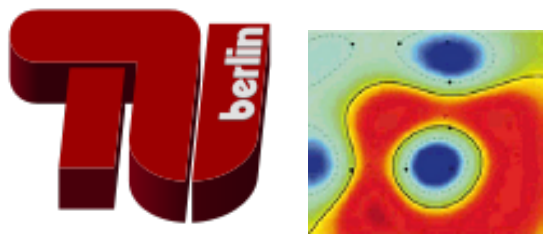
Lecture 0: Introduction

Machine Learning - Theory and Applications



TU Berlin – SoSe 2016

Bayesian Decision Theory

Known:

- Classes $\{\omega_1, \ldots, \omega_c\}$
- Class probabilities $P(\omega_j)$
- Observation $\boldsymbol{x} \in \mathbb{R}^d$
- Likelihood function $P(\boldsymbol{x}|\omega_j)$

Baye's Rule:

$$p(\omega_j|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|\omega_j)P(\omega_j)}{P(\boldsymbol{x})}$$

Law of total probability:

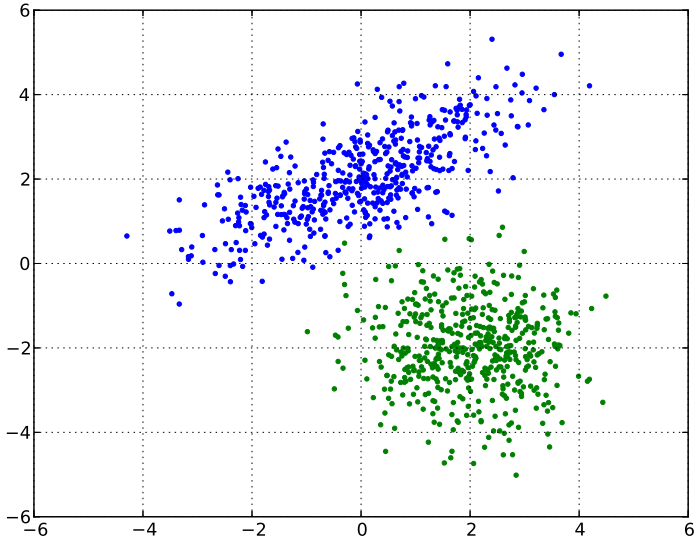
$$p(\boldsymbol{x}) = \sum_{j=1}^{c} p(\boldsymbol{x}|\omega_j)P(\omega_j)$$

Example of Probability Distribution

Normal distribution

$$P(\boldsymbol{x}|\omega_j) = \frac{1}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}_j|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^\top \boldsymbol{\Sigma}_j^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)}$$

Parameters $\theta_j = \{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}$.



Discriminant Functions

Not always interested knowing exactly $p(\omega_j|\boldsymbol{x})$ for all j.

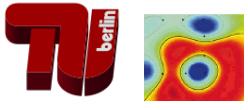
Instead:

$$j^\star = \arg\max_j p(\omega_j|\boldsymbol{x}) = \arg\max_j g_j(\boldsymbol{x}).$$

where $g_j(\boldsymbol{x})$ is a discriminant function.

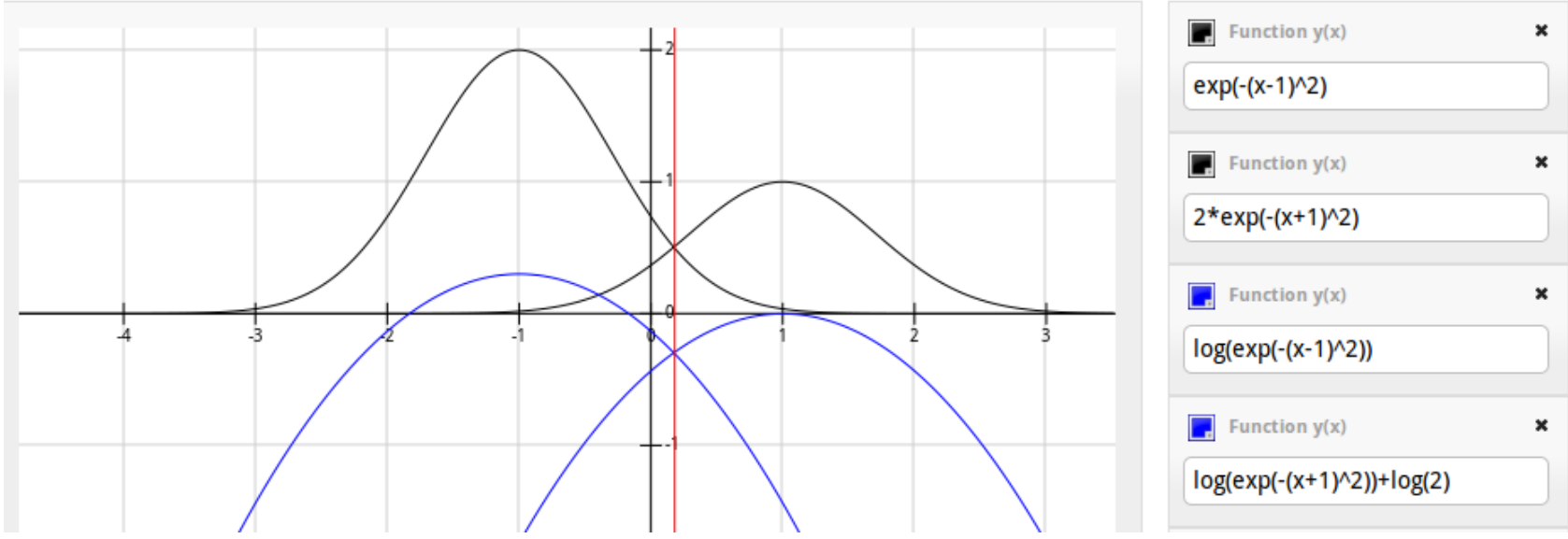
Example of discriminant functions:

- $g_j(\boldsymbol{x}) = p(\omega_j|\boldsymbol{x})$
- $g_j(\boldsymbol{x}) = p(\omega_j|\boldsymbol{x}) + a$
- $g_j(\boldsymbol{x}) = p(\omega_j|\boldsymbol{x}) \cdot \alpha \quad \alpha > 0$
- $g_j(\boldsymbol{x}) = \log p(\omega_j|\boldsymbol{x})$



Discriminant Functions

- $(g_1(\boldsymbol{x}), g_2(\boldsymbol{x}))$
- $(\log g_1(\boldsymbol{x}), \log g_2(\boldsymbol{x}))$



(plotted with www.fooplot.com)

Observation: Both pairs of discriminants meet at the same point on the x-axis (shown by the red line).

Discriminative Learning

Example: Perceptron

- Labeled dataset:

$$\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{N} \qquad \boldsymbol{x}_i \in \mathbb{R}^d \qquad y_i \in \{1, \ldots, c\}$$

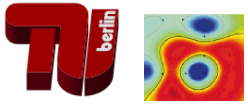
- Discriminants:

$$g_j(\boldsymbol{x}) = \boldsymbol{w}_j^\top \boldsymbol{x} - b_j \qquad j \in \{1, \ldots, c\}$$

- Objective:

$$\max_{\{\boldsymbol{w}_j, b_j\}} \quad \sum_{i=1}^{N} \mathbf{1}_{\arg\max_j g_j(\boldsymbol{x}_i) = y_i}$$

Remaining question: What if several different solutions classify the problem perfectly? Which one to choose?



Model Selection

From which function class should we choose g_j ?

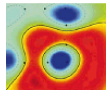
- **Good idea:** Choose function class with low VC-dimension h. Example: Large-margin classifiers

$$h \leq \min\left(\left[\frac{R^2}{\Delta^2}\right], d\right) + 1.$$

where Δ is size of the margin, R is the radius of the minimum enclosing sphere of $\mathcal{D}$ and d is the number of dimensions of the input space.

- **Bad idea:** Choose function class with small number of parameters. Example: $\sin(ax)$ with $a \in \mathbb{R}$ has only one parameter, but can shatter infinitely many data points on $\mathbb{R}$.

$$h \leq \infty.$$

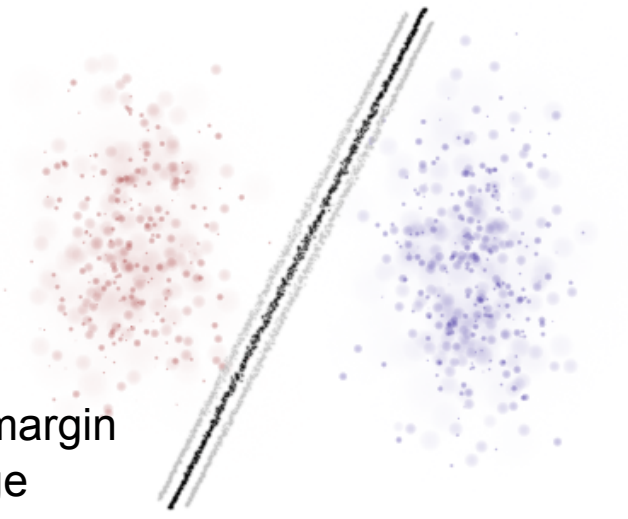


Model Selection

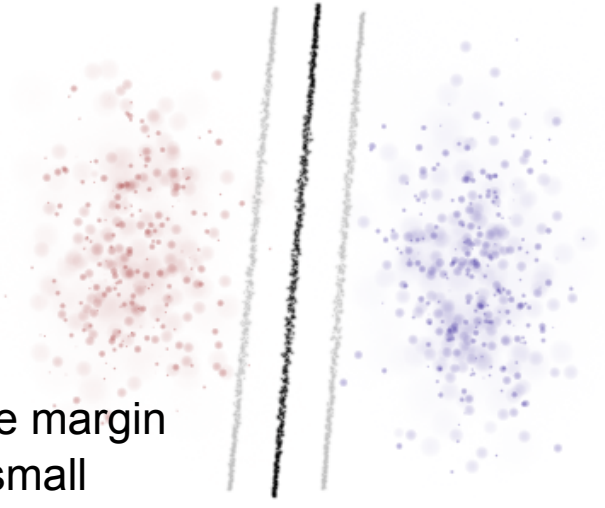
Low VC dimension gives guarantees on generalization error of the model:

$$R[f] \leq R_{\text{emp}}[f] + \sqrt{\frac{h(\log \frac{2N}{h} + 1) - \log(\eta/4)}{N}}$$

with probability $1 - \eta$.



small margin
- h large
- high risk



large margin
- h small
- low risk

Enforcing Large Margin In Practice

Support Vector Machine

- Labeled dataset:

$$\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{N} \qquad \boldsymbol{x}_i \in \mathbb{R}^d \qquad y_i \in \{-1, 1\}$$

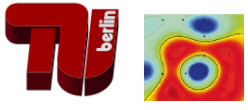
- Classifier:

$$g_2(\boldsymbol{x}) - g_1(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} - b$$

- Objective:

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \qquad \text{subject to} \qquad (\boldsymbol{w}^\top \boldsymbol{x}_i - b) \cdot y_i \geq 1$$

Various techniques to optimize SVMs: quadratic optimization, subgradient methods.



NonLinear Classification via Kernelization

Step 1: Define *Lagrangian*:

$$L(\boldsymbol{w}, \boldsymbol{\alpha}) = \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2}_{\text{objective}} + \sum_{i=1}^{n} \alpha_i \cdot \underbrace{[1 - (\boldsymbol{w}^\top \boldsymbol{x}_i - b) \cdot y_i]}_{(>0) \Rightarrow \text{ constraint violation}}$$

Minimize with $\boldsymbol{w}, b$ under maximally penalized constraints violations $(\max_{\boldsymbol{\alpha} \succeq \mathbf{0}})$:

$$\begin{aligned}
\min_{\boldsymbol{w},b} \max_{\boldsymbol{\alpha} \succeq \mathbf{0}} L(\boldsymbol{w}, \boldsymbol{\alpha}) &= \max_{\boldsymbol{\alpha} \succeq \mathbf{0}} \min_{\boldsymbol{w},b} L(\boldsymbol{w}, \boldsymbol{\alpha}) \\
&= \max_{\boldsymbol{\alpha} \succeq \mathbf{0}} \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \, \alpha_j \, y_i \, y_j \, \boldsymbol{x}_i^\top \boldsymbol{x}_j \\
&\quad \text{with } \textstyle\sum_i \alpha_i \, y_i = 0
\end{aligned}$$

NonLinear Classification via Kernelization

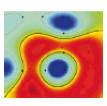
Step 2: Replace $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ by a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ that must satisfy

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0 \qquad \text{for all} \quad c_i, c_j \in \mathbb{R}$$

If $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a kernel, then, there exists a feature map $\phi(\boldsymbol{x})$, possibly infinite-dimensional such that

$$k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}')$$

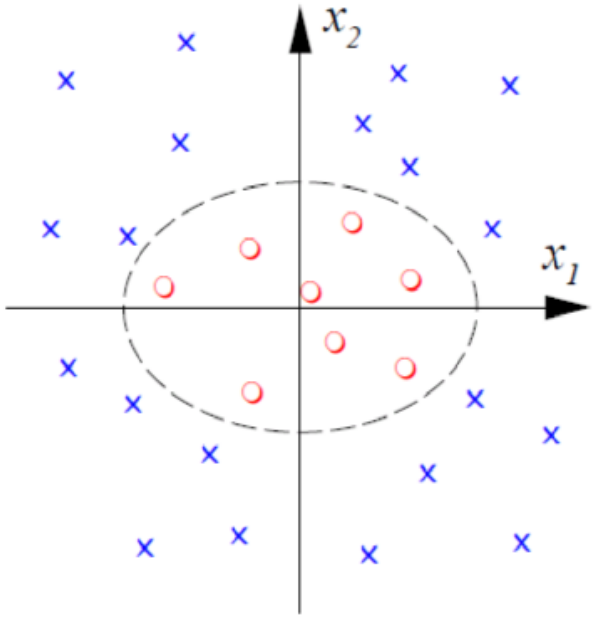
Therefore, a kernel SVM is equivalent to a nonlinear SVM built directly on the nonlinear representation $\phi(\boldsymbol{x})$.

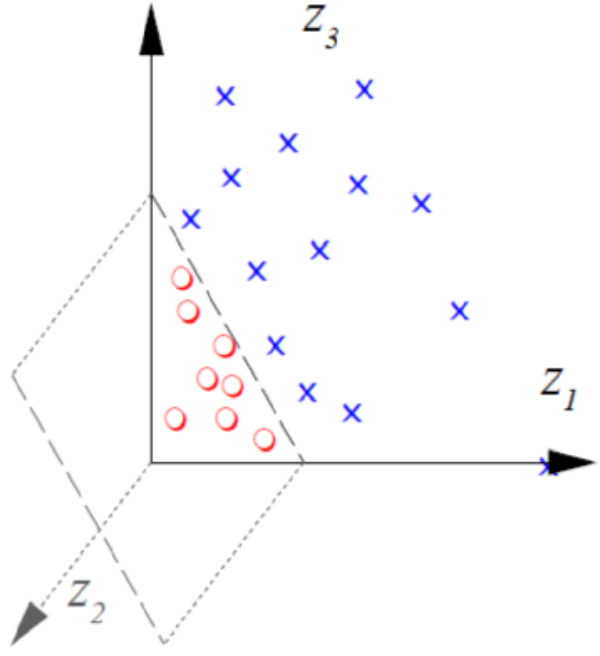


NonLinear Classification via Kernelization

Example: all second order monomials

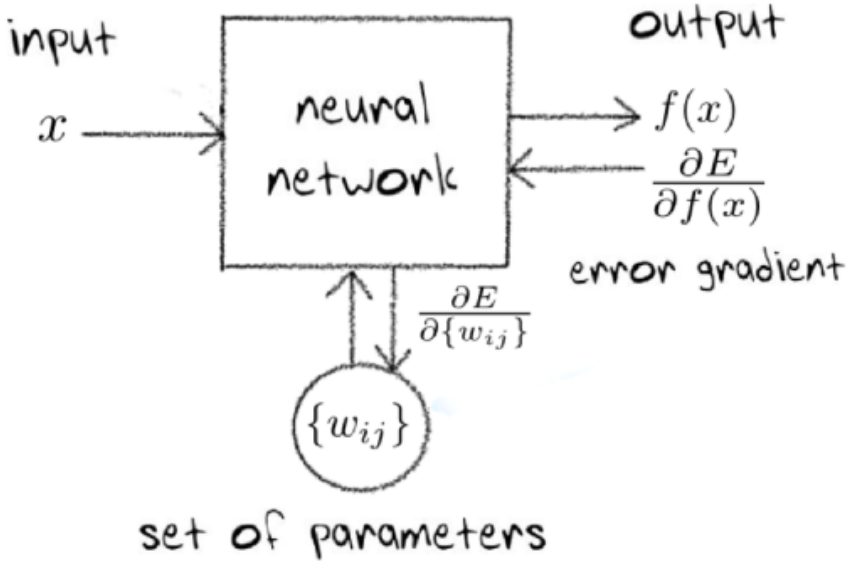
$$\begin{aligned} \Phi : \mathbf{R}^2 &\rightarrow \mathbf{R}^3 \\ (x_1, x_2) &\mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2) \end{aligned}$$

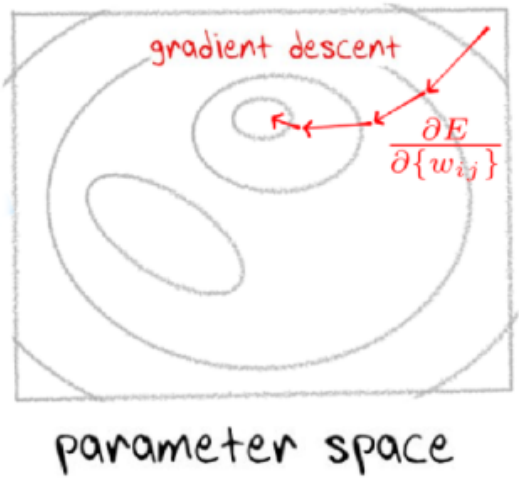




Another Approach to NonLinear Learning

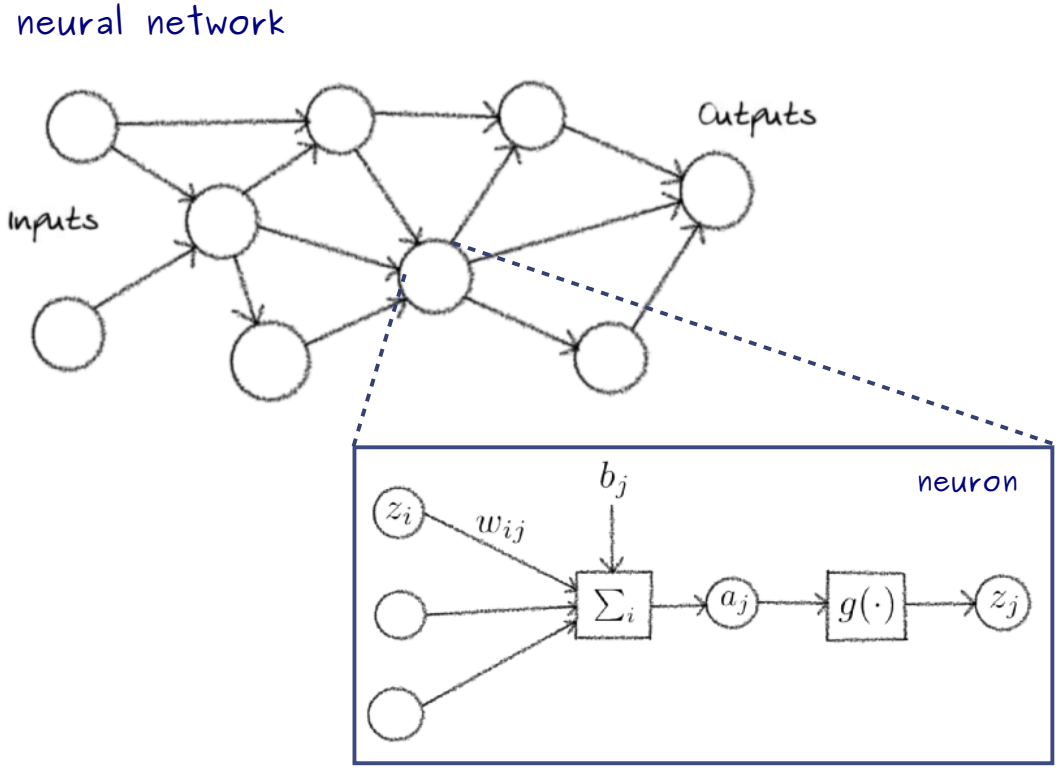
Replace the linear discriminant $g_2(\boldsymbol{x}) - g_1(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} - b$ by a more complex nonlinear function composed of many trainable parameters, and for which a gradient can be computed easily. Example: neural networks.





Neural Networks

A neural network is a graph of many simple interconnected computational units (neurons), that together form a complex function.



Gradient is obtained using the error backpropagation algorithm (an efficient application of the chain rule in a graph).