Content-based image and video analysis

Machine learning for multimedia retrieval

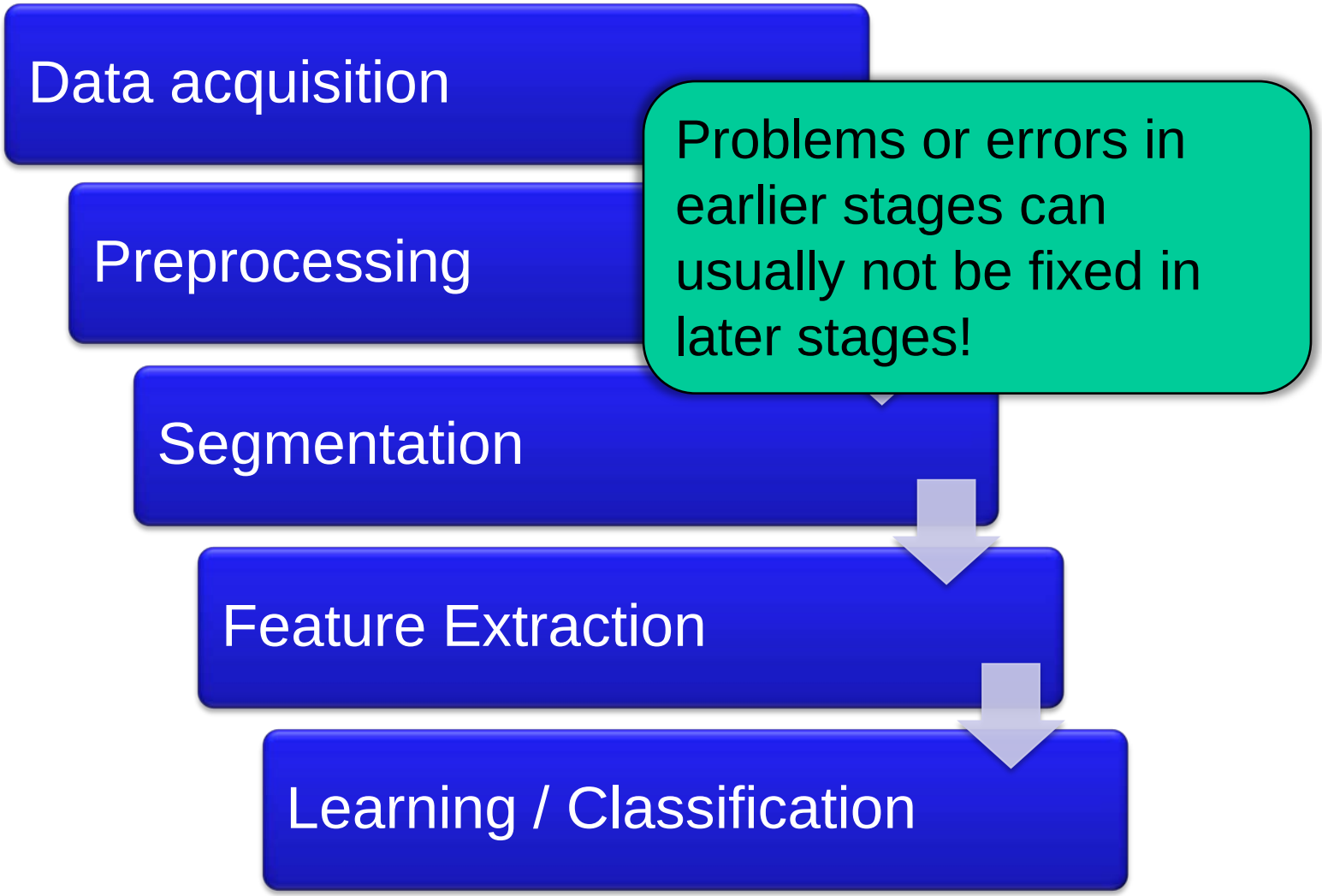
16.10.2012

cv:hci Computer Vision for Human-Computer Interaction
Research Group
Universität Karlsruhe (TH)
Research University · founded 1825
Fakultät für Informatik

Machine Learning / Classification

- An important task in machine learning is classification:
  - *Given an input pattern **x**, assign in to a class ω_i*
    - Example: Given an image, assign label “face” or “non-face”
  - **x** can be an image, a video, or (more commonly) any feature vector that can be extracted from them
  - ω_i is the desired (discrete) class label
    - If “class label” is real number or vector ➔ *Regression*
  - ML: Use example patterns with given class labels to automatically learn
- Many tasks in multimedia retrieval can be posed as classification tasks
  - Genre classification
  - Object detection
  - High-level feature detection
  - …

Classification pipeline

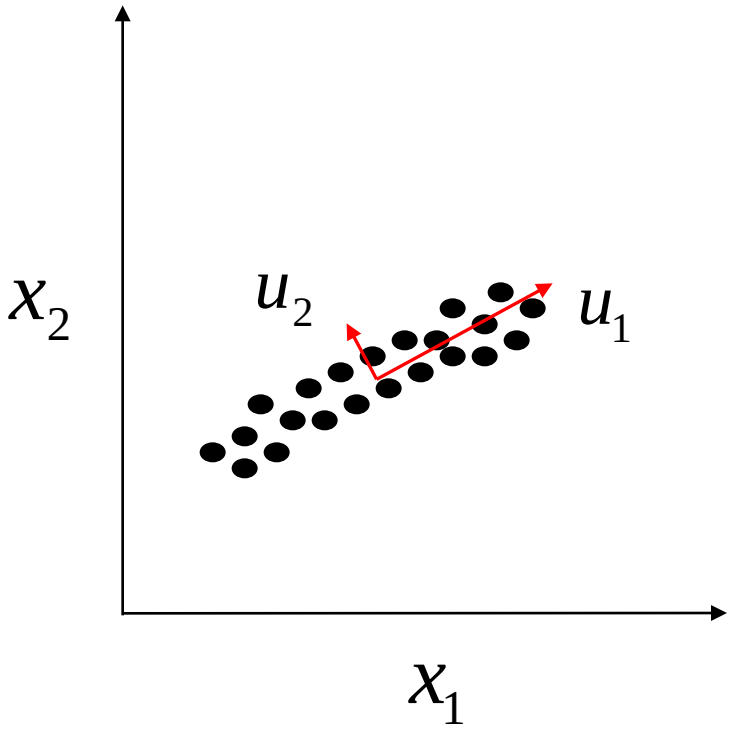


Curse of dimensionality

- In computer vision, the extracted feature vectors are often high-dimensional
- Many intuitions about linear algebra are no longer valid in high-dimensional spaces
- Classifiers often work better in low-dimensional spaces
- These problems that present themselves in high-dimensional spaces are often called ***"curse of dimensionality"***

Dimensionality reduction

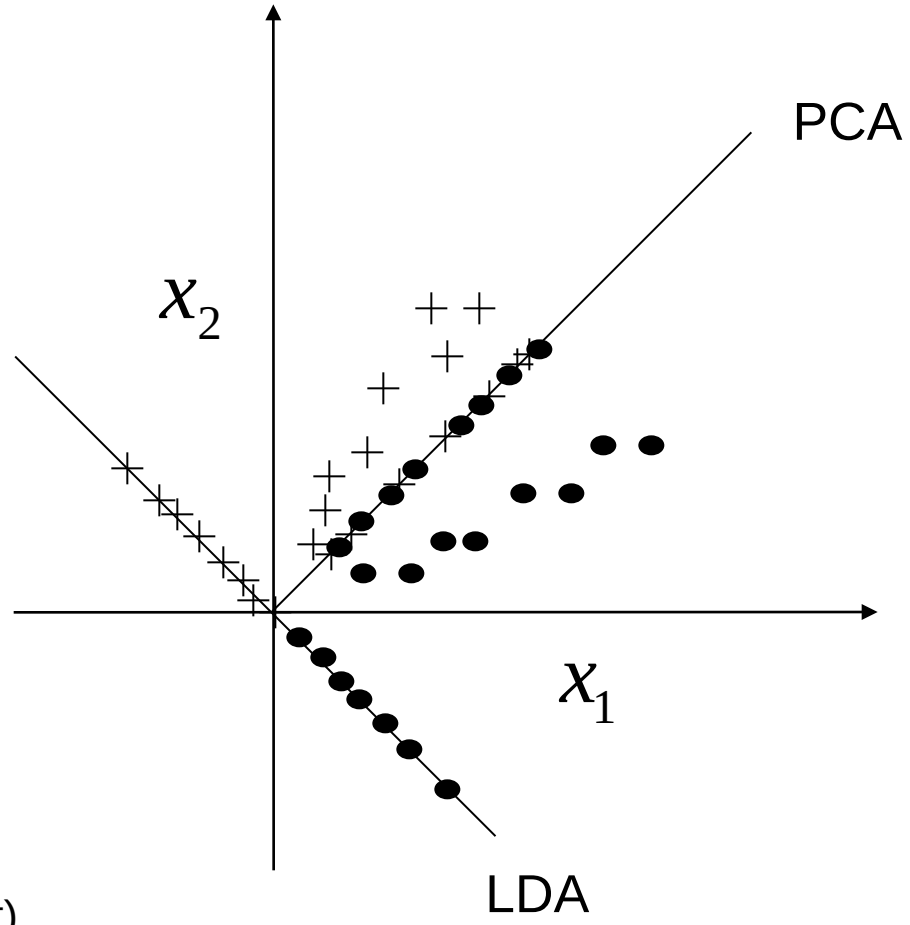
- PCA: Leave out dimensions and minimize error made



(see Duda & Hart)

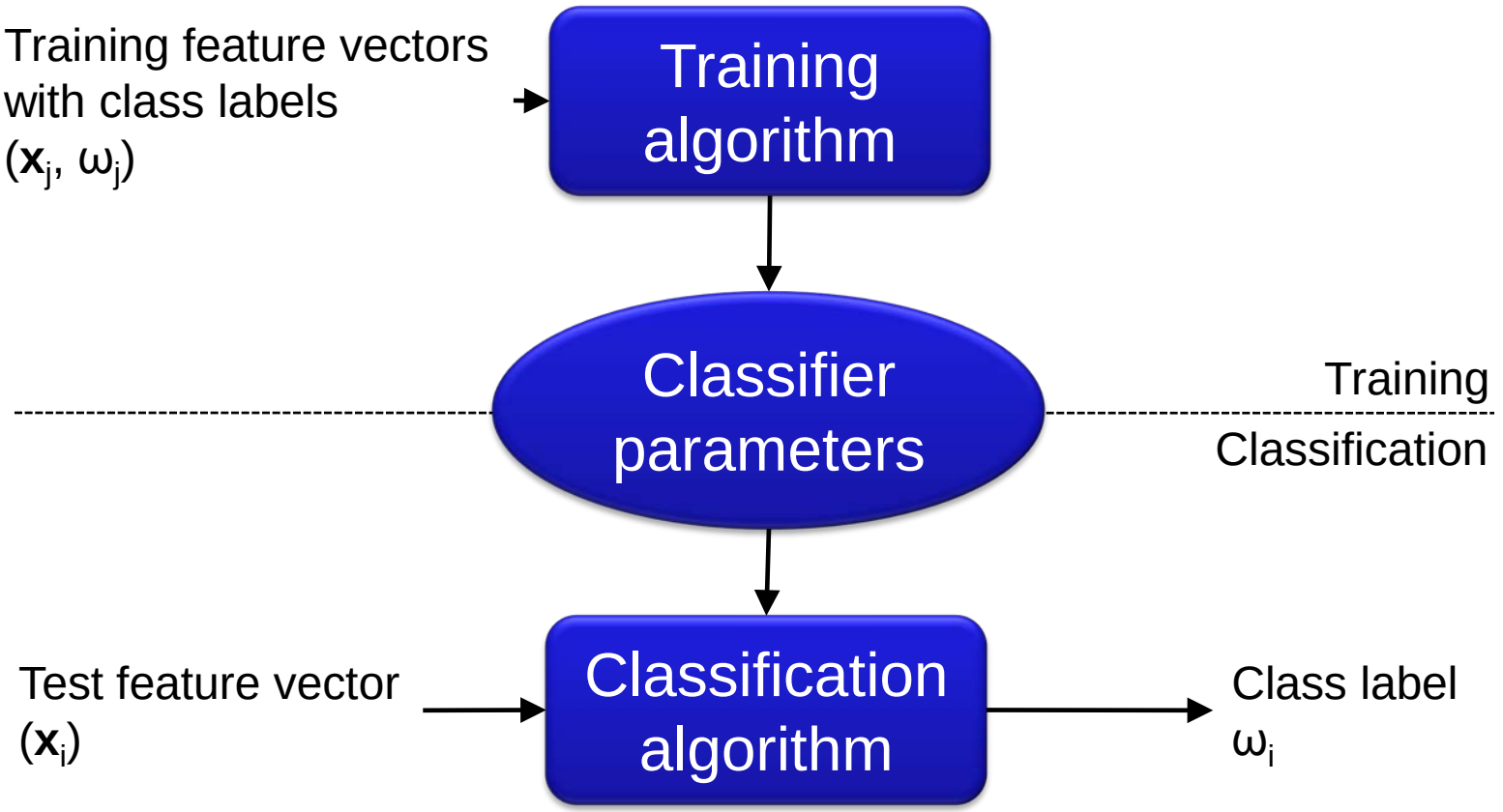
Dimensionality reduction

- LDA: Maximize class separability



(→ see Duda & Hart)

Classification process



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:

$$P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i) P(\omega_i)}{p(\mathbf{x})} \quad \rightarrow \quad \text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{normalization factor}}$$

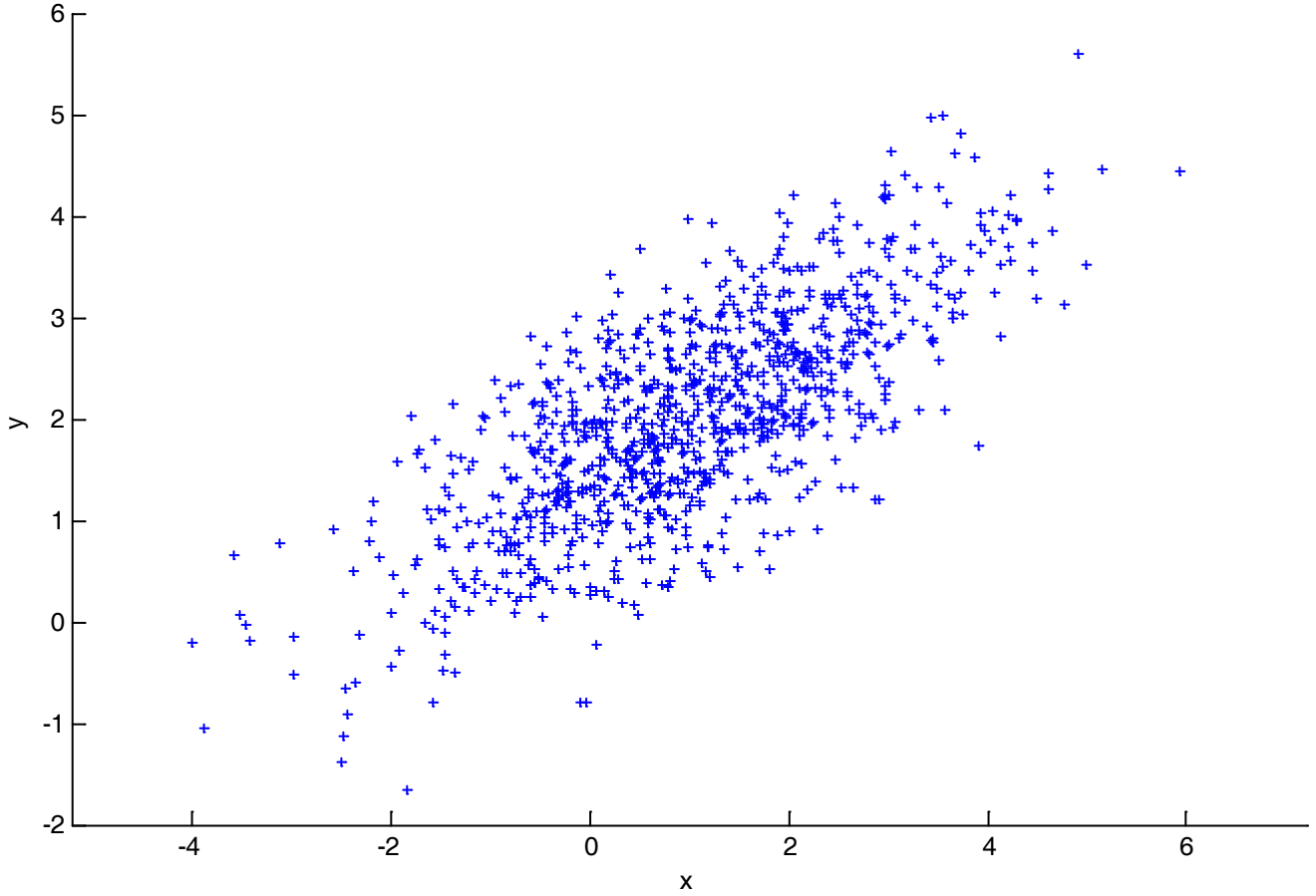
$$\text{with} \quad p(\mathbf{x}) = \sum_i p(\mathbf{x} \mid \omega_i) P(\omega_i)$$

- Decide for the class ω_i with maximum posterior probability
- If we know $P(\mathbf{x}|w_i)$ 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)

Gaussian classification

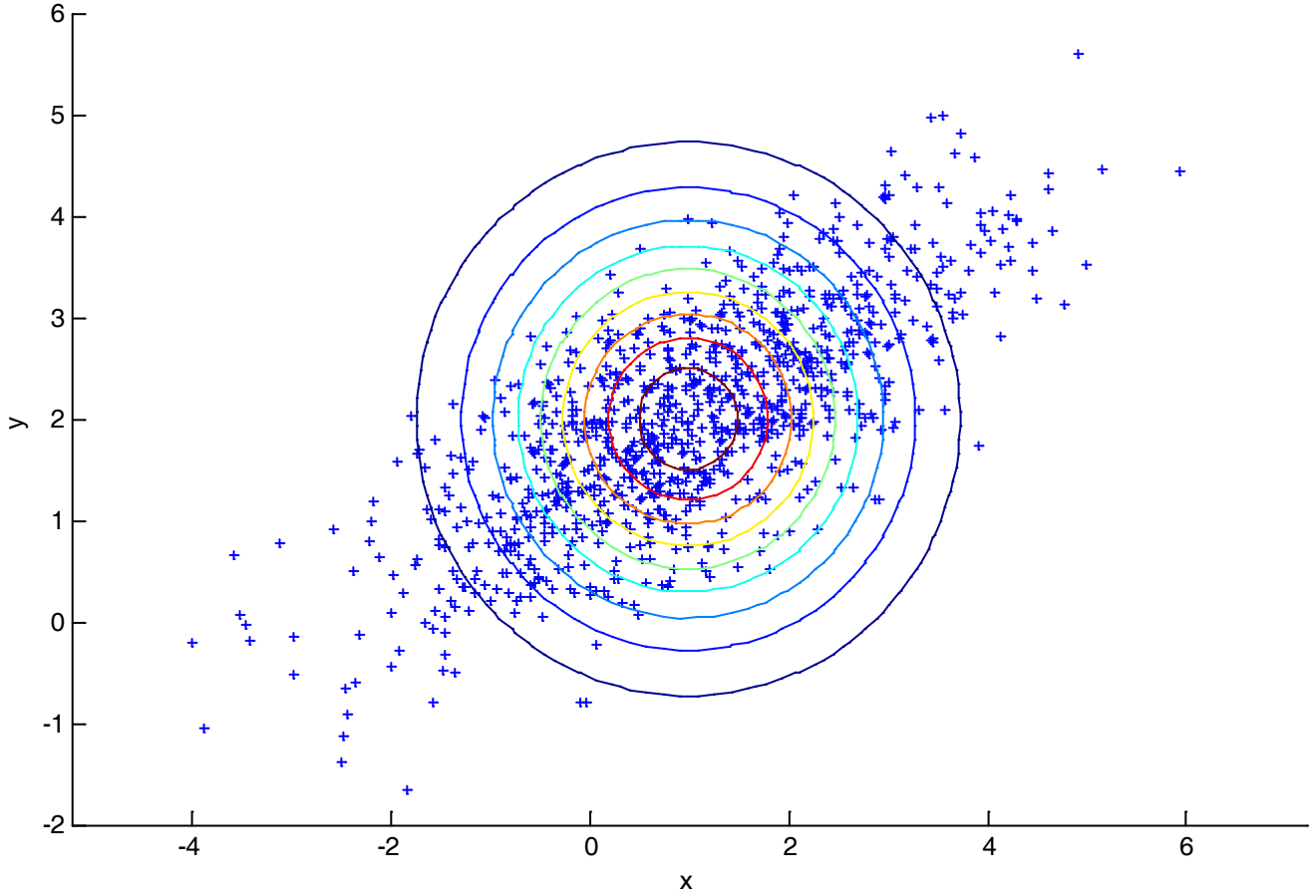
- Assumption: $p(\mathbf{x}|\omega_i) \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$
- This makes estimation easier:
  - Only mu and sigma 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 (often severely)
- Estimation of mu and 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



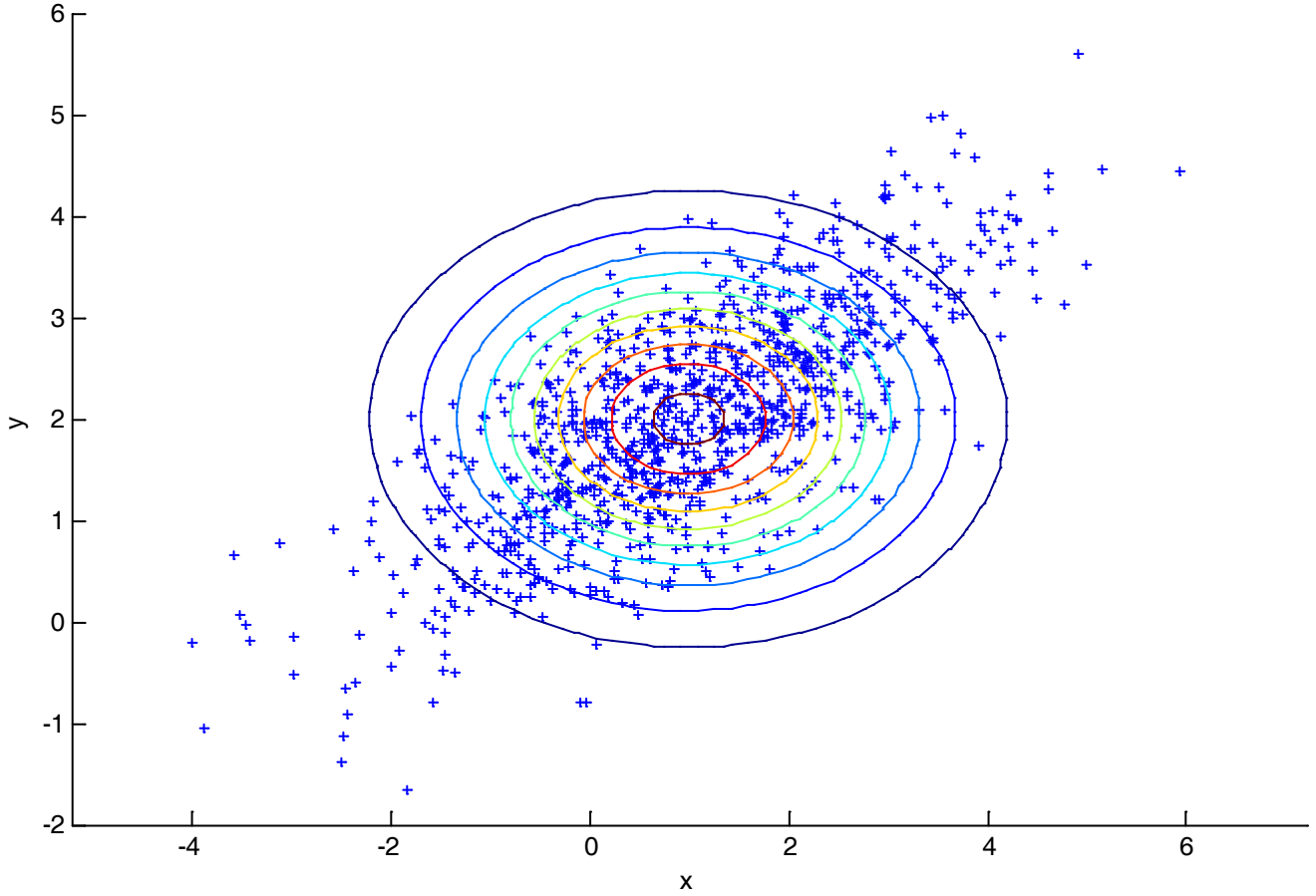
Random samples drawn from a Gaussian distribution

Gaussian Example



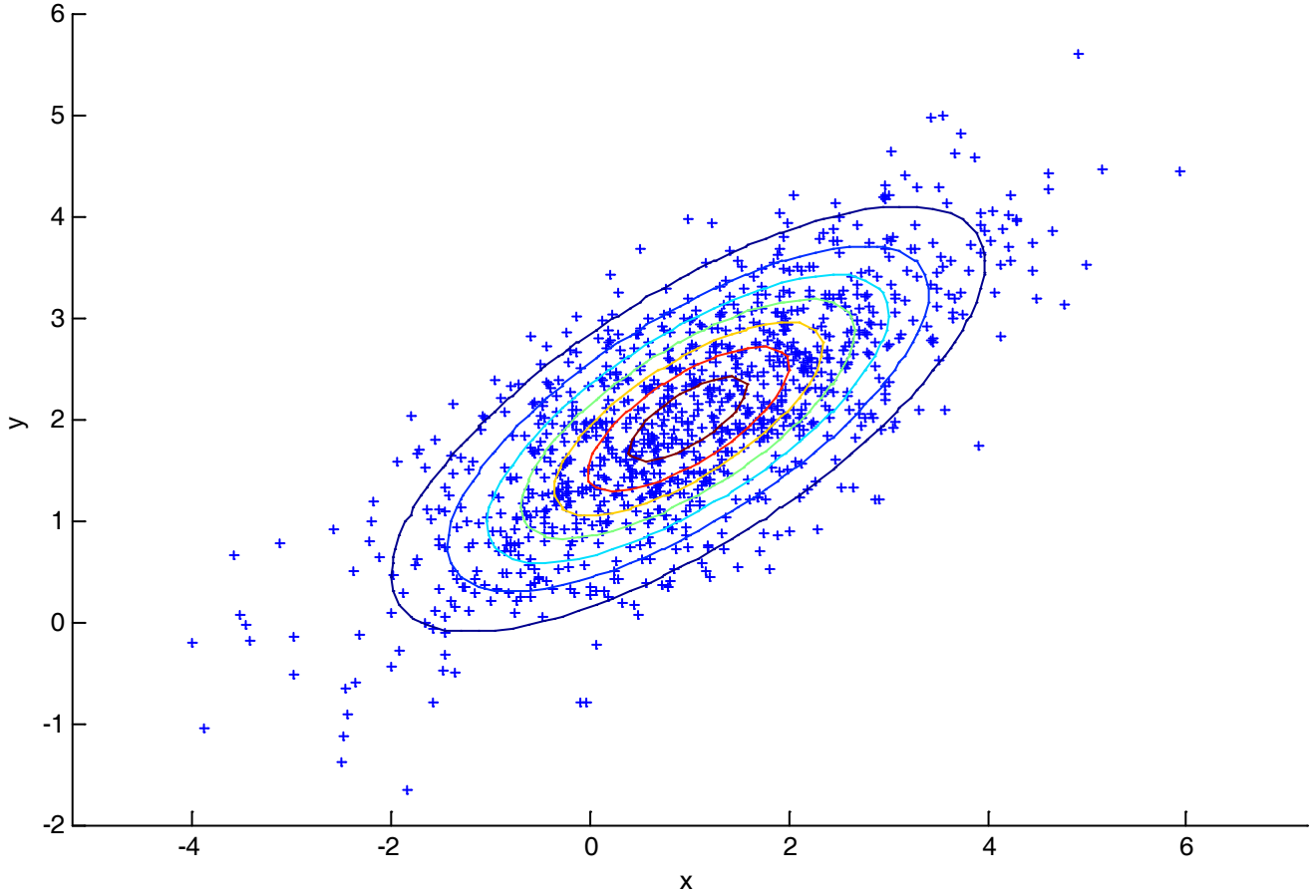
Estimation of Gaussian with multiple of unit covariance matrix

Gaussian Example



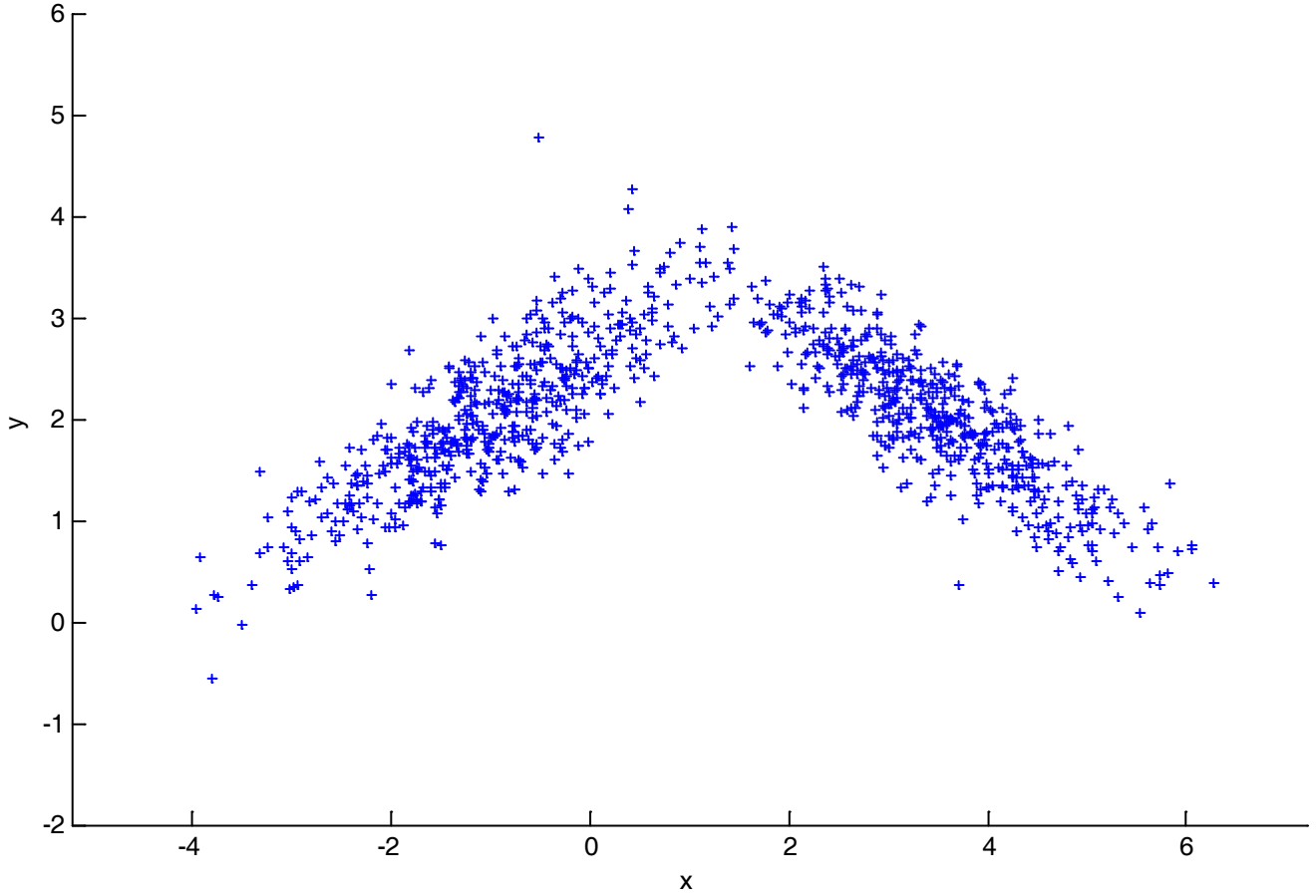
Estimation of Gaussian with diagonal covariance matrix

Gaussian Example



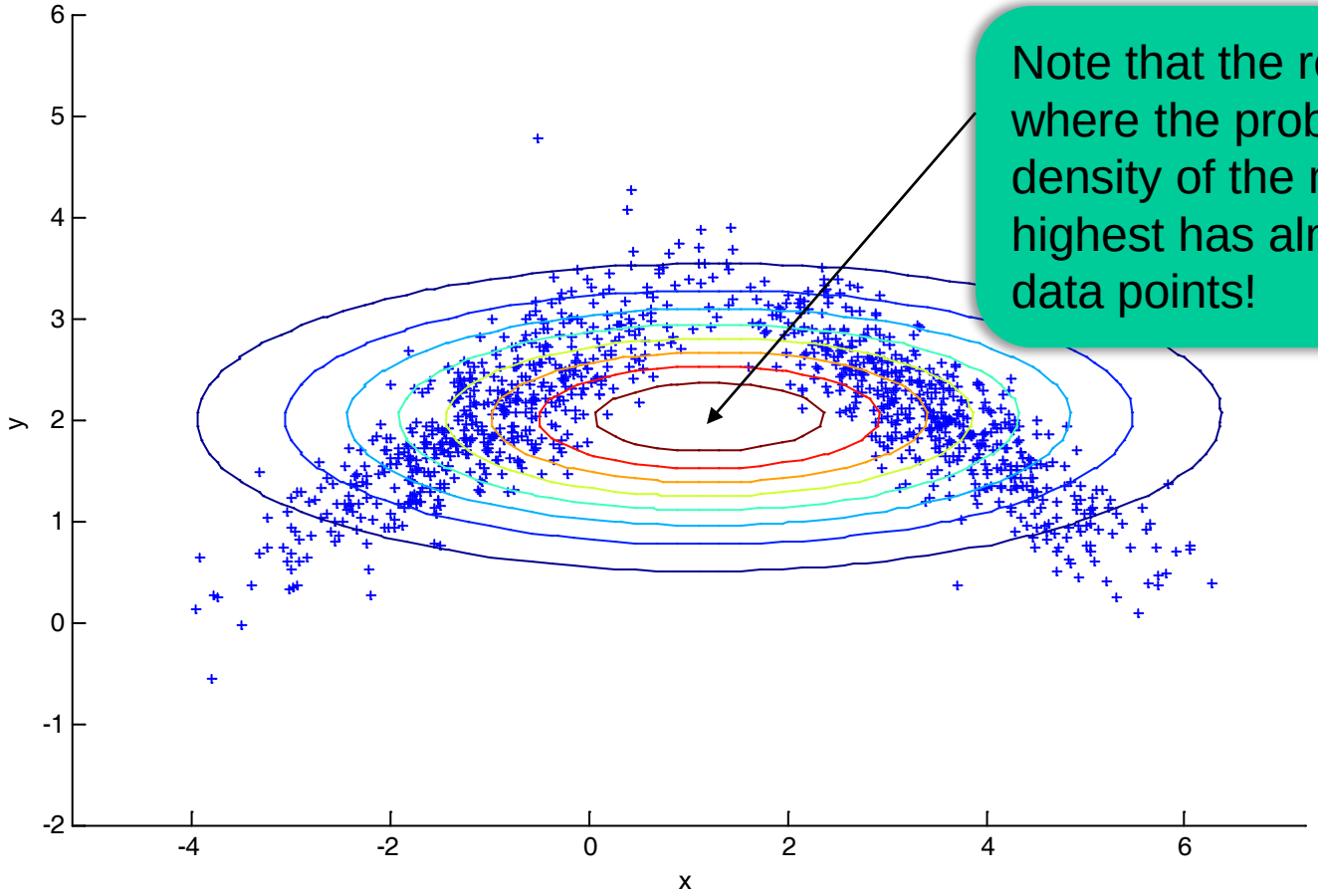
Estimation of Gaussian with full covariance matrix

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)

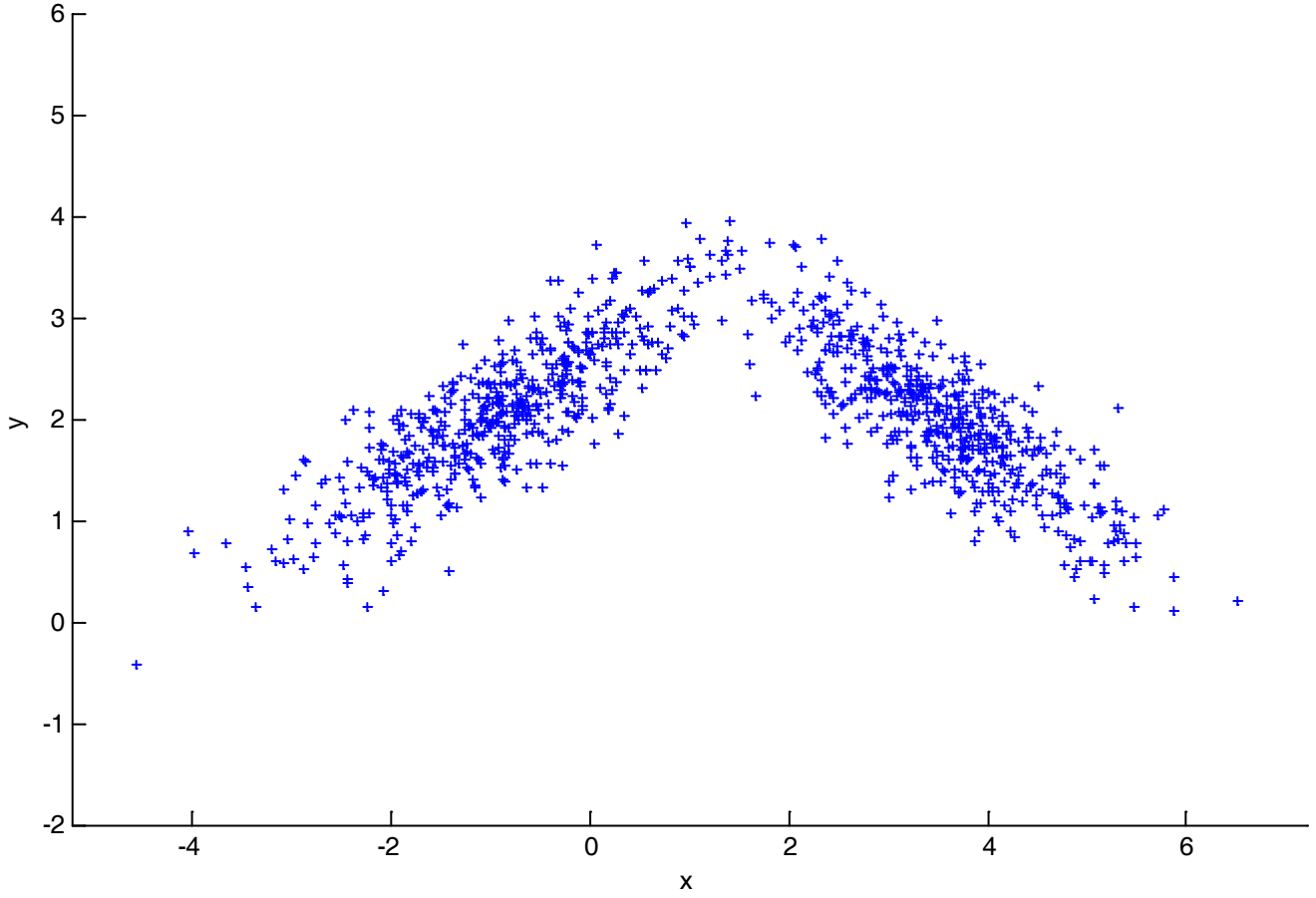
- Approximate true density function using a weighted sum of several Gaussians

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

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

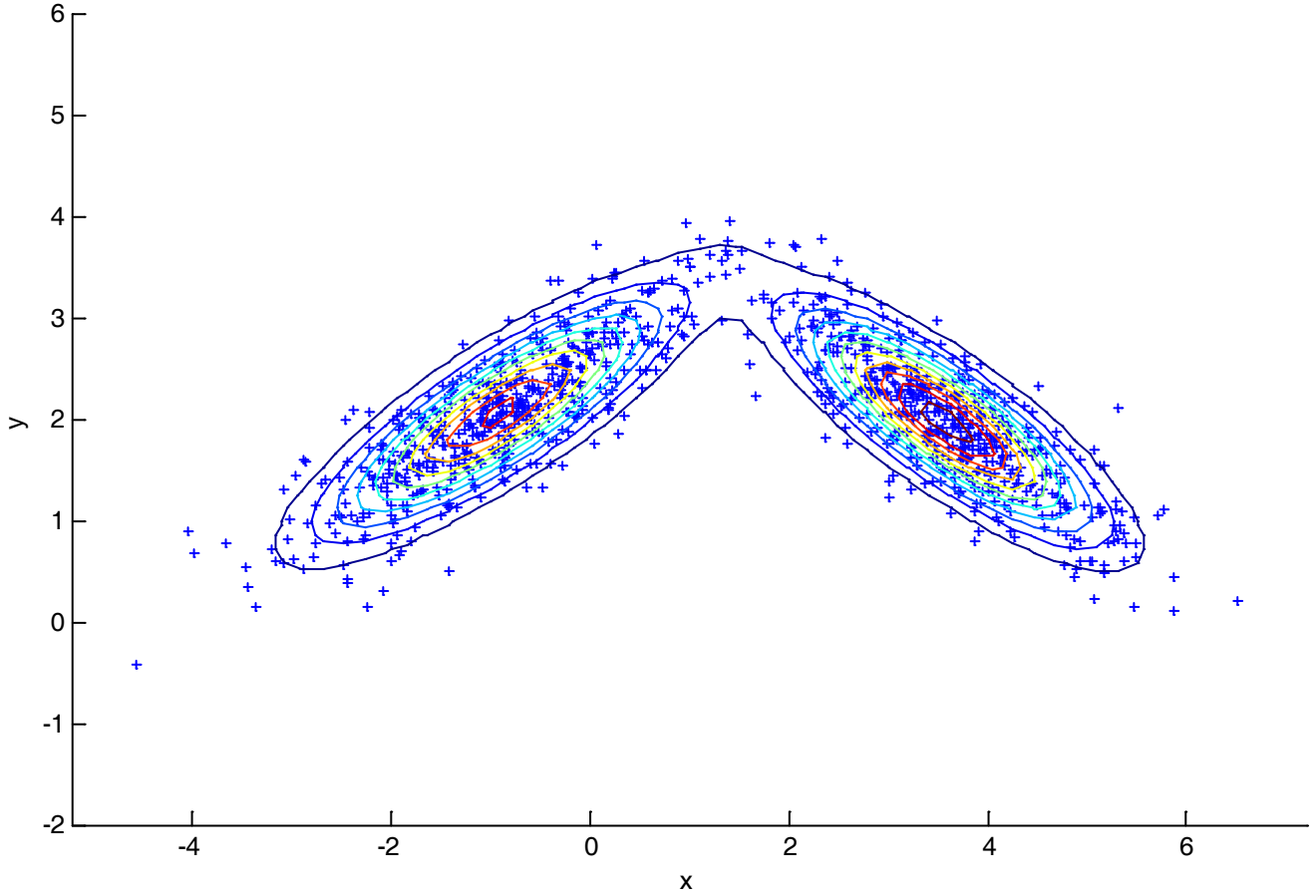
- Any density can be approximated this way with arbitrary precision
  - But might need many Gaussians
  - ➔ Difficult to estimate many parameters
    - Restrict covariance matrices as before
    - Only one shared covariance matrix for all Gaussians
- Now need to estimate parameters of the Gauss functions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

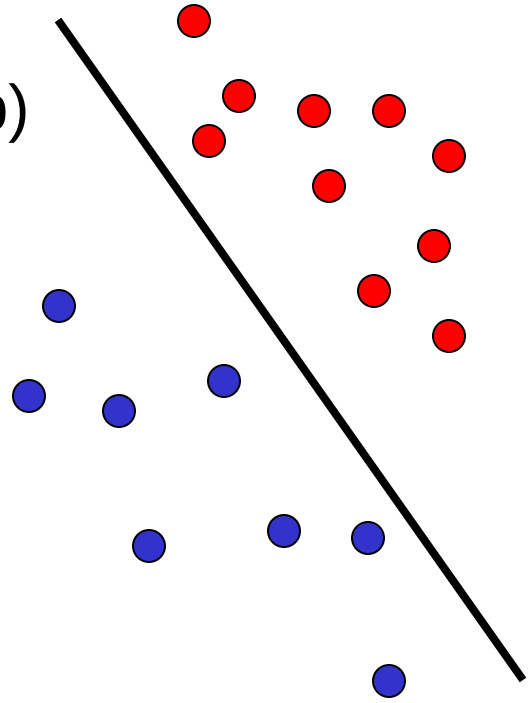
- Gaussian and GMMs are called parametric classifiers
  - They assume a specific form of probability distribution with some parameters
  - Then only the parameters need to be estimated
- There are also methods which do not assume a specific form of probability distribution
  - They are called non-parametric
  - 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

- Simplest possible discriminative classifier
- $F(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b) = \text{sign}(\Sigma\, w_i \cdot x_i + b)$
- Hyperplane dividing feature space
- First, a notational trick
  - Define $w' = [b\ w]^T$ and $x' = [1\ x]^T$
  - $\mathbf{w}^T\mathbf{x} + b$ then becomes $\mathbf{w'}^T\mathbf{x'}$
  - This makes the math a bit easier
- y_i is the label of $\mathbf{x}_i$
  → Binary classification: $y_i \in \{-1, 1\}$
- How to find **w** and b (or **w'**)?
  - First and easiest solution: Perceptron learning algorithm

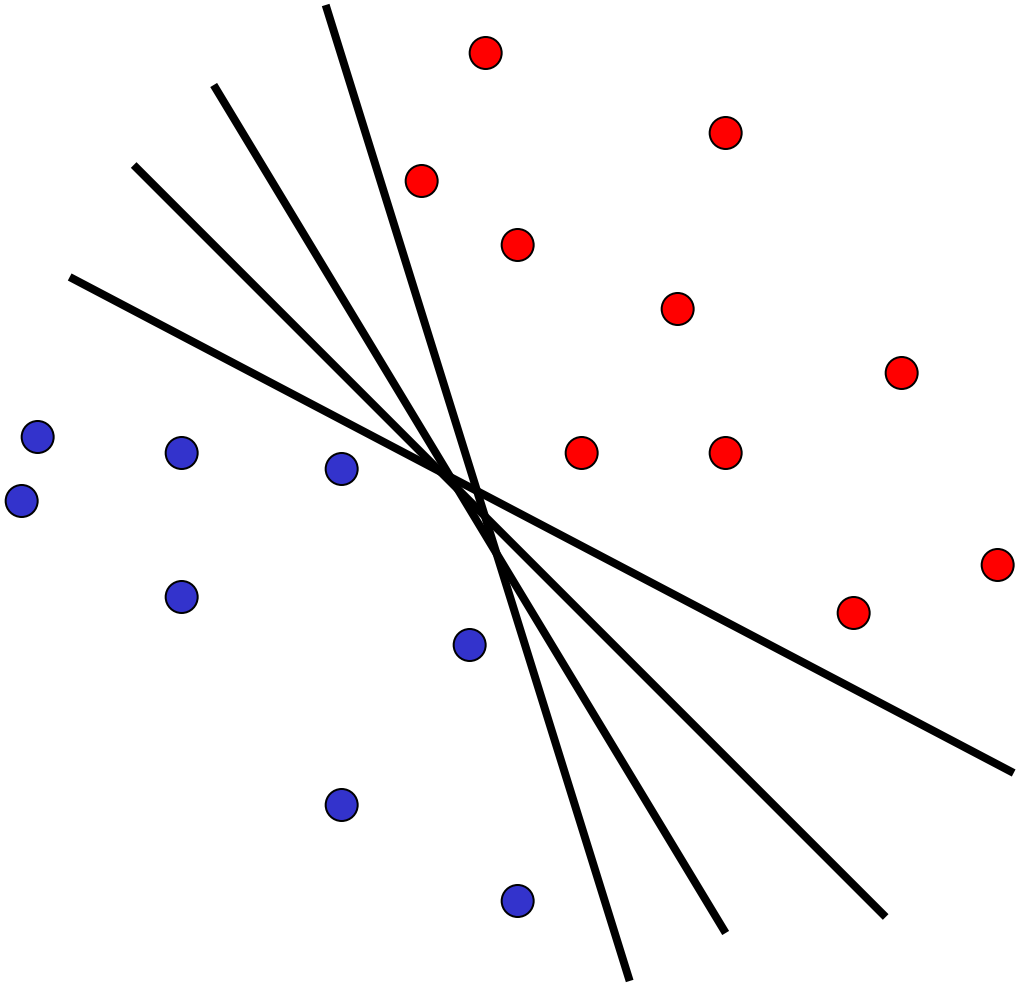


Perceptron learning

- Perceptron Algorithm
  - Start with initial $\mathbf{w_0}$ (usually random or zero)
  - Classify all training samples using $\hat{y}_i = \text{sign}(\mathbf{w}^T x_i)$
  - For all misclassified training samples adjust $\mathbf{w}$:
    - $\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot y_i \cdot x_i$
      - η is the learning rate
  - Repeat until training samples correctly classified
- Demo applet:
  - http://isl.ira.uka.de/neuralNetCourse/2004/VL_11_5/Perceptron.html

Which hyperplane is best?

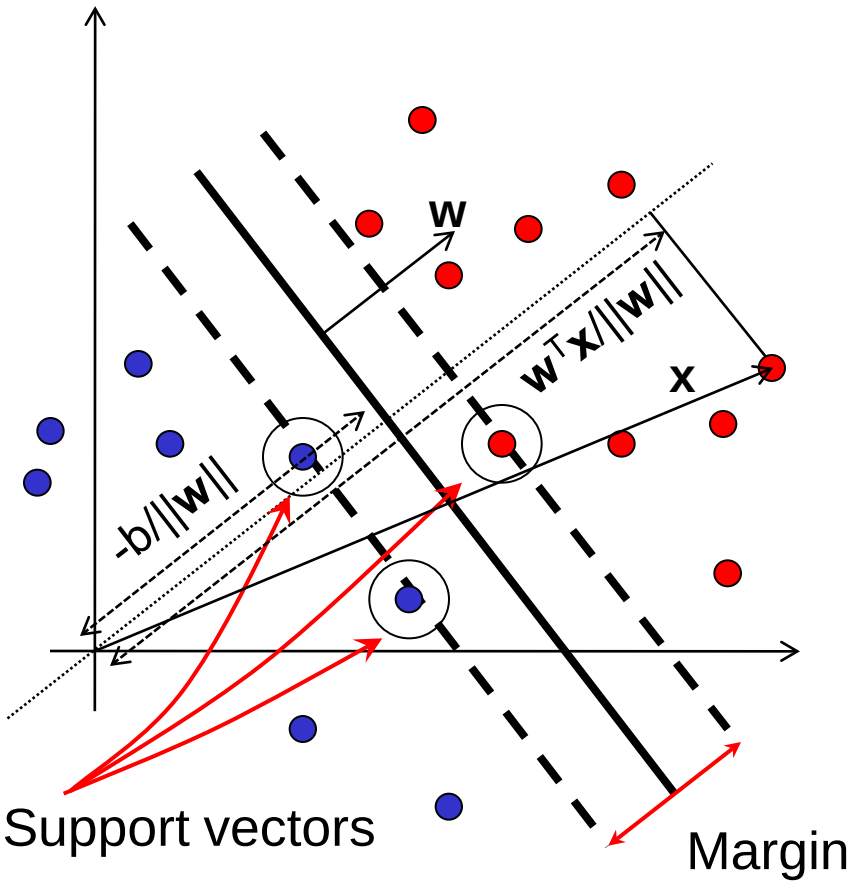
- Perceptron algorithm finds any hyperplane that separates the data



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

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

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?
$1 / ||\mathbf{w}||$

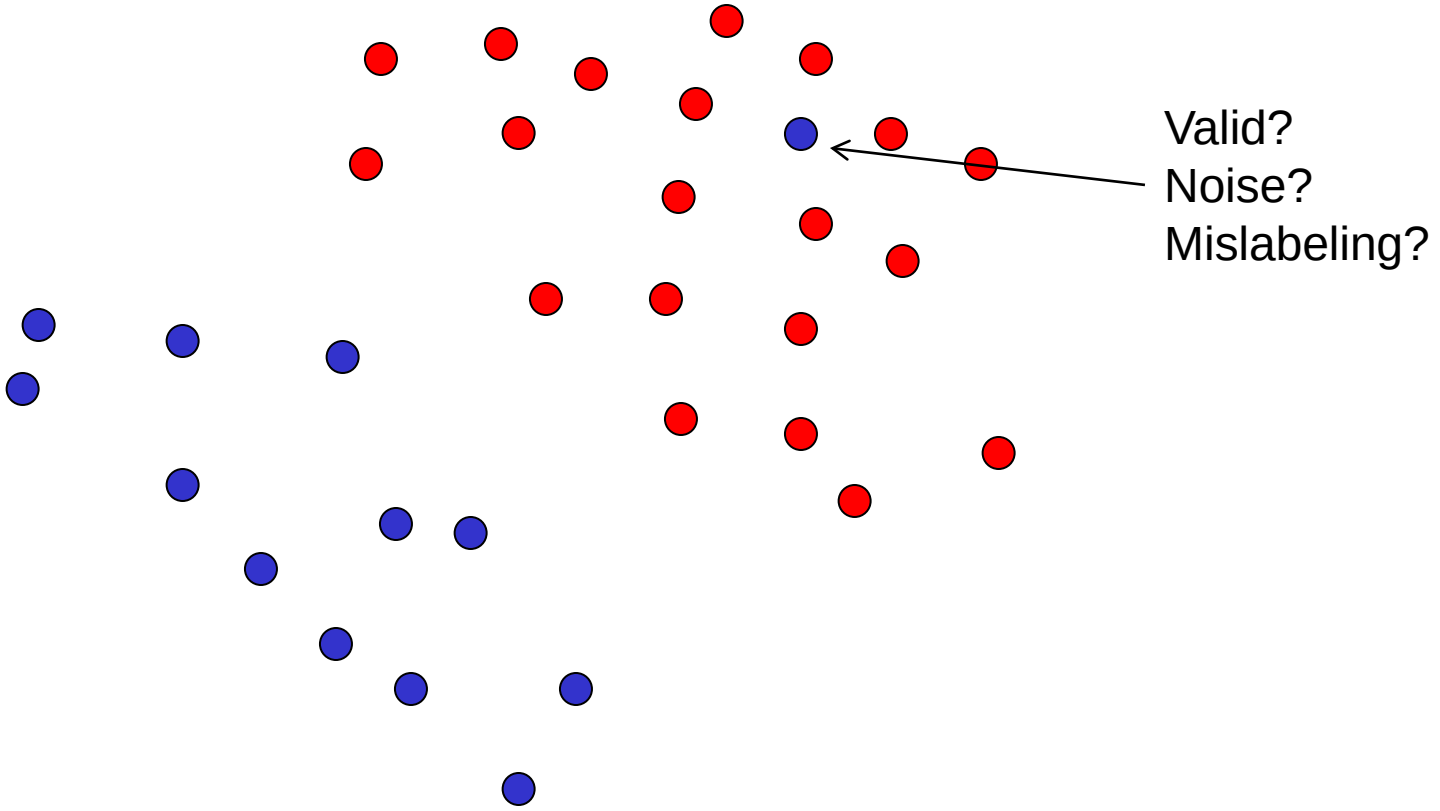
Margin?
$2 / ||\mathbf{w}||$

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

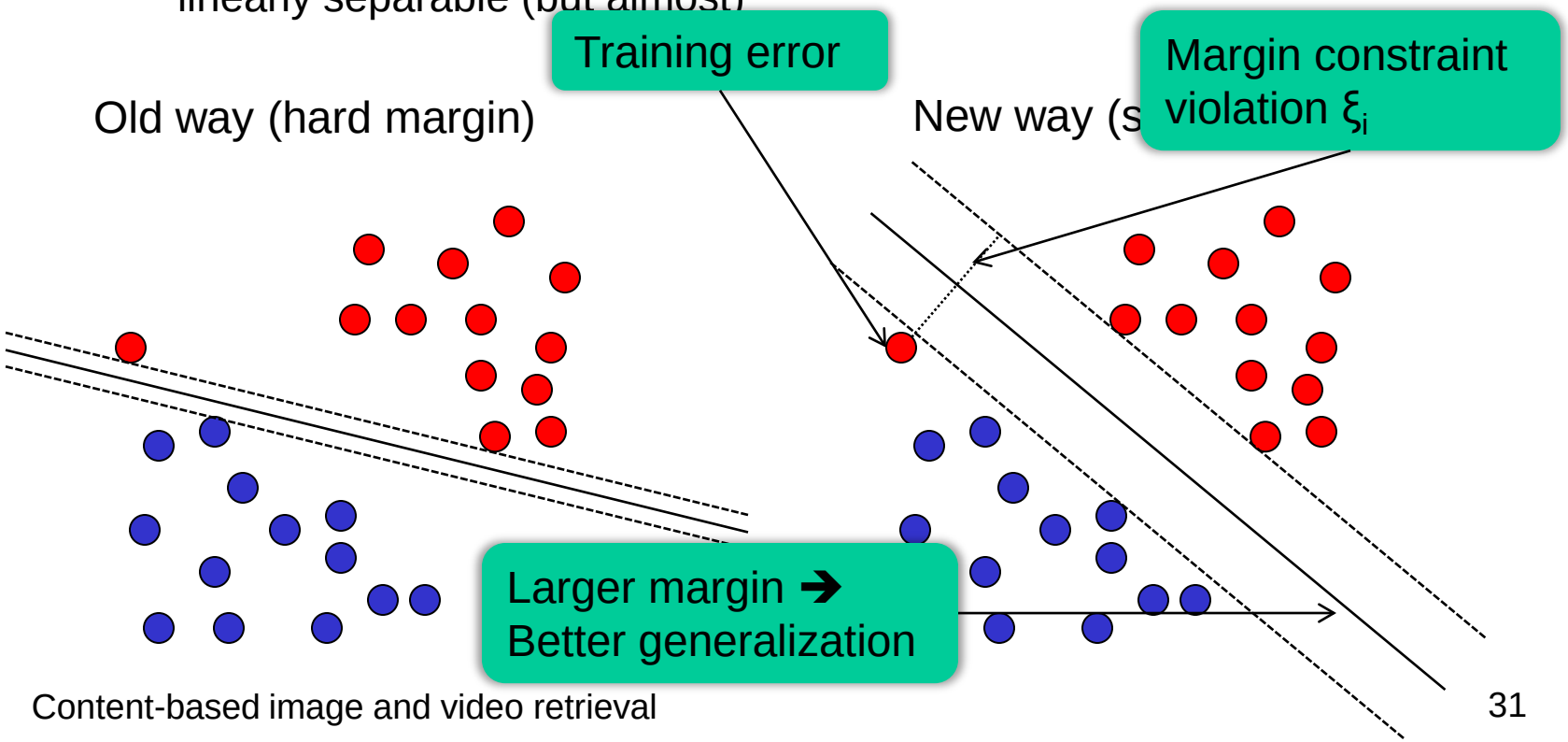
Non-separable data

- 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)

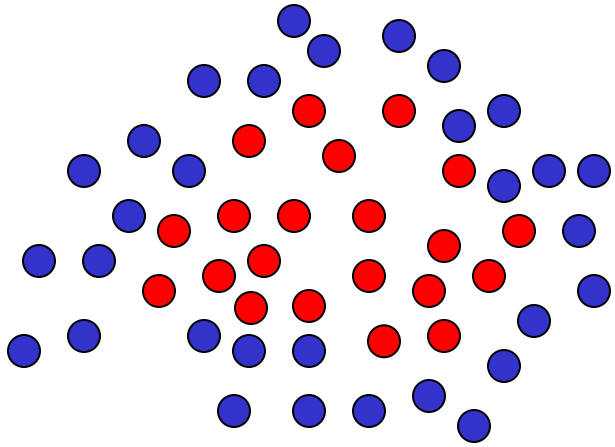


Soft-margin SVM formulation

- 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

Really non-separable data

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

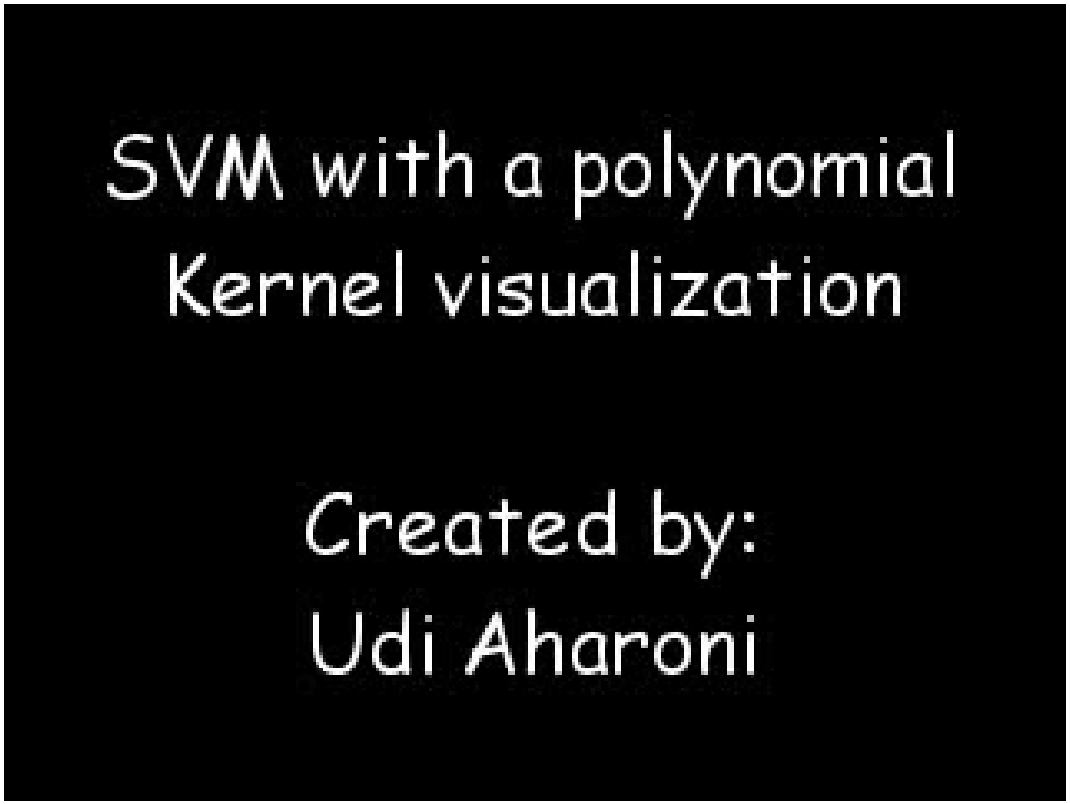


No hyperplane will be able to perform well On this kind of data!

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

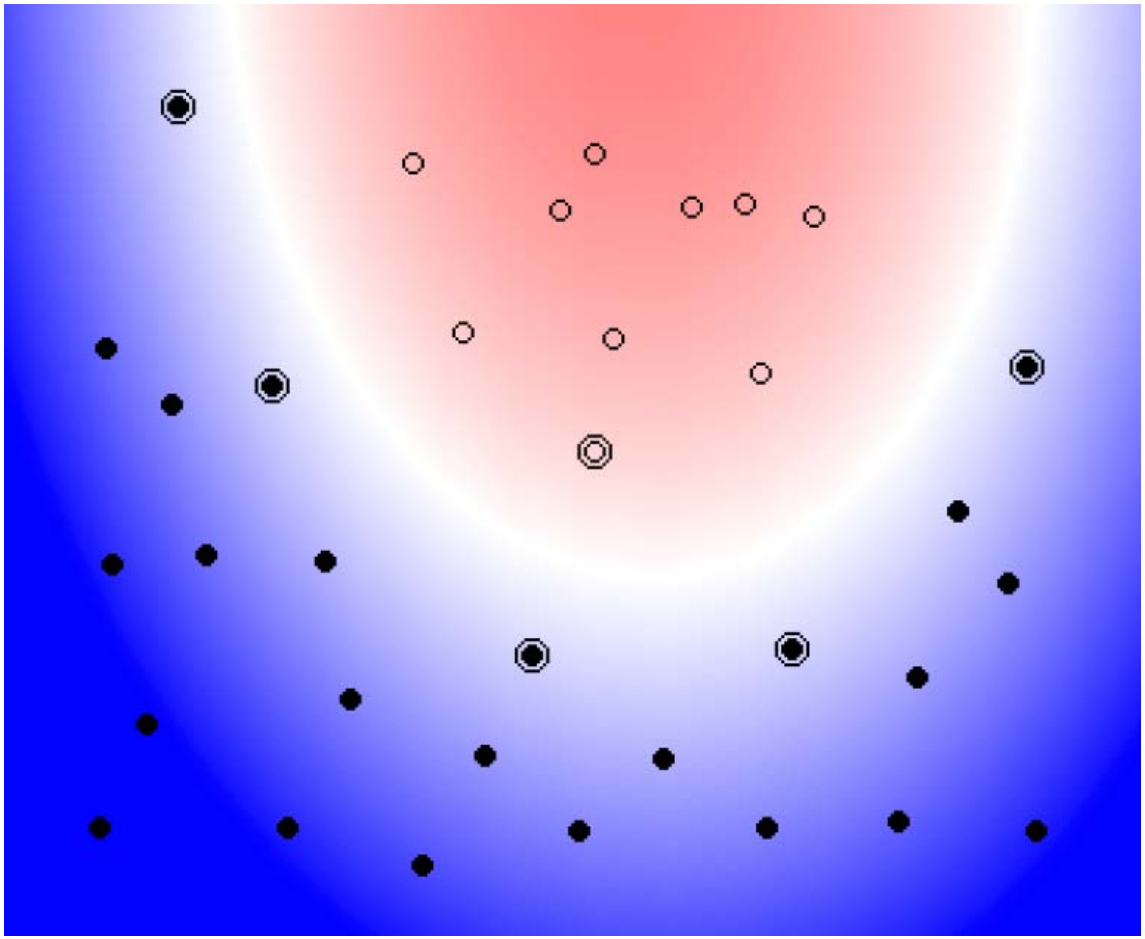
(for mathematical details: See e.g. Burges 1998)

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

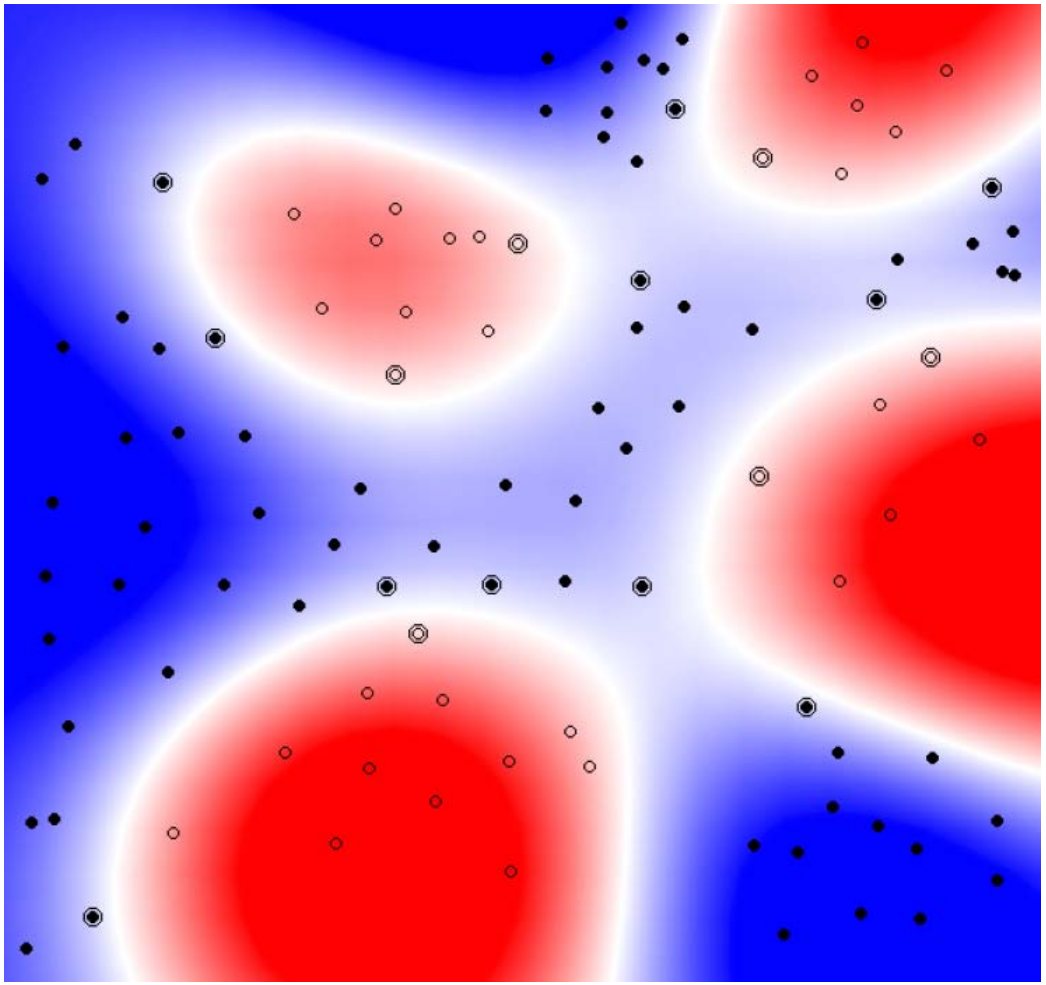
Kernel SVM example

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

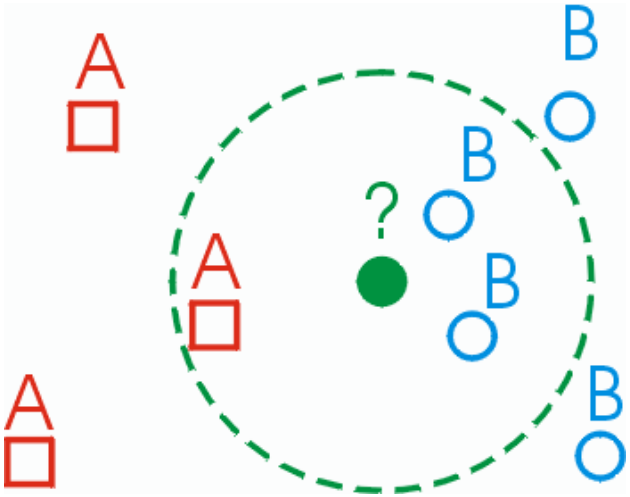
- Normalization of feature vectors very important
  - Normalize each feature separately
    - Zero mean, unit variance or [-1; 1] are popular
  - Normalize feature vector to unit norm
    - $\mathbf{x}' = \mathbf{x} / ||\mathbf{x}||$
- Otherwise performance will suffer
  - Training will be much slower 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

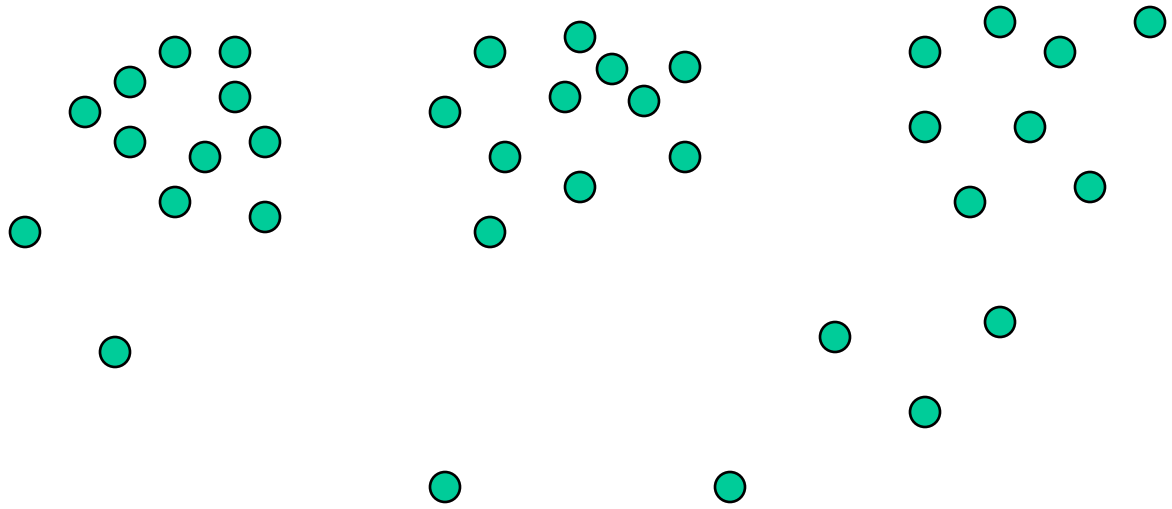
k-nearest-neighbors (KNN)

- Look at the k closest training samples and assign the most frequent label among them
- Model consists of all training samples
  - Pro: No information is lost
  - Con: A lot of data to manage
  - Naïve implementation: compute distance to each training sample every time
- Distance metric is needed
  - Important design parameter
  - L_1, L_2, L_∞, Mahalanobis, …
  - Problem-specific distances
- kNN often good classifier, but:
  - Needs enough data
  - Scalability issues



Clustering

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



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