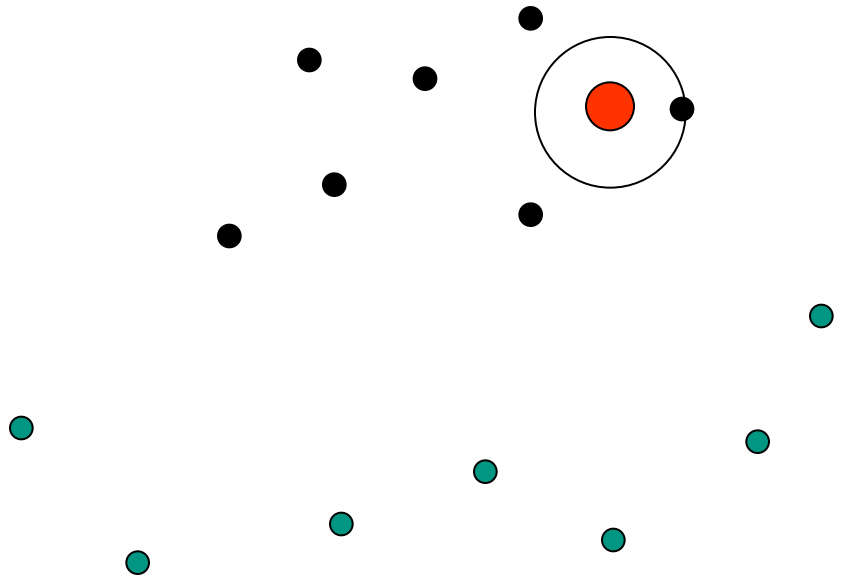
Instance-based Learning

- Learning=storing all training instances
- Classification=assigning target function to a new instance
- Referred to as “Lazy” learning Instance
- Examples:
  - Template-Matching
  - K-Nearest Neighbor

84

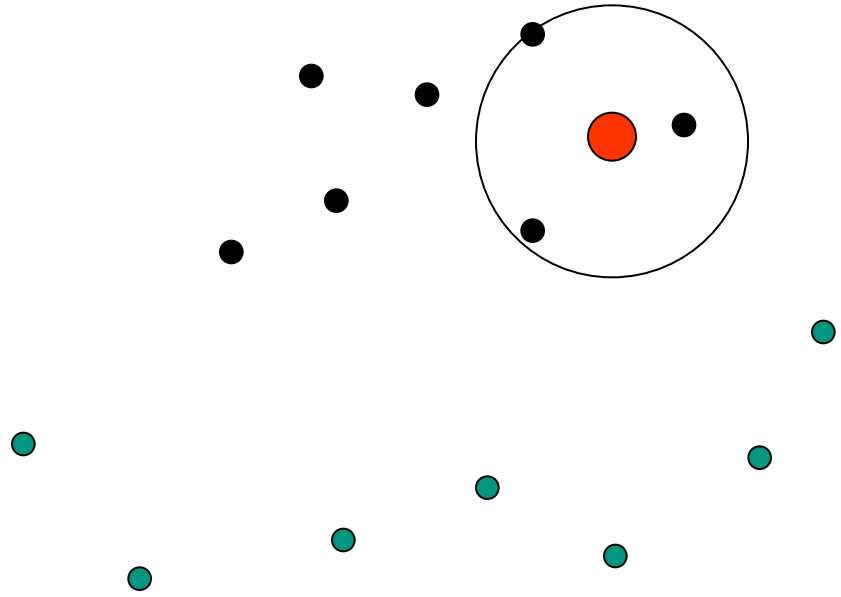
1-Nearest-Neighbor

- Features
  - All instances correspond to points in an n-dimensional Euclidean space
  - Classification done by comparing feature vectors of the different points



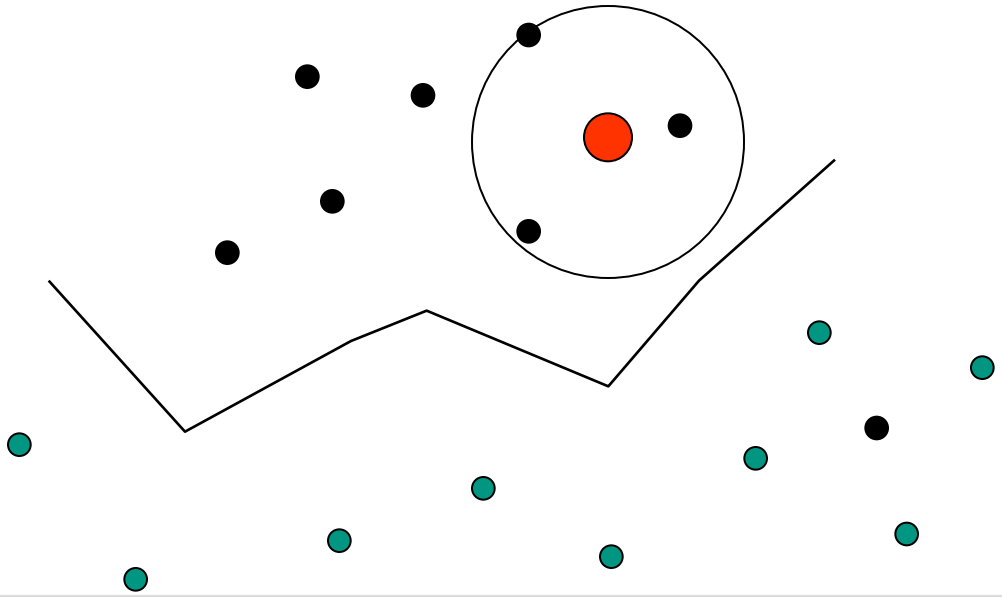
3-Nearest-Neighbor

- Search for the 3 nearest neighbors
  - Typically majority vote, when not all neighbors are from the same class



3-Nearest-Neighbor

- Decision surface
  - Described by the Voronoi diagram
  - Voronoi diagram is the dual of the Delaunay triangulation (computable in O(nlogn))



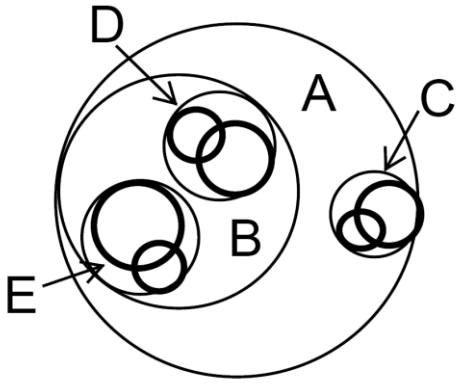
87

When to Consider Nearest Neighbor ?

- Lots of training data
- Less than 20 attributes (dimensions) per example
- Advantages:
  - Training is very fast
  - Learn complex target functions
  - Don’t lose information
- Disadvantages:
  - Slow at query time
  - Easily fooled by irrelevant attributes

K-Nearest Neighbors

- Typically decision surface is not computed, but for each new test sample, we compute the nearest neighbors on the fly
- Probabilistic interpretation: estimate density in a local neighborhood
- There are efficient memory indexing techniques in order to retrieve the stored training examples
  - kd-tree [Friedman et al. 1977]
  - Ball-tree



Literature

- Classification (Bayes, Gaussians, EM, …)
  - Duda, Hart, Stork: Pattern Classification, 2nd ed., 2000)
  - Mitchell: Machine Learning, 1997
  - Bishop: Pattern Recognition and Machine Learning, 2008
- SVMs
  - C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 2, 121-167 (1998)
  - Shawe-Taylor, Cristianini: Kernel Methods for Pattern analysis, 2004
  - Schölkopf, Smola: Learning with Kernels, 2001

End of Lecture

91

92

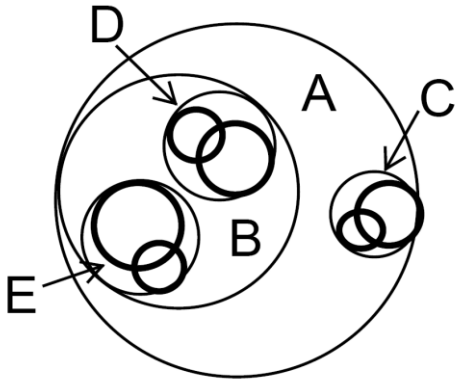
Further Details

(Not discussed during Lecture)

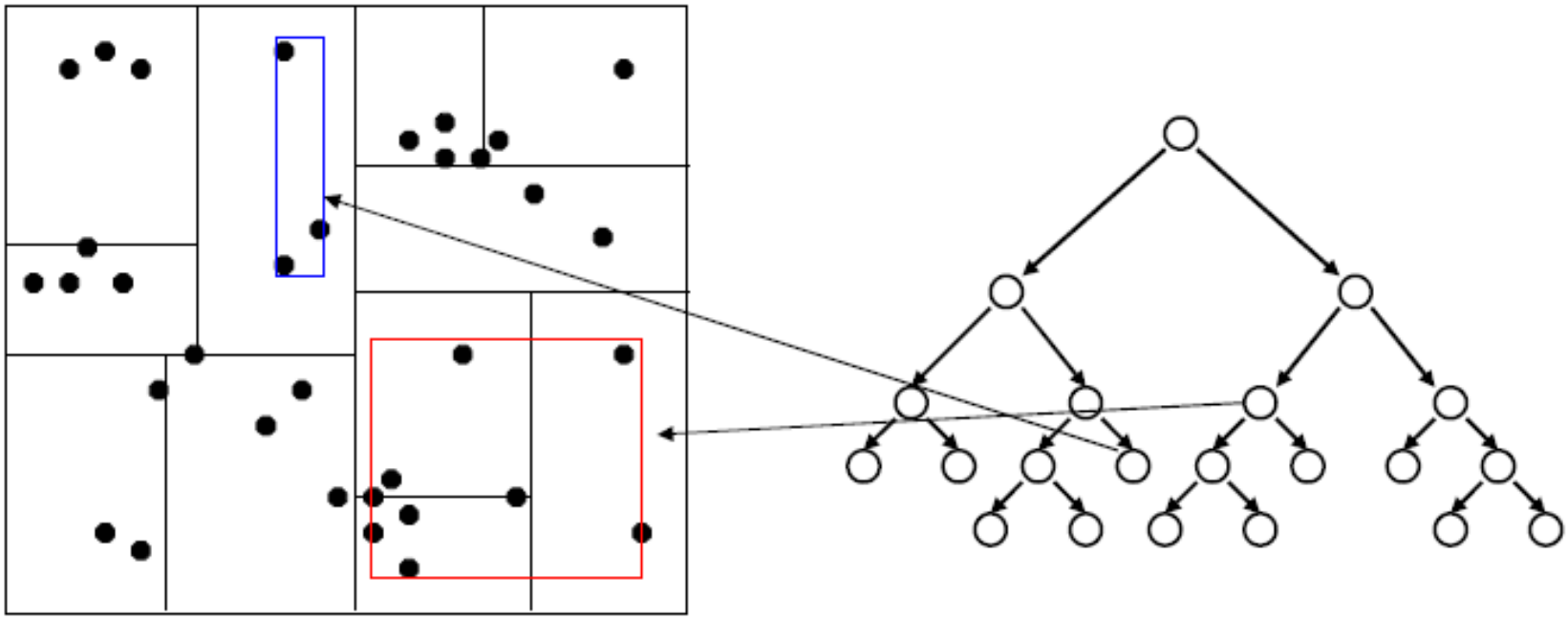
93

K-Nearest Neighbors

- Typically decision surface is not computed, but for each new test sample, we compute the nearest neighbors on the fly
- Probabilistic interpretation: estimate density in a local neighborhood
- There are efficient memory indexing techniques in order to retrieve the stored training examples
  - kd-tree [Friedman et al. 1977]
  - Ball-tree



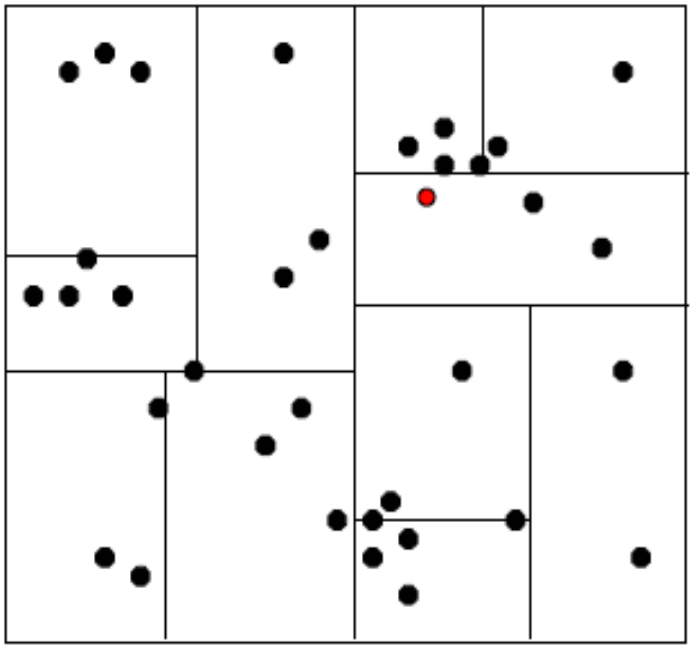
KD Tree for NN Search

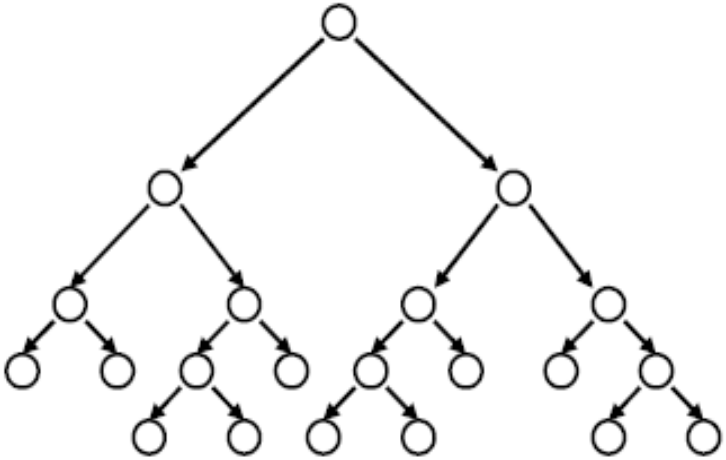


- Each node contains
  - Children information
  - The tightest box that bounds all the data points within the node.

95

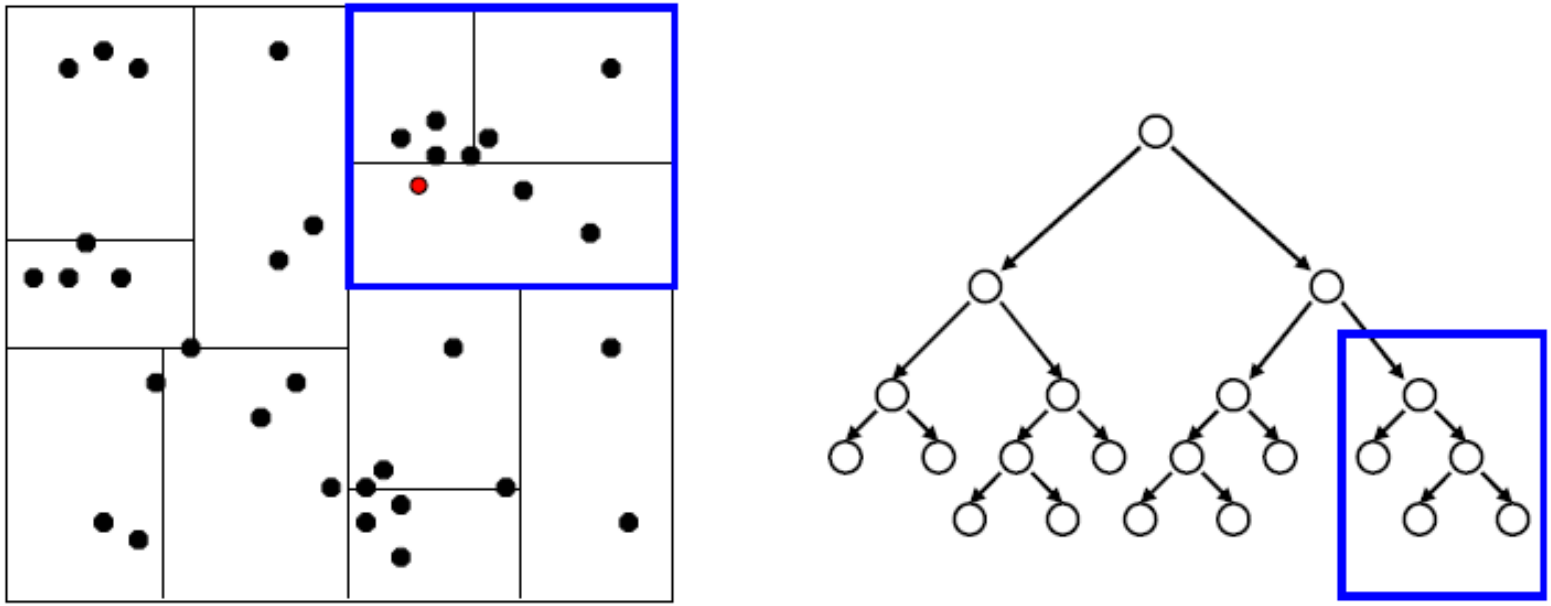
NN Search by KD Tree





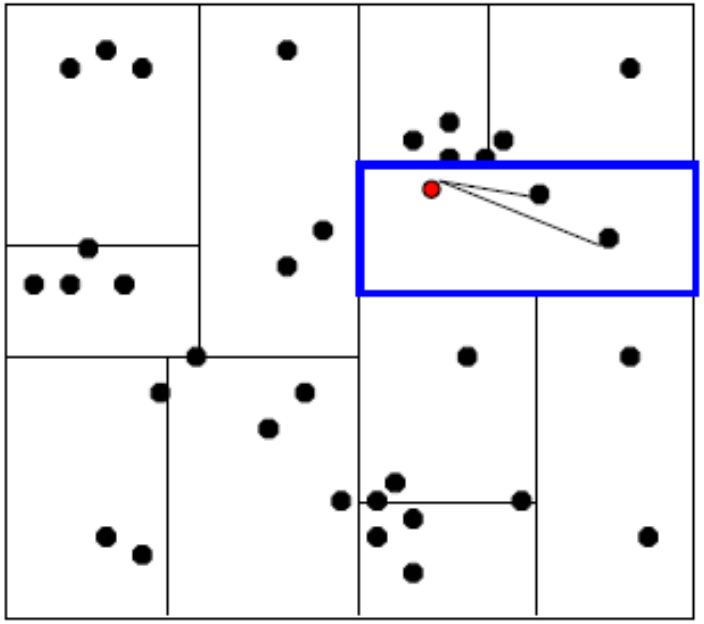
We traverse the tree looking for the nearest neighbor of the query point.

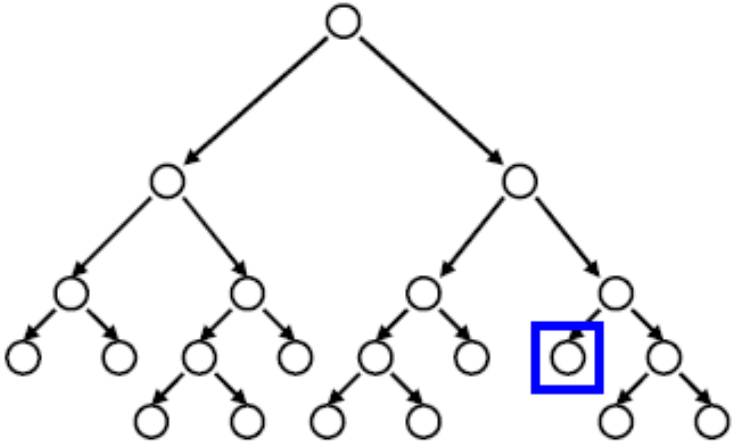
NN Search by KD Tree



Examine nearby points first: Explore the branch of the tree that is closest to the query point first.

NN Search by KD Tree

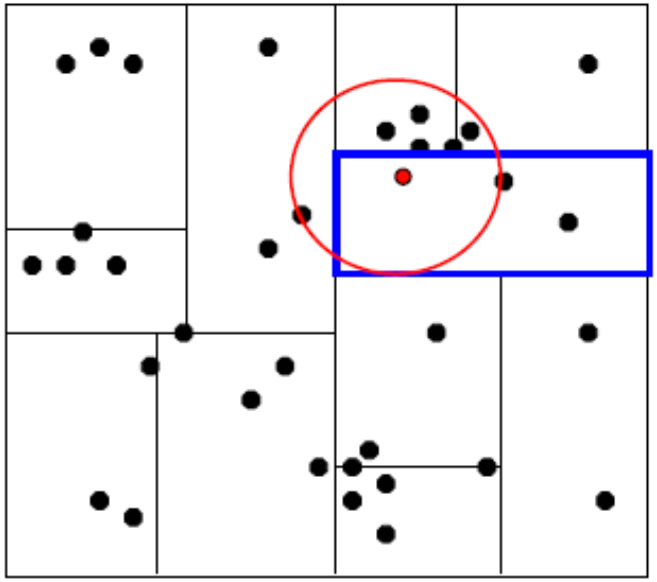


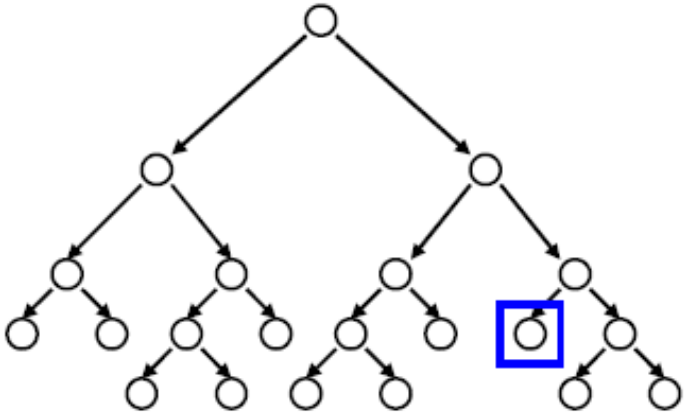


When we reach a leaf node: compute the distance to each point in the node.

98

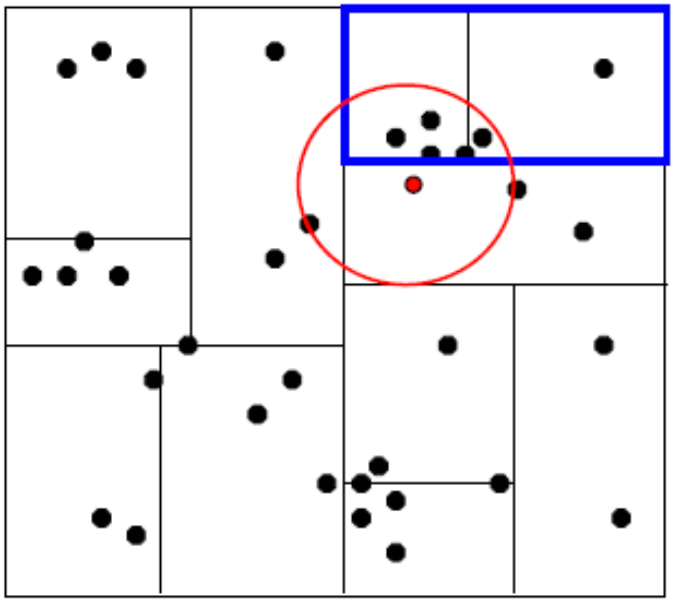
NN Search by KD Tree

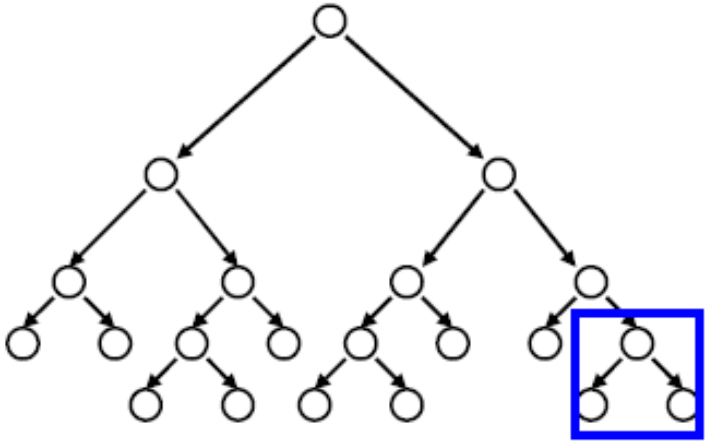




When we reach a leaf node: compute the distance to each point in the node.

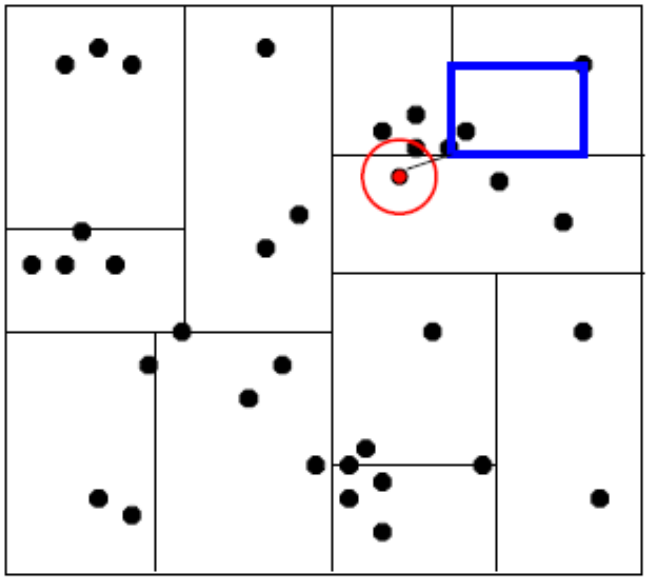
NN Search by KD Tree

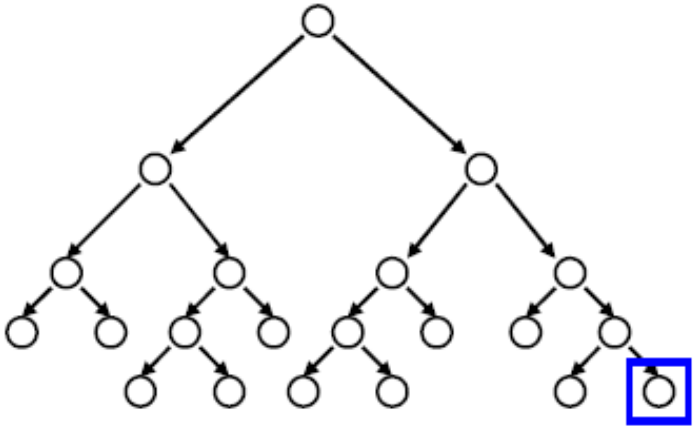




Then we can backtrack and try the other branch at each node visited.

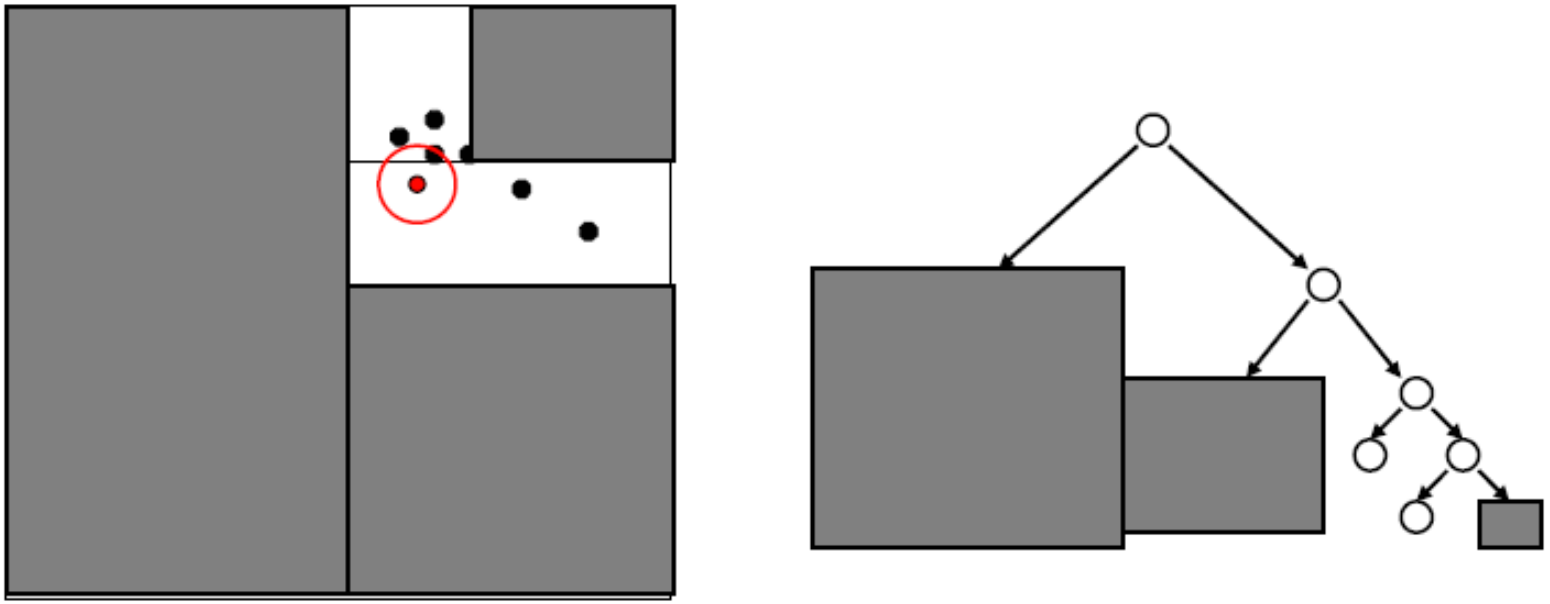
NN Search by KD Tree





Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

NN Search by KD Tree



Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

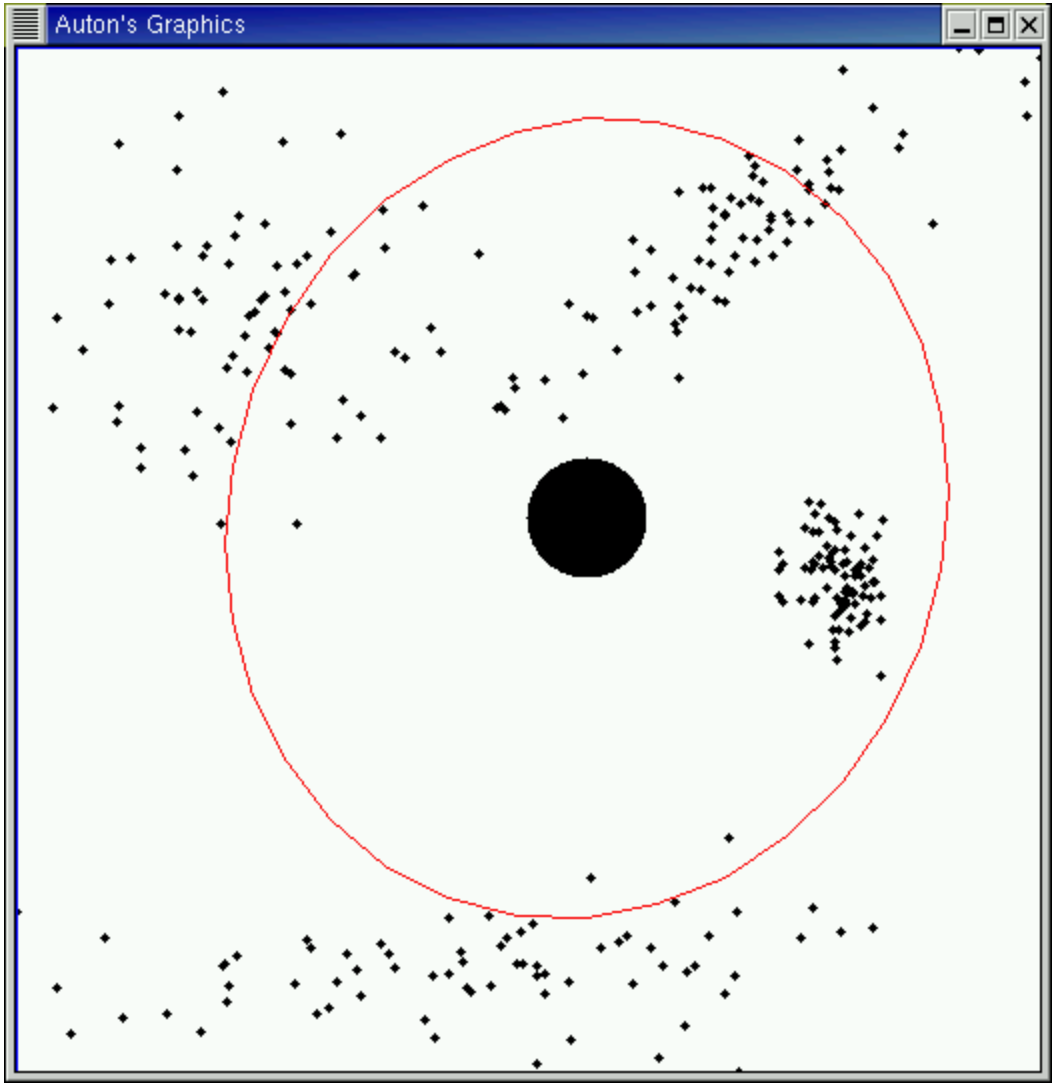
Standard Kd-tree construction

- Choose splitting planes by cycling through the dimensions
  - Example:
    - Root node: split in x-direction
    - Next level: split in y-direction
    - Next level: split in z-direction
- The position of the splitting plane is chosen to be the median of the points (with respect to their coordinates in the axis being used)
- Typically generates a quite balanced tree

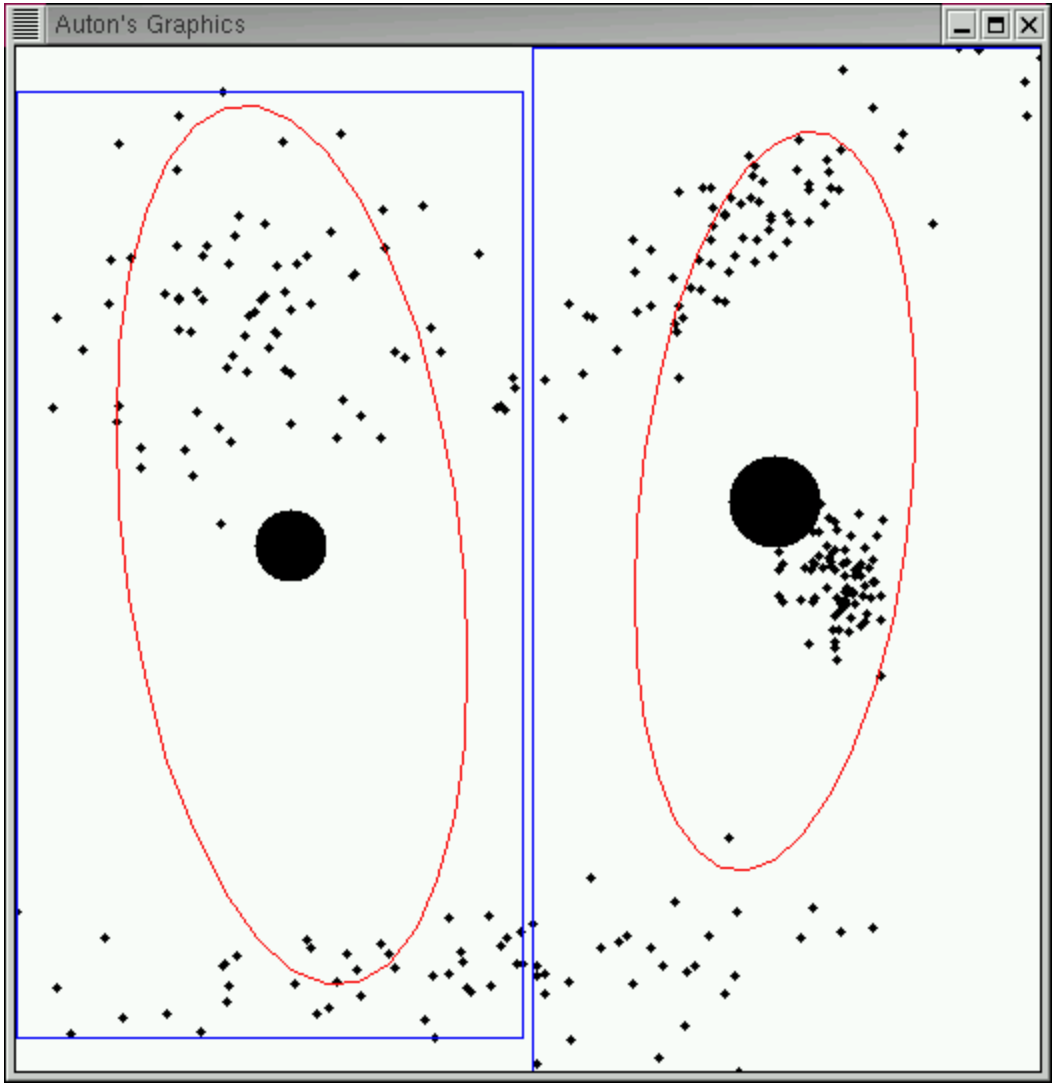
Kd-tree construction

- There exist many (sometimes application specific) heuristics that try to speed up kd-tree creation
  - E.g. split in the dimension of the highest variance
- Kd-trees can be built incrementally
  - Useful for incremental learning (for example when exploring a new territory with a robot)
- Existing libraries:
  - C++: libkdtree++

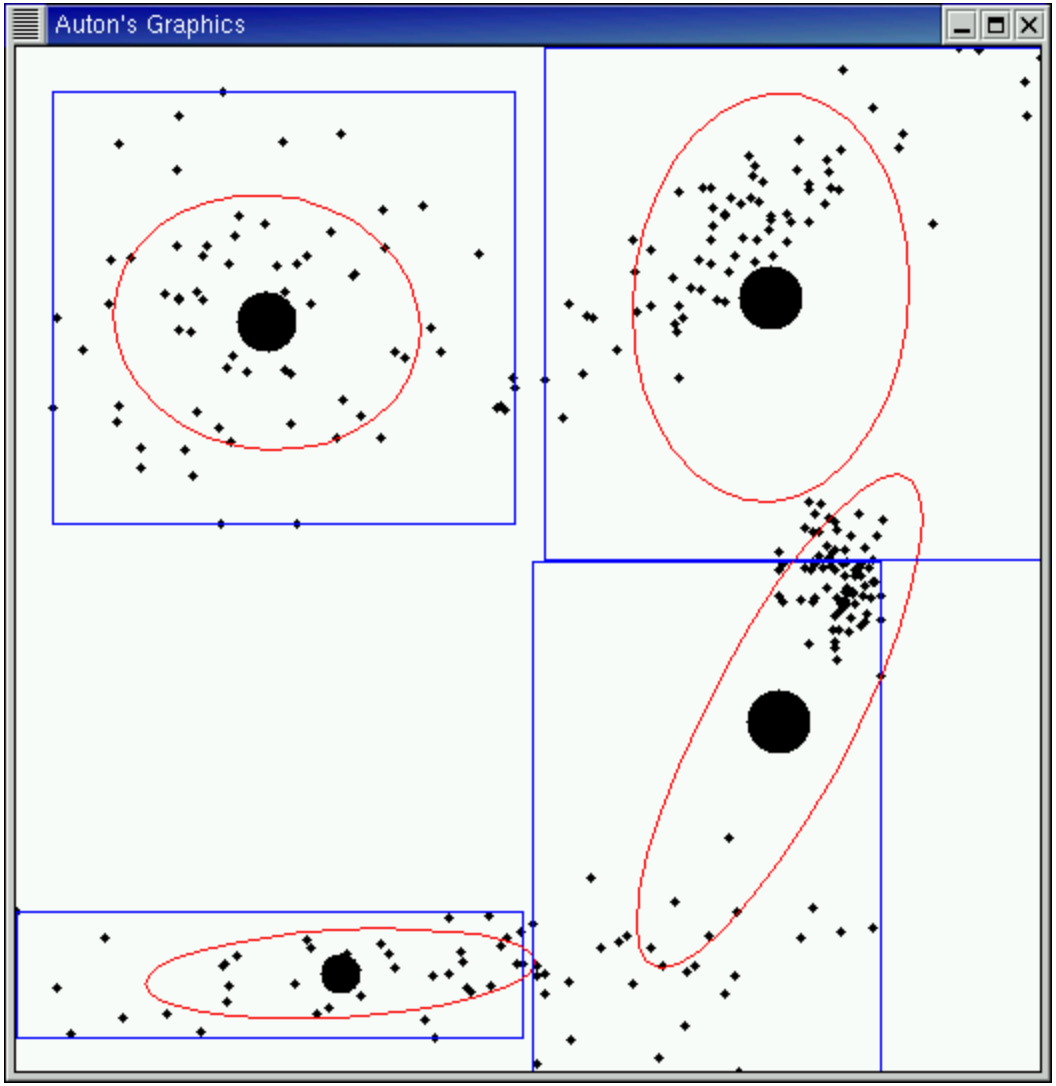
A *kd*-tree: level 1



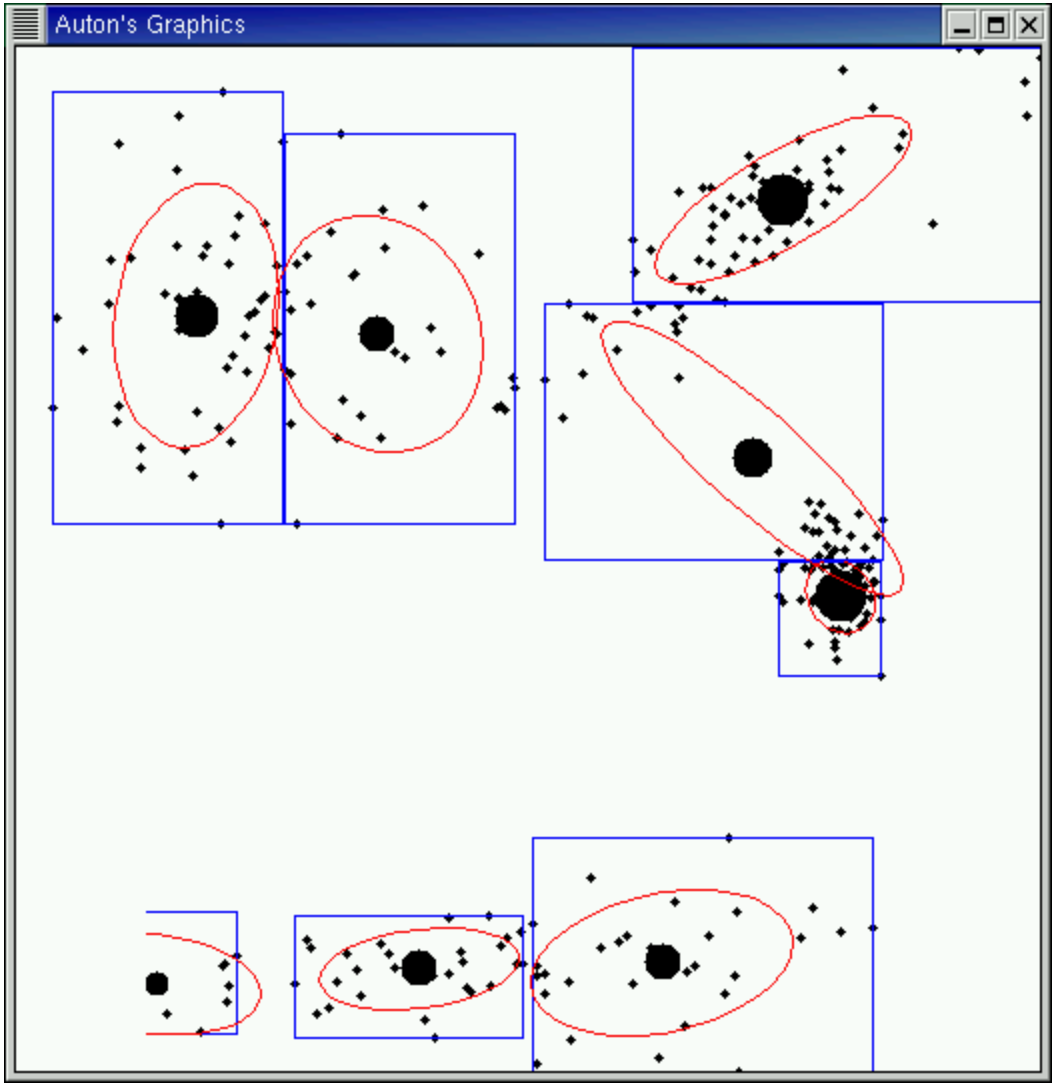
A *kd*-tree: level 2



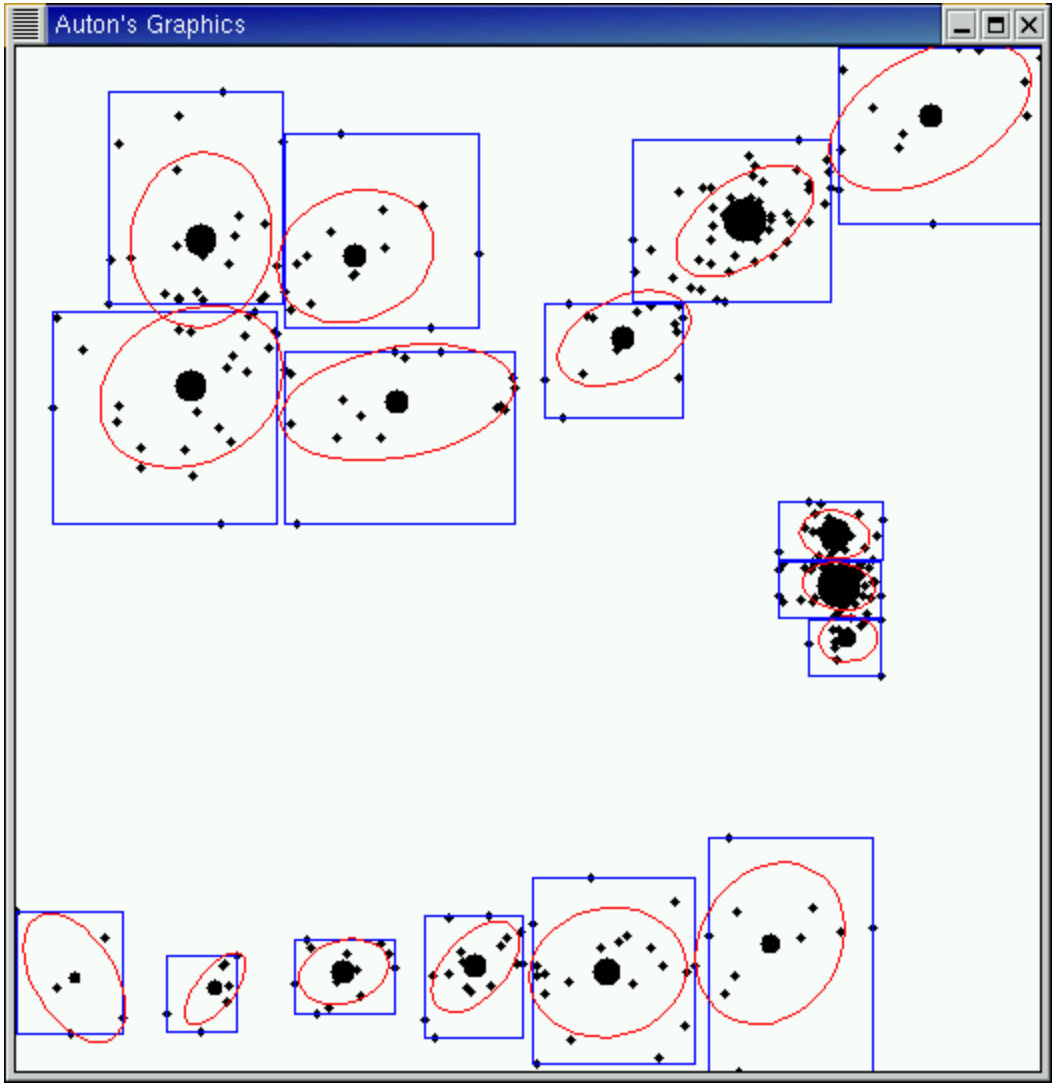
A *kd*-tree: level 3



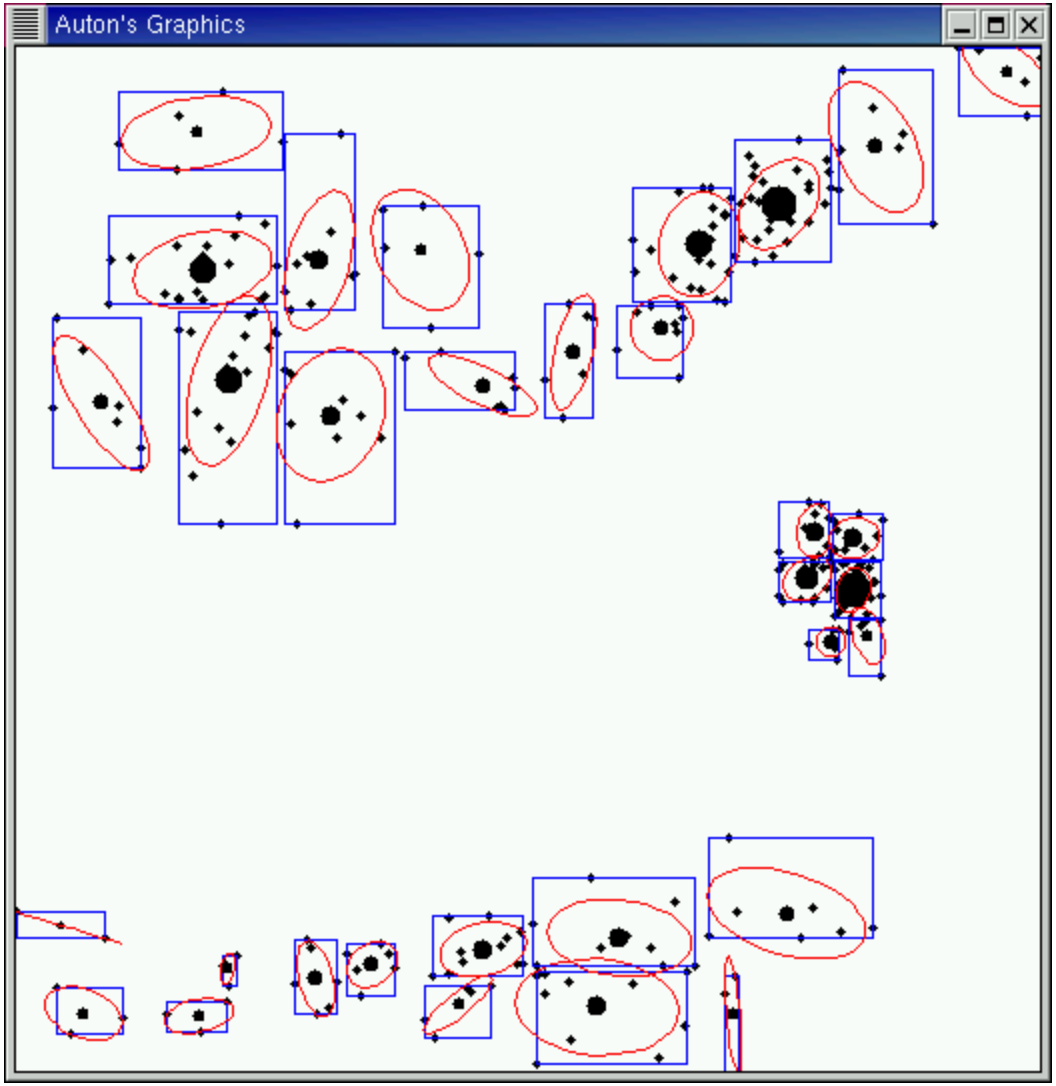
A *kd*-tree: level 4



A *kd*-tree: level 5



A *kd*-tree: level 6



Complexity

- Building a static *k*d-tree from *n* points: O(*n* log *n*)
  - O(log n) tree levels, O(n) median search
- Insertion into a balanced *k*d-tree: O(log *n*)
- Removal from a balanced *k*d-tree O(log *n*)
- Query of an axis-parallel range in a balanced *k*d-tree: $O(n^{1-1/d} + k)$
  - with k the number of the reported points, and d the dimension of the *k*d-tree.

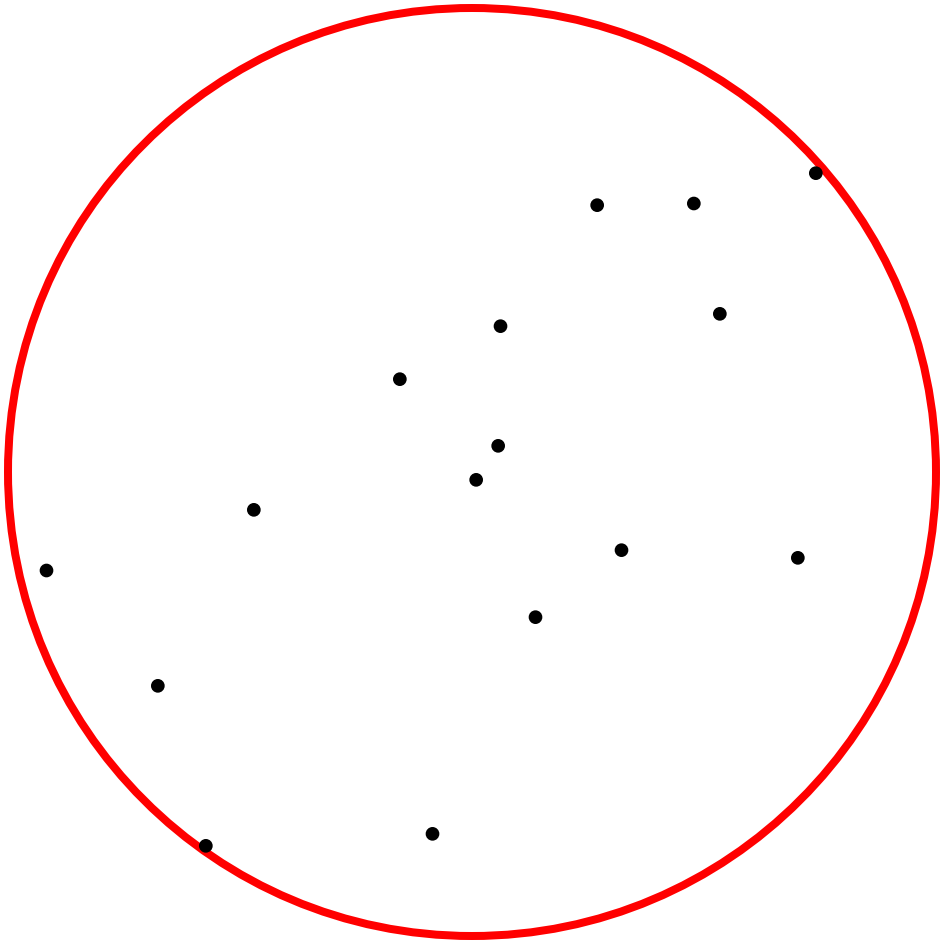
Problem of Dimensionality

- Imagine instances described by 20 attributes, but only 2 are relevant to target function
- Curse of dimensionality: nearest neighbor is easily mislead when high dimensional X

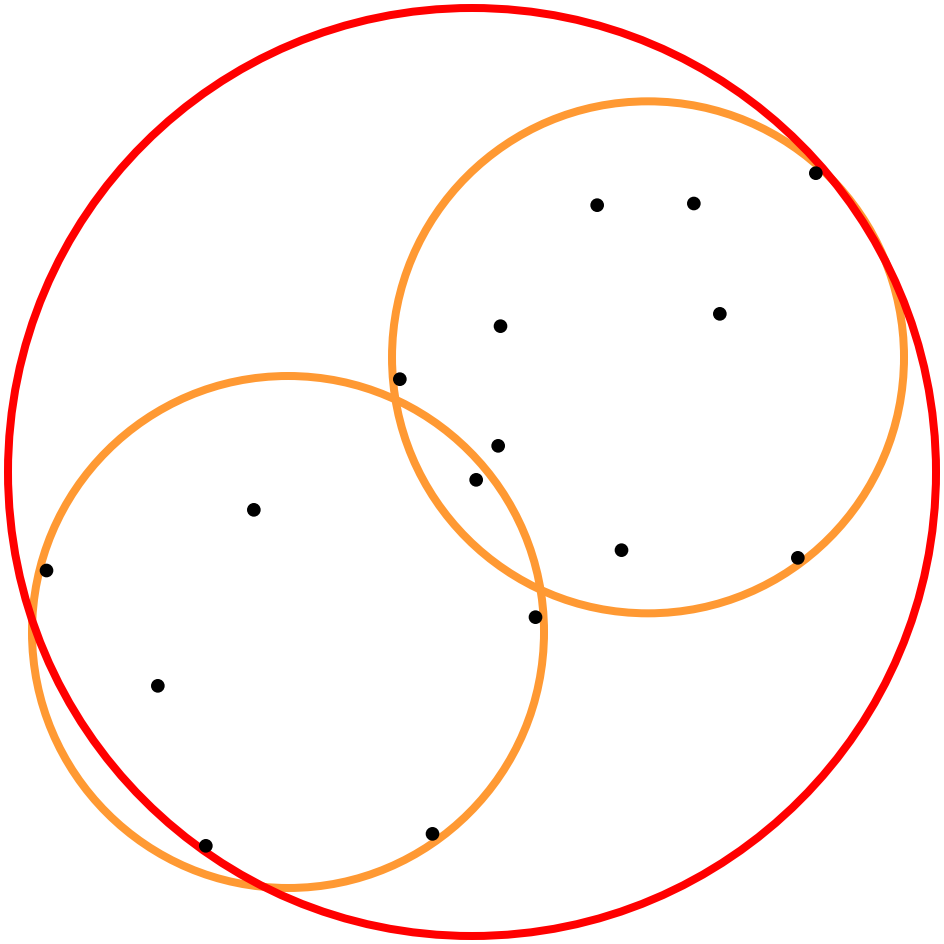
Ball Trees

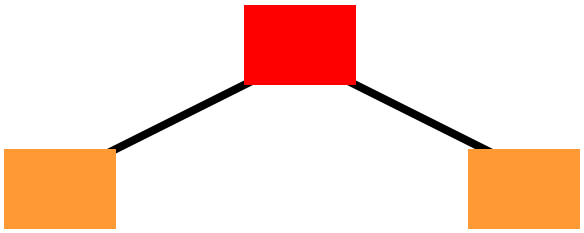
- Can be constructed in a similar fashion as kd-trees
- Ball trees have shown to be superior to kd-trees in many applications (though there is high variance and dataset dependence)
  - The Proximity Project [Gray, Lee, Rotella, Moore 2005]

A ball-tree: level 1

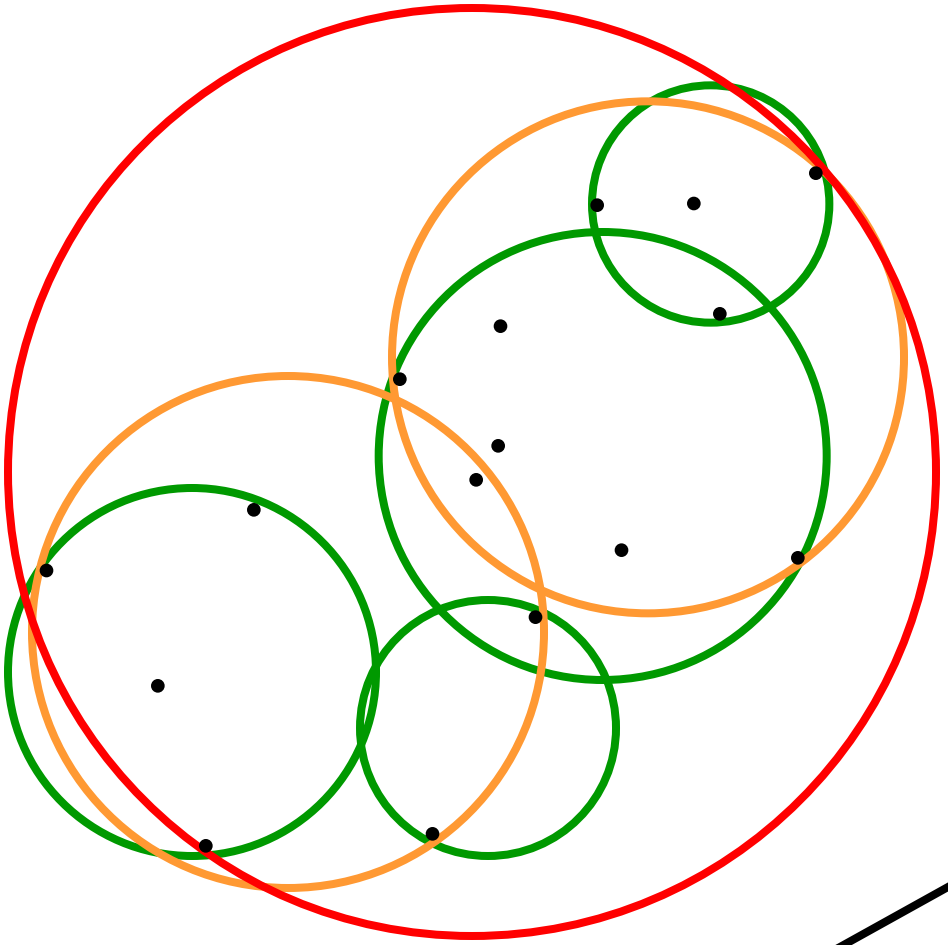


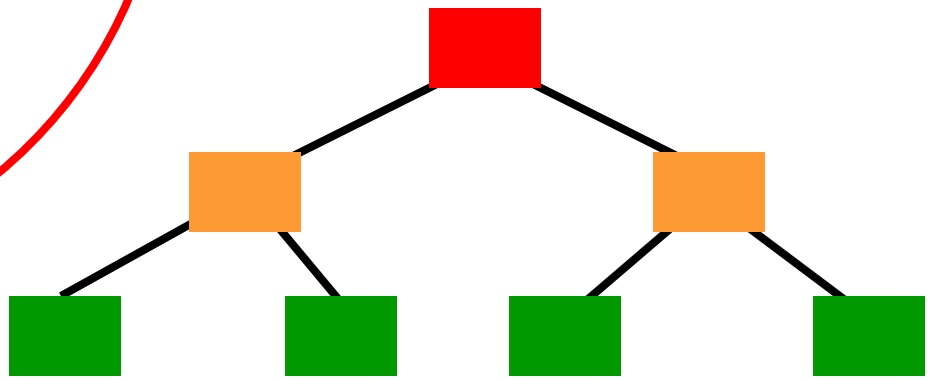
A ball-tree: level 2



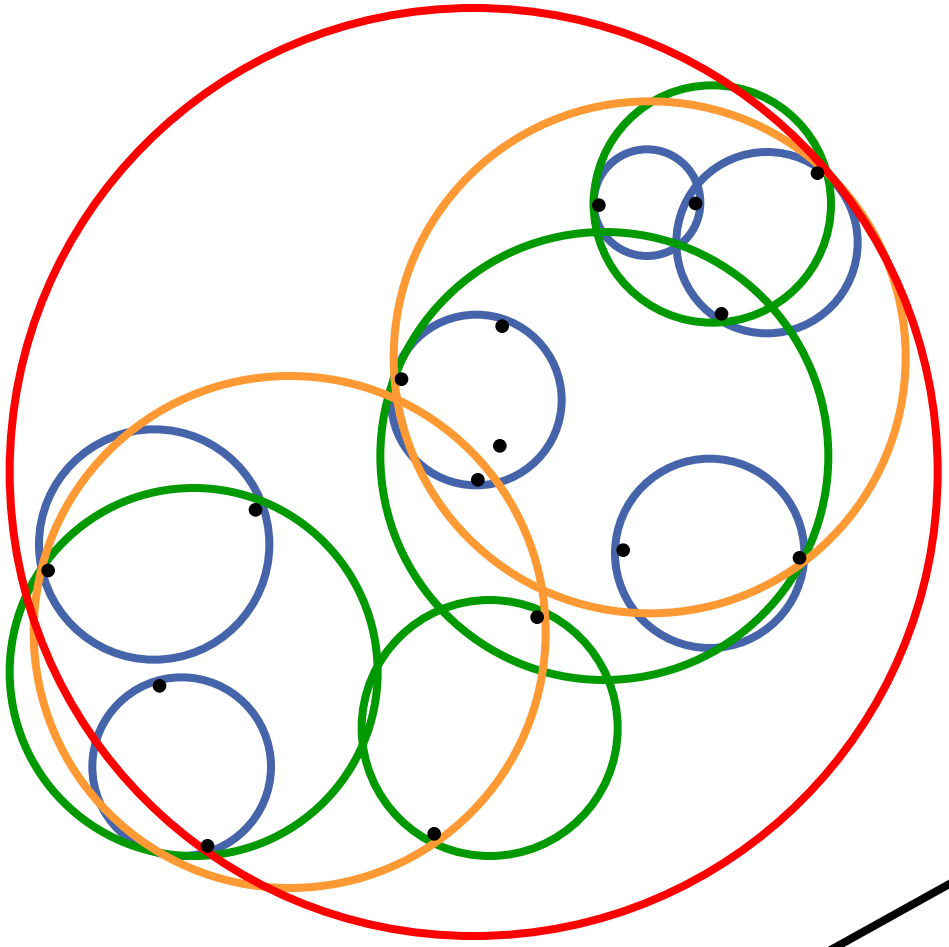


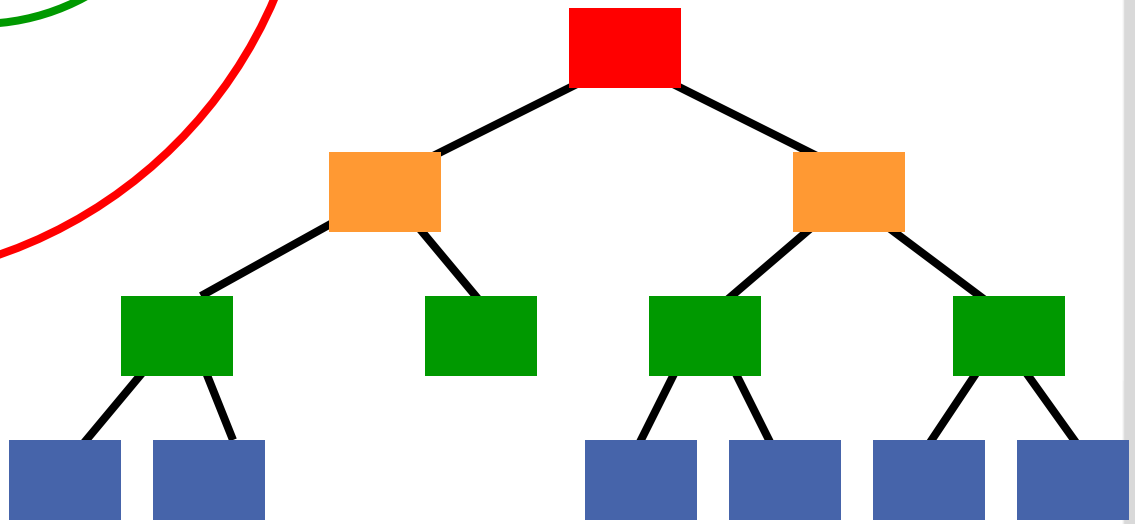
A ball-tree: level 3





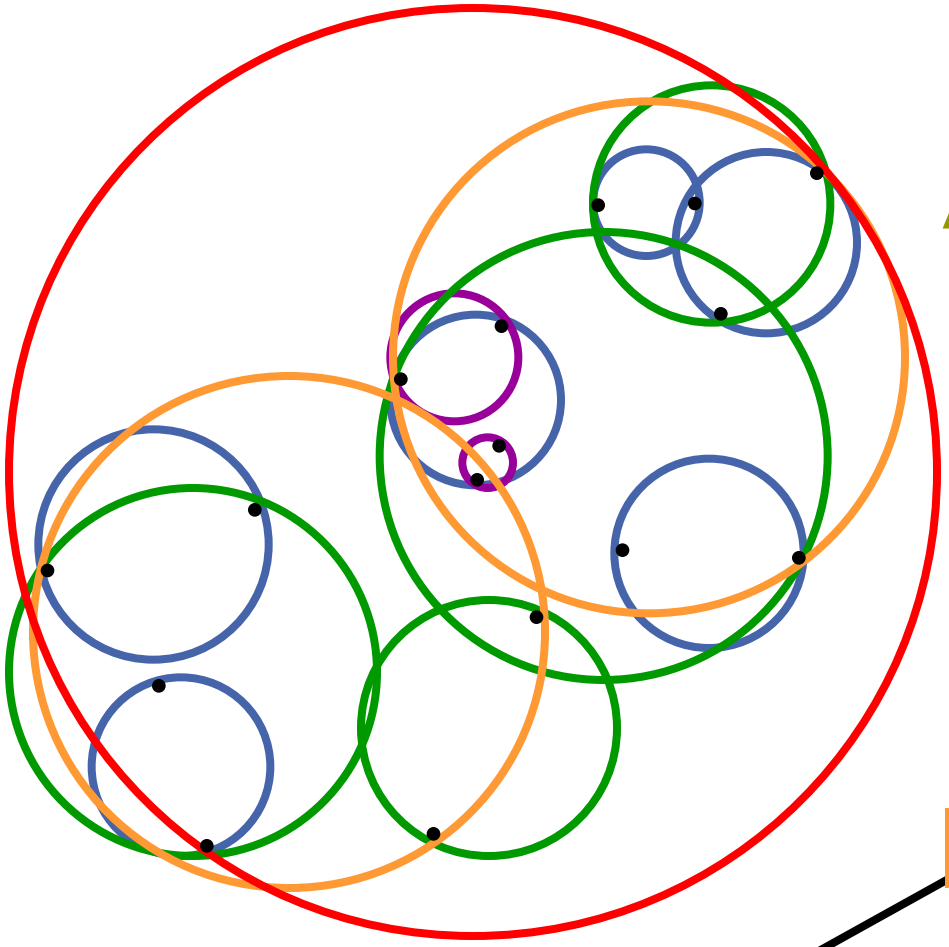
A ball-tree: level 4

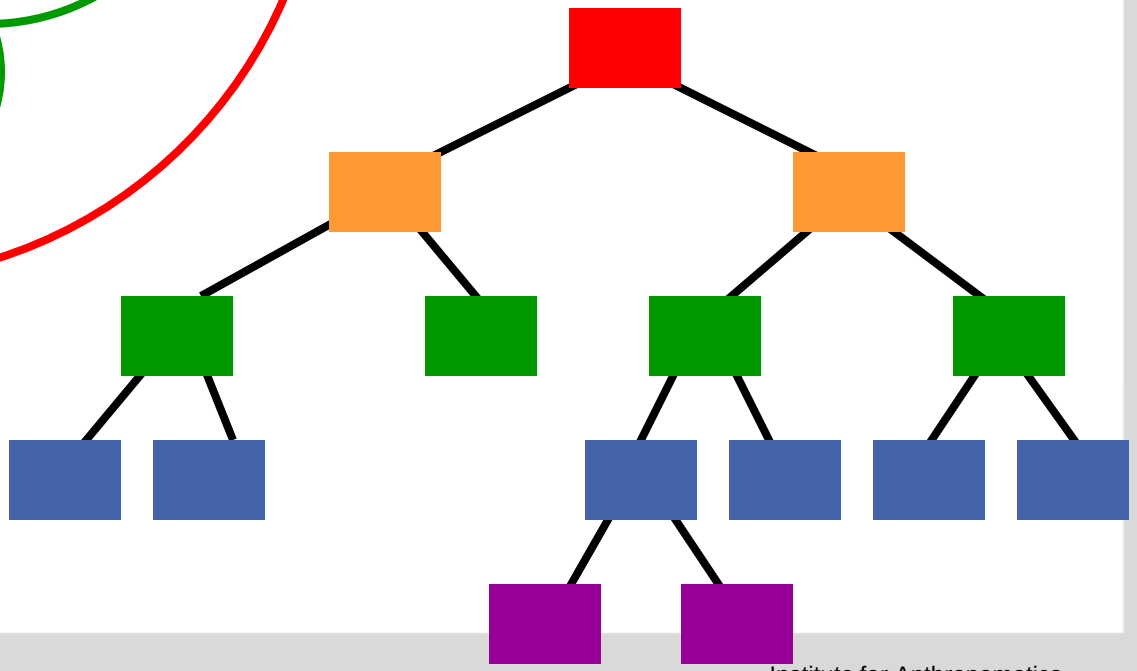




118

A ball-tree: level 5





119

kNN Discussion

- Highly effective inductive inference method for noisy training data and complex target functions
- Target function for a whole space may be described as a combination of less complex local approximations
- Learning is very simple
- Classification can be time consuming, if training data set is large: O(logn)

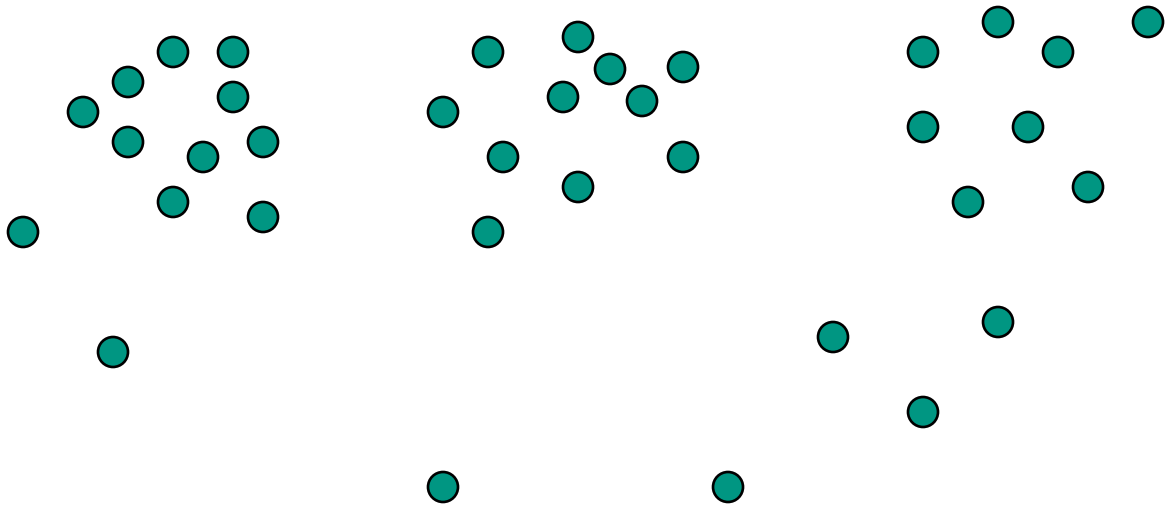
120

Clustering

121

Clustering

- New problem setting
  - Only data points are given, no class labels
  - Find structures in given data
- Generally no single correct solution possible



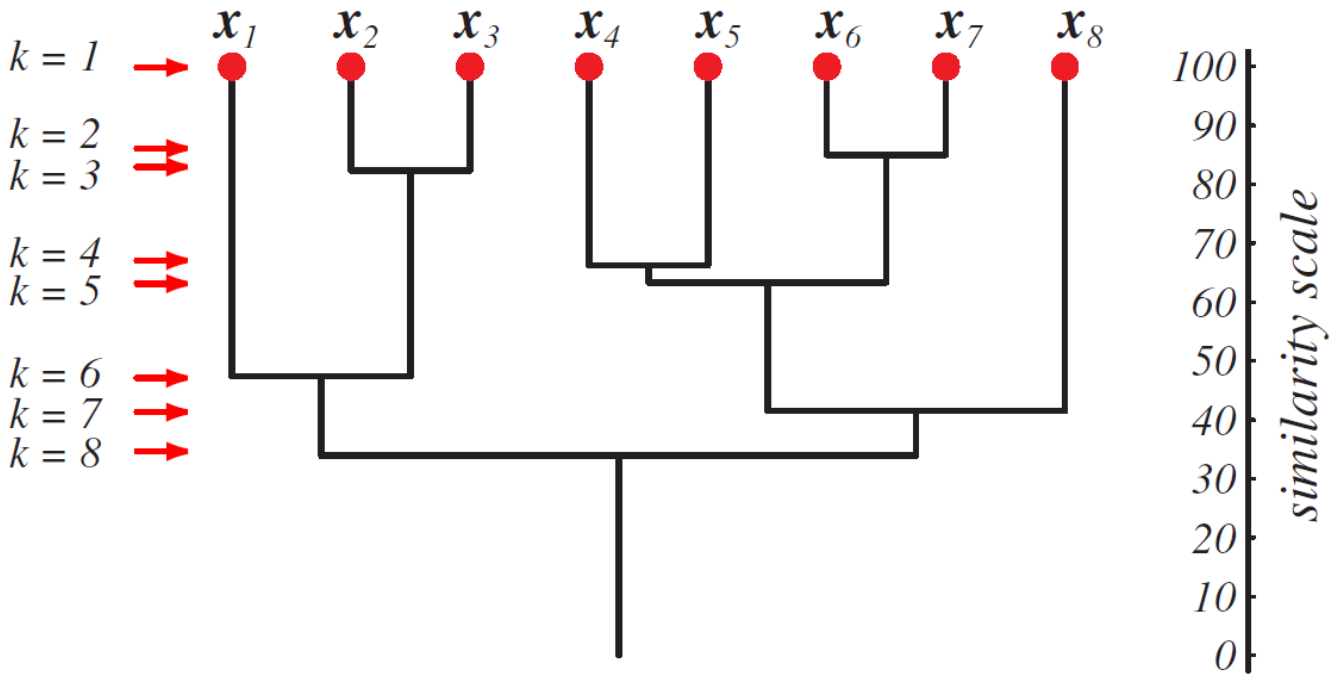
K-means

- Algorithm
  - Randomly initialize k cluster centers
  - Repeat until convergence:
    - Assign all data points to closest cluster center
    - Compute new cluster center as mean of assigned data points
- Pros
  - Simple and efficient
- Cons
  - k needs to be known in advance
  - Results depend on initialization
  - Does not work well for clusters that are not hyperspherical (round) or clusters that overlap
- K-means is very similar to the EM algorithm
  - Uses hard assignments instead of probabilistic assignments (EM)
  - EM algorithm can also be used for clustering
- Applets
  - http://www.inf.ethz.ch/personal/porbanz/ml2/applets/Cluster/kMeansApplet.html
  - http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

Agglomerative Hierarchical Clustering

- Algorithm
  - Start with one cluster for each data point
  - Repeat
    - Merge two closest clusters
- Several possibilities to measure cluster distance
  - Min: minimal distance between elements
  - Max: maximal distance between elements
  - Avg: average distance between elements
  - Mean: distance between cluster means
- Result is a tree called a dendrogram

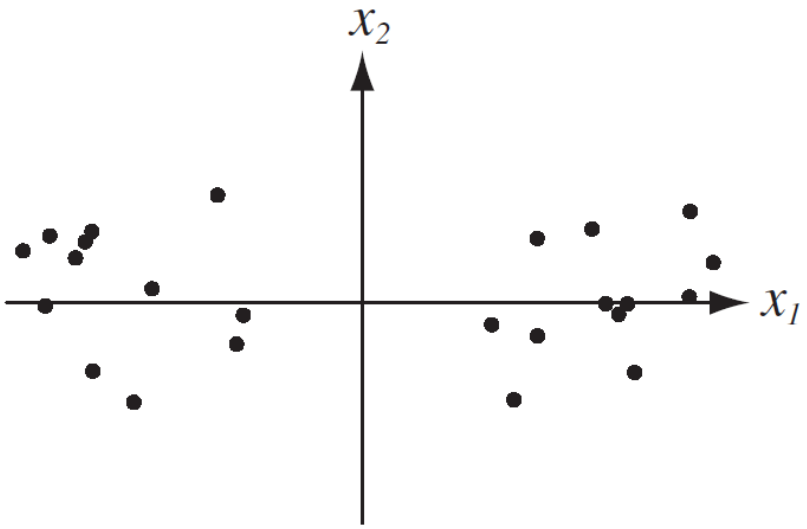
AHC dendrogram



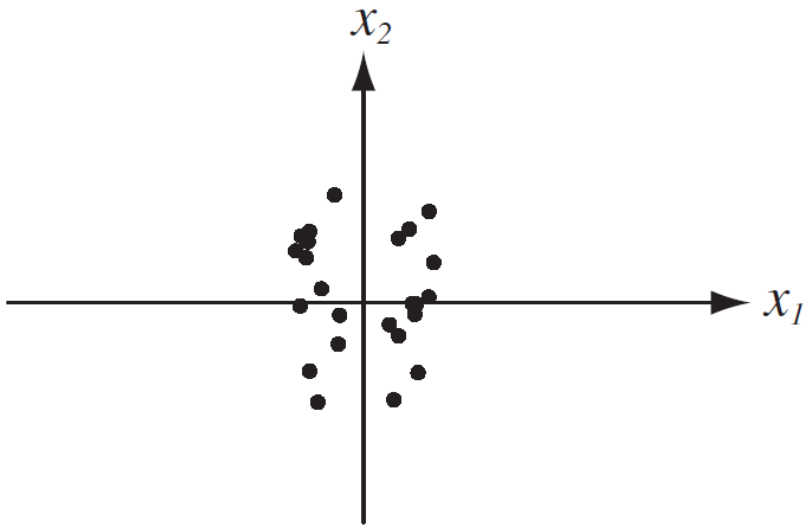
- AHC applet
  - http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletH.html
  - http://metamerist.com/slhc/example40.htm

Scaling

- Scaling of features can have a large impact on clustering



Nice clusters



Clustering not so clear anymore