Elements of Machine Learning & Data Science

Winter semester 2025/26

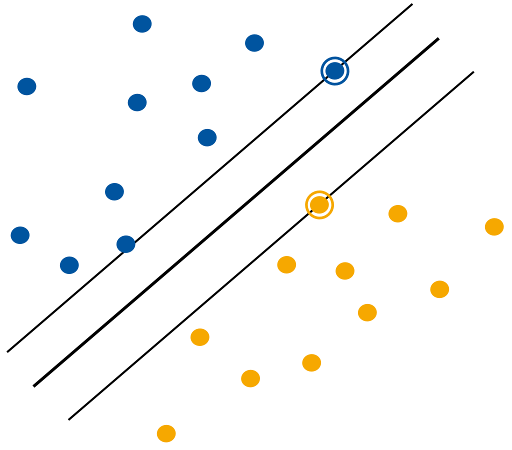
Lecture 15 – Support Vector Machines II

15.12.2025

Prof. Bastian Leibe

Machine Learning Topics

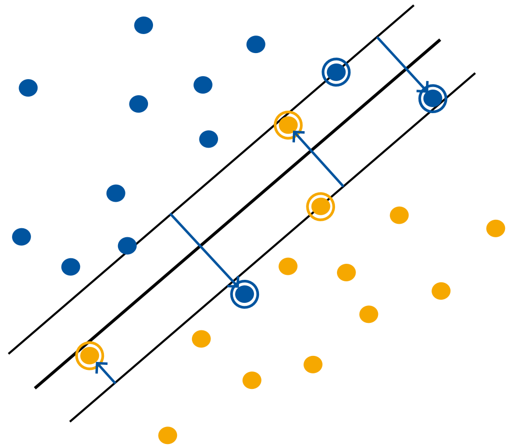
8. Introduction to ML
9. Probability Density Estimation
10. Linear Discriminants
11. Linear Regression
12. Logistic Regression
13. **Support Vector Machines**
14. Neural Network Basics



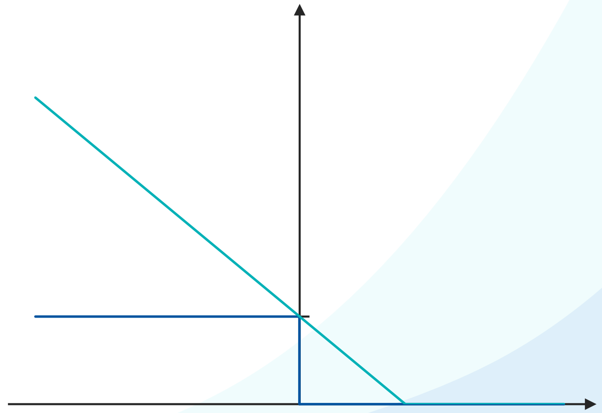
Maximum Margin Classification

$$L_p(\mathbf{w}, b, \mathbf{a})$$
$$L_d(\mathbf{a})$$

Primal & Dual Form



Soft-Margin SVM



Hinge Loss

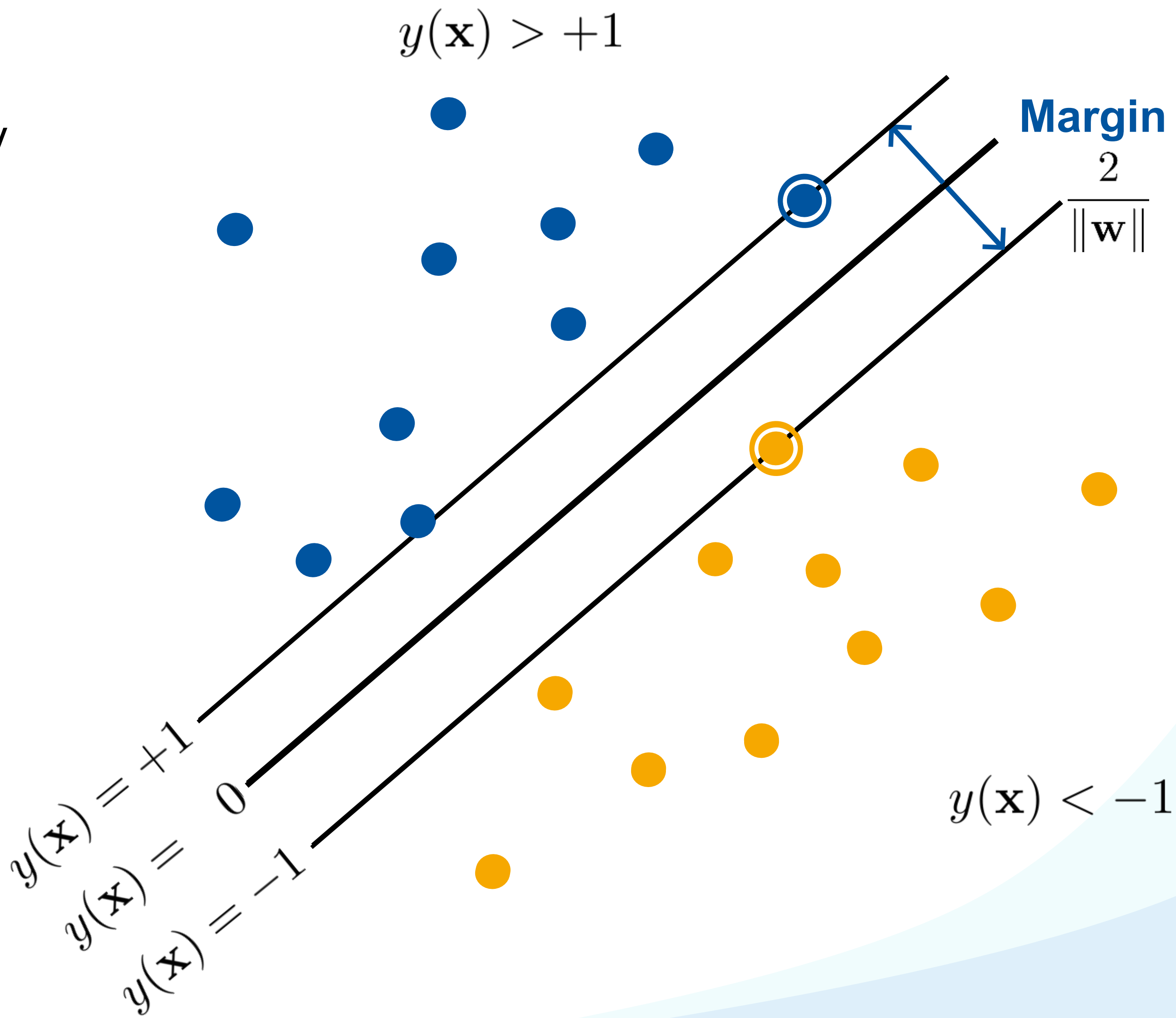
Recap: Maximum Margin Classification

- Intuitively, we want to choose the classifier which leaves maximal "safety room" for future data points.
- This classifier has the largest margin between positive and negative points.
- We can rescale $\mathbf{w}$ such that the distance of the points on the margin to the decision boundary is exactly 1.

$$t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) = 1$$

- If the data is linearly separable, then for all points, the following must hold:

$$t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 \quad \forall n$$



- Optimization problem
  - Find the hyperplane with maximum margin by optimizing:

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

“Maximize the margin”

such that

$$t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 \quad \forall n$$

“such that each point is on the correct side of the margin”

- This is a quadratic programming problem with linear constraints.

Recap: Primal SVM Formulation

- Recall the SVM objective:

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{such that} \quad t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 \quad \forall n$$

- We introduce positive Lagrange multipliers $a_n \geq 0$ and get the primal form of SVMs:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left[t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1\right]$$

Necessary and sufficient conditions:

$$a_n \geq 0$$
$$t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1 \geq 0$$
$$a_n[t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1] = 0$$

KKT conditions:

$$\lambda \geq 0$$
$$f(\mathbf{x}) \geq 0$$
$$\lambda f(\mathbf{x}) = 0$$

Recap: Dual Form of the SVM Objective

- We can equivalently reformulate the SVM to maximize

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\mathbf{x}_m^{\mathsf{T}} \mathbf{x}_n)$$

under the conditions

$$a_n \geq 0 \quad \forall n$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

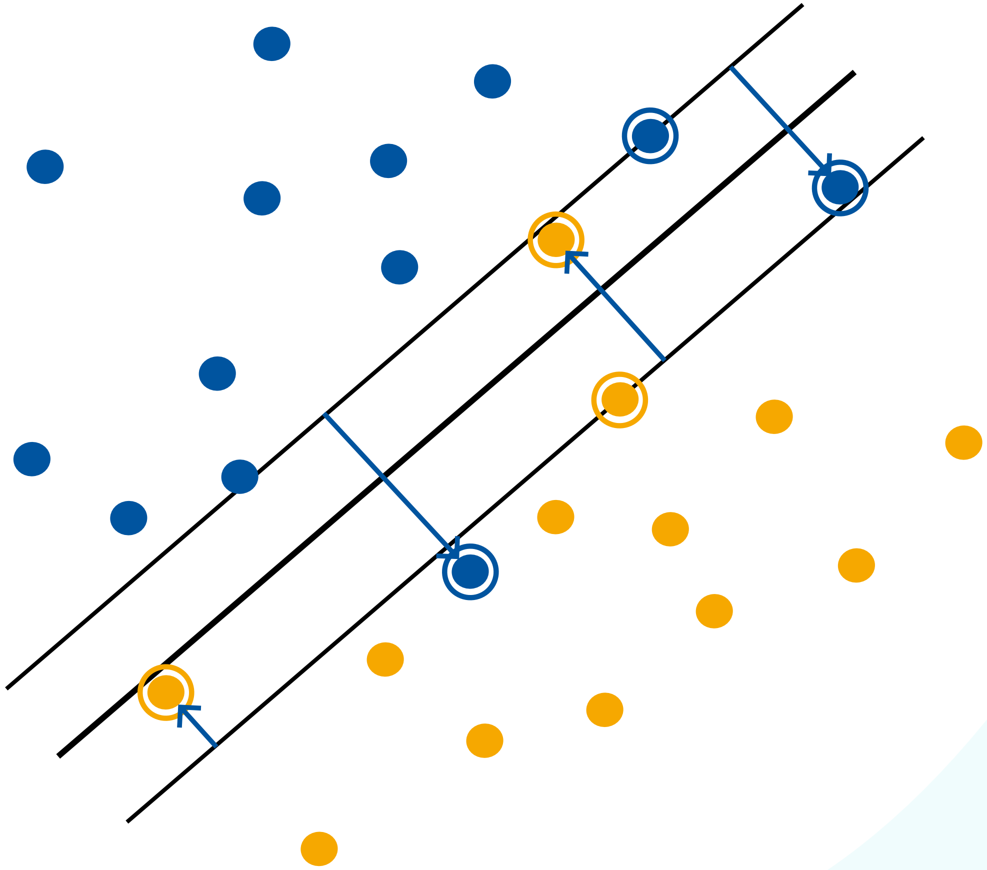
- We now have an optimization problem in N variables.

$\Rightarrow$ Complexity: $\mathcal{O}(N^3)$

For the derivation, please watch the video

Support Vector Machines

1. Maximum Margin Classification
2. Primal Formulation
3. Dual Formulation
4. **Soft-Margin SVMs**
5. Non-linear SVMs
6. Error Function Analysis

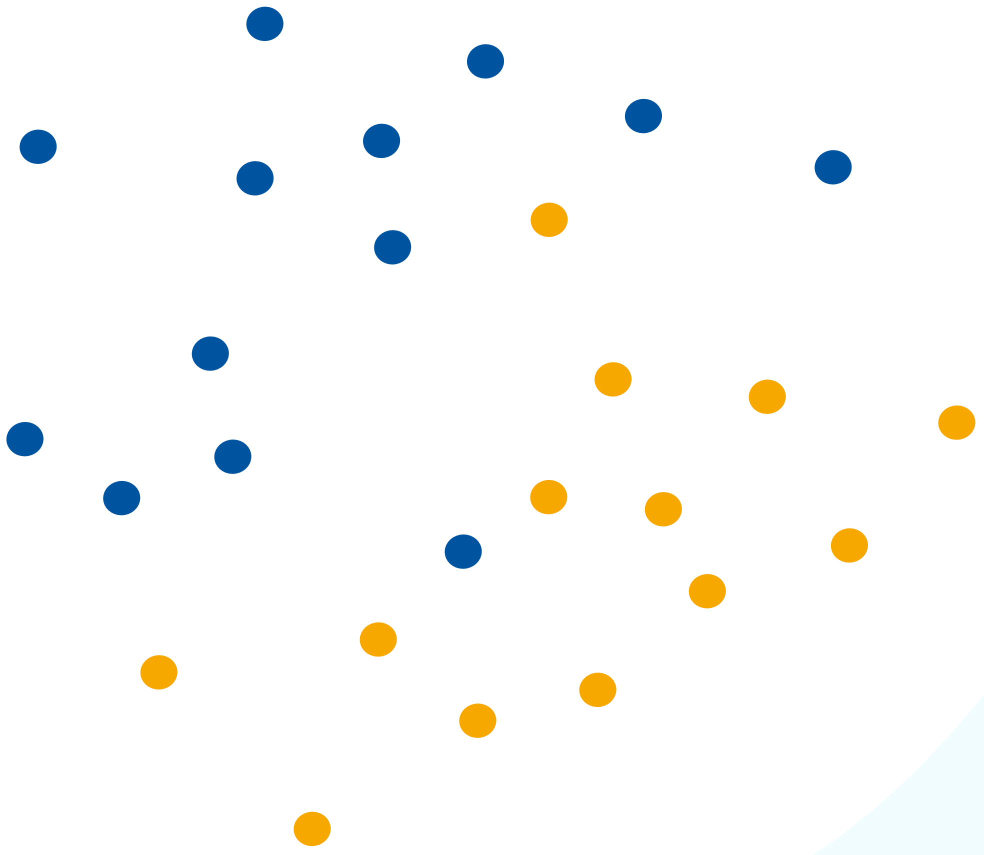


Soft-Margin SVM

- So far, we assumed linearly separable data.
  - Our current formulation has no solution if the data are not linearly separable!

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2,$$

such that $$t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 \quad \forall n$$



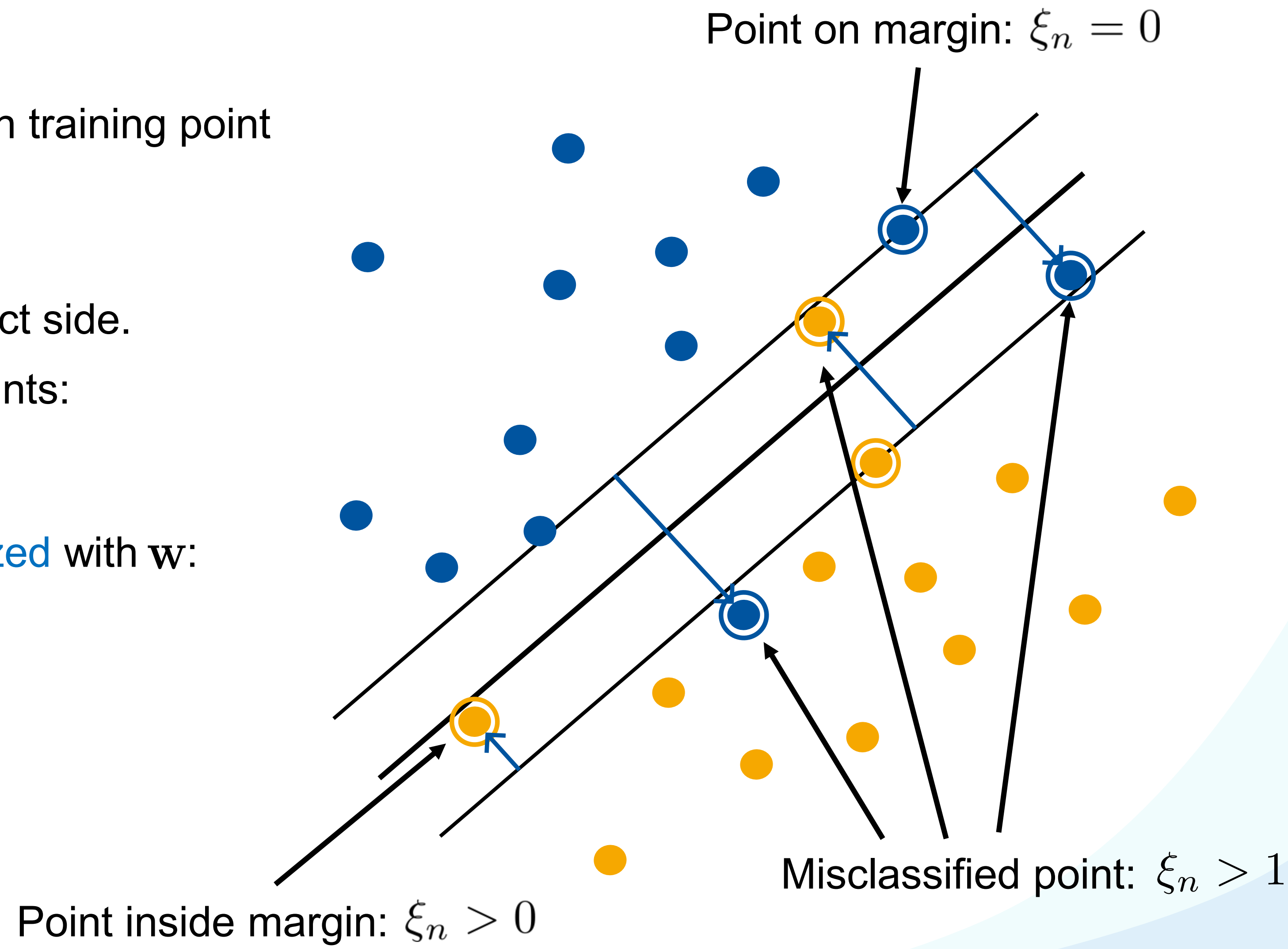
- Need to introduce tolerance to outlier data points.
  - The resulting model is called soft-margin SVM.

Recap: Soft-Margin SVM with Slack Variables

- Slack variables
  - One slack variable ξ_n for each training point
- Effect
  - $\xi_n = 0$ for points on the correct side.
  - Linear penalty for all other points:
    $\xi_n = |t_n - y(\mathbf{x}_n)|$
- Slack variables are jointly optimized with $\mathbf{w}$:

$$\min \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{n=1}^{N} \xi_n$$

where C is a tradeoff parameter.



Recap: New Primal Formulation

- Minimize

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \underbrace{\sum_{n=1}^{N} a_n \left[t_n(y(\mathbf{x}_n) - 1 + \xi_n\right]}_{\substack{\text{Constraint} \\ t_n y(\mathbf{x}_n) \geq 1 - \xi_n}} - \underbrace{\sum_{n=1}^{N}\mu_n\xi_n}_{\substack{\text{Constraint} \\ \xi_n \geq 0}}$$

- KKT conditions

$$\begin{aligned} a_n &\geq 0 & \mu_n &\geq 0 \\ t_n y(\mathbf{x}_n) - 1 + \xi_n &\geq 0 & \xi_n &\geq 0 \\ a_n\left[t_n y(\mathbf{x}_n) - 1 + \xi_n\right] &= 0 & \mu_n\xi_n &= 0 \end{aligned}$$

Recap: New Dual Formulation

- Maximize

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\mathbf{x}_m^{\mathsf{T}} \mathbf{x}_n)$$

- Under the side conditions

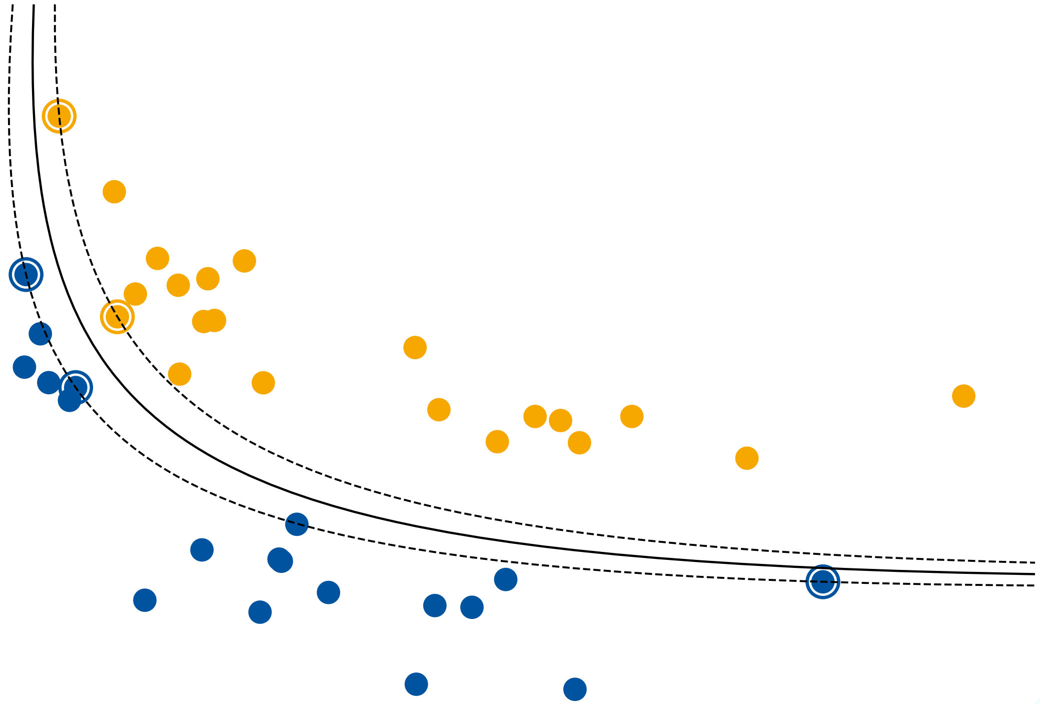
$$\boxed{0 \leq a_n \leq C \quad \forall n}$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

This is the only difference to before.

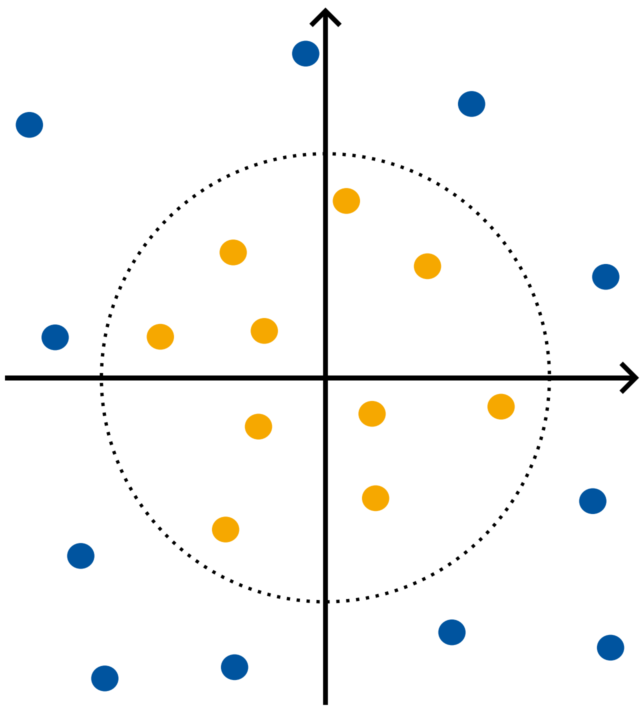
Support Vector Machines

1. Maximum Margin Classification
2. Primal Formulation
3. Dual Formulation
4. Soft-Margin SVMs
5. **Non-Linear SVMs**
6. Error Function Analysis



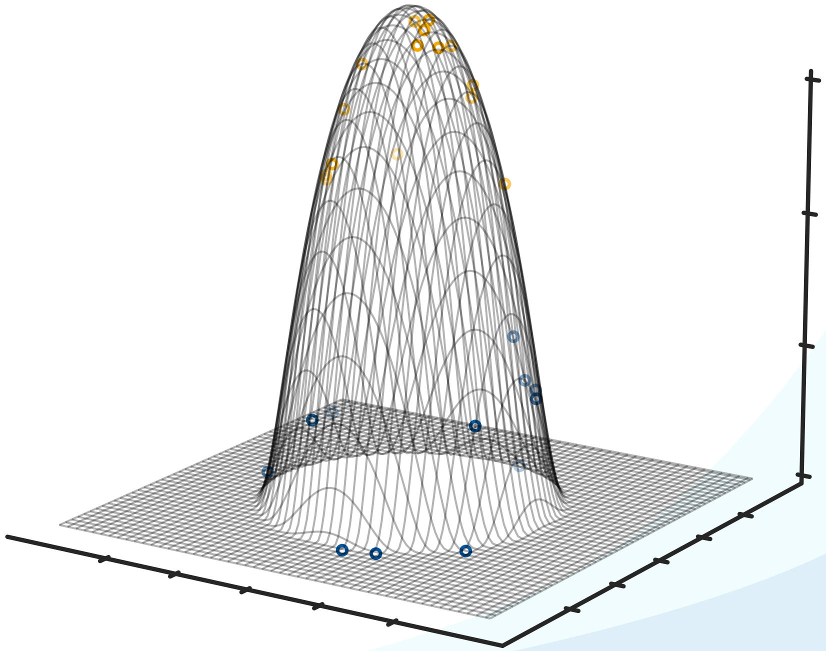
Non-Linear SVMs

- So far, we have only considered linear decision boundaries.
- We now combine non-linear basis functions with SVMs.



$$\phi : \mathbb{R}^D \to \mathbb{R}^M$$

$$M \gg D$$



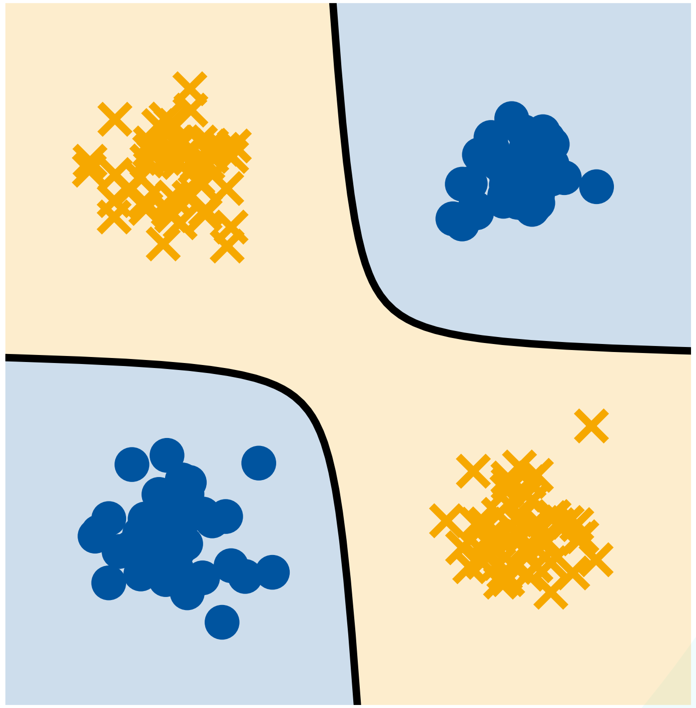
Feature Spaces

- We have already seen non-linear basis functions:
  - Apply a nonlinear transformation ϕ to the data points $\mathbf{x}_n$:
    $$\mathbf{x} \in \mathbb{R}^D, \quad \boldsymbol{\phi} : \mathbb{R}^D \rightarrow \mathbb{R}^M$$
  - Classify with a hyperplane in higher-dim. space $\mathbb{R}^M$:
    $$\mathbf{w}^\mathsf{T}\phi(\mathbf{x}) + b = 0$$

  $\Rightarrow$ Linear classifier in $\mathbb{R}^M$, nonlinear classifier in $\mathbb{R}^D$.

- Let us now apply this to SVMs…
  - We can train our SVM on the transformed features $\phi(\mathbf{x})$ to get non-linear decision boundaries.
  - Usually, $M \gg D$: evaluating $\mathbf{w}^\mathsf{T}\phi(\mathbf{x})$ can be quite expensive!

$\mathbf{x} = (x_1, x_2)^\mathsf{T}$



$\phi(\mathbf{x}) = (x_1, x_2, x_1x_2, x_1^2, x_2^2)^\mathsf{T}$

The Kernel Trick

- On a closer look, $\phi(\mathbf{x})$ only appears in the form of dot products:

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (\boldsymbol{\phi}(\mathbf{x}_m)^\mathsf{T} \boldsymbol{\phi}(\mathbf{x}_n))$$

$$\begin{aligned} y(\mathbf{x}) &= \mathbf{w}^\mathsf{T} \boldsymbol{\phi}(\mathbf{x}) + b \\ &= \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n)^\mathsf{T} \boldsymbol{\phi}(\mathbf{x}) + b \end{aligned}$$

$$\boxed{\mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n)}$$

The Kernel Trick

- On a closer look, $\phi(\mathbf{x})$ only appears in the form of dot products:

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\phi(\mathbf{x}_m)^\mathsf{T}\phi(\mathbf{x}_n))$$

$$y(\mathbf{x}) = \mathbf{w}^\mathsf{T}\phi(\mathbf{x}) + b$$

$$= \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)^\mathsf{T}\phi(\mathbf{x}) + b$$

$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^\mathsf{T}\phi(\mathbf{y})$$

Define a kernel function $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^\mathsf{T}\phi(\mathbf{y})$

$\Rightarrow$ *Use the kernel instead of the dot product.*

The Kernel Trick

- On a closer look, $\phi(\mathbf{x})$ only appears in the form of dot products:

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_m, \mathbf{x}_n)$$

$$\begin{aligned} y(\mathbf{x}) &= \mathbf{w}^{\mathsf{T}}\phi(\mathbf{x}) + b \\ &= \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n, \mathbf{x}) + b \end{aligned}$$

$$\boxed{k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^{\mathsf{T}}\phi(\mathbf{y})}$$

Define a kernel function $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^{\mathsf{T}}\phi(\mathbf{y})$

$\Rightarrow$ *Use the kernel instead of the dot product.*

- $k(\cdot, \cdot)$ implicitly maps the data to some higher-dimensional space, without having to compute $\phi(\mathbf{x})$.

When Can We Apply the Kernel Trick?

- In order for this to work, $k(\cdot,\cdot)$ needs to define an implicit mapping.
- Formally
  - A function $k(\mathbf{x}_1,\mathbf{x}_2) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is a kernel function, iff
    - There is a mapping $\phi(\mathbf{x}) : \mathbb{R}^D \to \mathcal{H}$ such that

$$k(\mathbf{x}_1,\mathbf{x}_2) = \phi(\mathbf{x}_1)^{\mathsf{T}}\phi(\mathbf{x}_2) \quad \forall \mathbf{x}_1,\mathbf{x}_2$$

- *When will this be the case?*

- When is a function $k(\mathbf{x}_1, \mathbf{x}_2)$ a valid kernel function? Two ways to check:

  1. Every Gram matrix K of k is symmetric positive definite:

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

A matrix M is positive definite if all eigenvalues of K are positive.

  - This is easy to verify for a given training set $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$.
  - Unfortunately, it has to hold for *every possible* such set.

$\Rightarrow$ *Very hard to prove in practice.*

- When is a function $k(\mathbf{x}_1, \mathbf{x}_2)$ a valid kernel function? Two ways to check:

  2. We can construct valid kernels from other valid kernels:
     - Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following combinations will also be valid

$$k(\mathbf{x}, \mathbf{x}') = c \cdot k_1(\mathbf{x}, \mathbf{x}')$$
$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}')$$
$$k(\mathbf{x}, \mathbf{x}') = polynomial(k_1(\mathbf{x}, \mathbf{x}')) \quad (with\ nonnegative\ coefficients)$$
$$k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}'))$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}')$$
$$k(\mathbf{x}, \mathbf{x}') = k_3(\boldsymbol{\phi}(x), \boldsymbol{\phi}(x'))$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathsf{T}} \mathbf{A} \mathbf{x}'$$

$\Rightarrow$ *Much easier to apply in practice.*

New SVM Formulation

- Maximize

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_m, \mathbf{x}_n)$$

under the constraints $\quad 0 \leq a_n \leq C \quad \forall n$

$$\sum_{n=1}^{N} a_n t_n = 0$$

- Classify new data points using

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n, \mathbf{x}) + b$$

Example: Polynomial Kernel

- We slightly adjust the polynomial basis function that we know:

$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)^\top$$

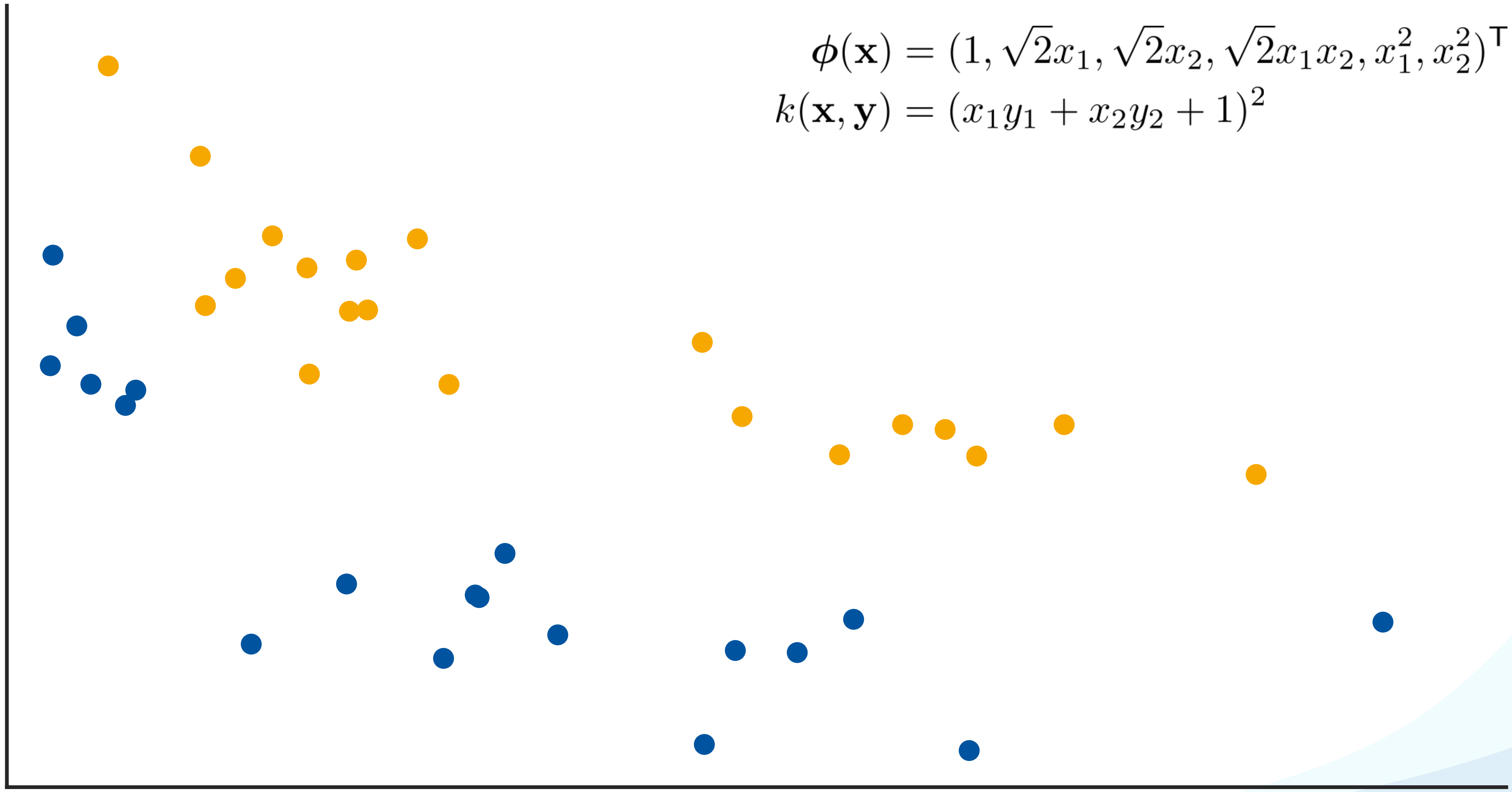
$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + 1)^2 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$$

- In fact, $(\mathbf{x}^\top \mathbf{y} + 1)^p$ is the kernel function for a polynomial of degree p.

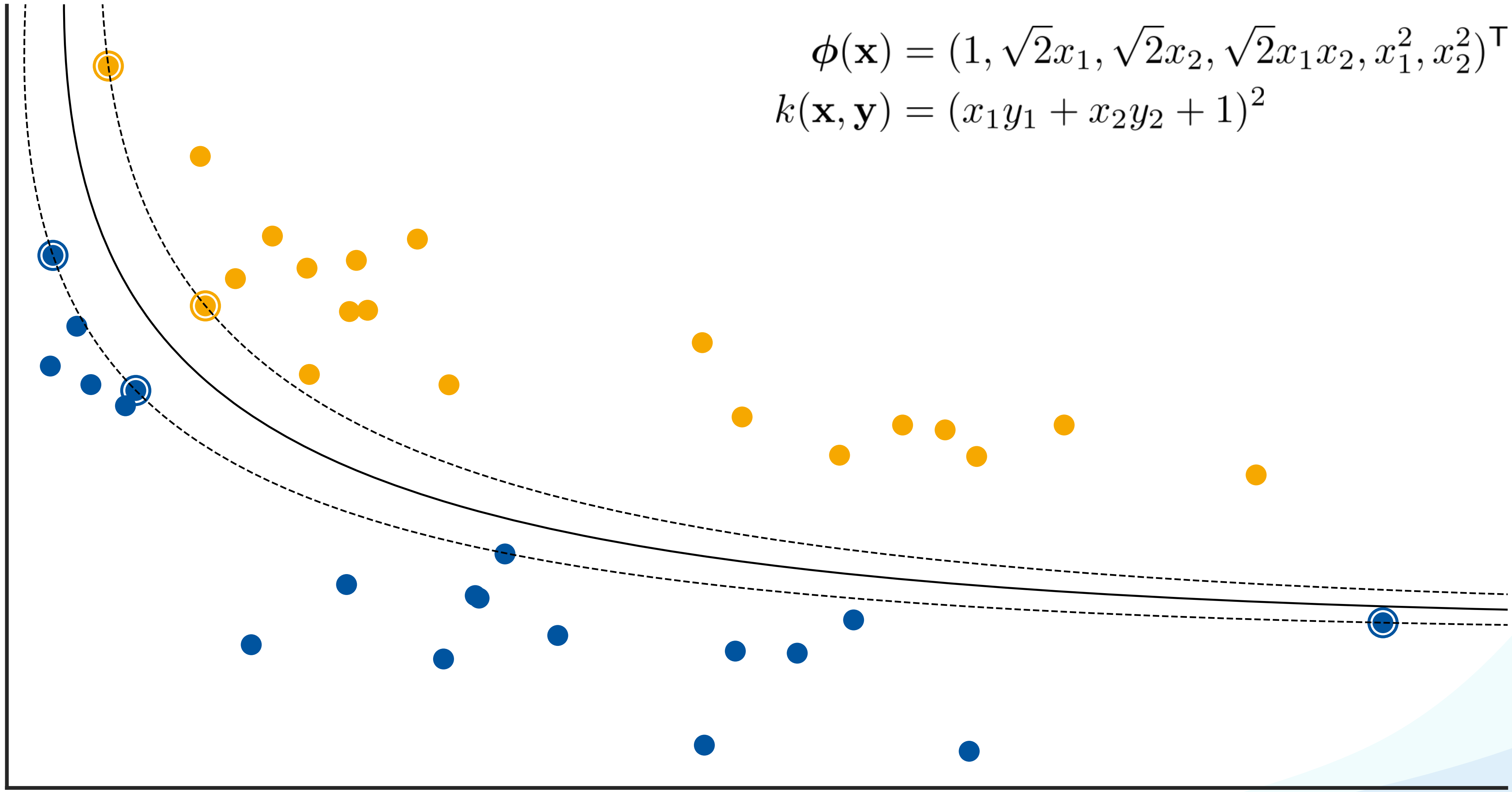
Example

$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)^\mathsf{T}$$

$$k(\mathbf{x}, \mathbf{y}) = (x_1y_1 + x_2y_2 + 1)^2$$



Example



$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)^\mathsf{T}$$

$$k(\mathbf{x}, \mathbf{y}) = (x_1y_1 + x_2y_2 + 1)^2$$

Advantages

- We can use high-dimensional or even infinite dimensional feature spaces
  - Since $\phi(\mathbf{x})$ is never computed explicitly.
- We can work with non-vector space data
  - We can define kernel functions for arbitrary data types!
  - Graphs, Sets, Sequences, Histograms, …
- Simple to use and work very well in most cases

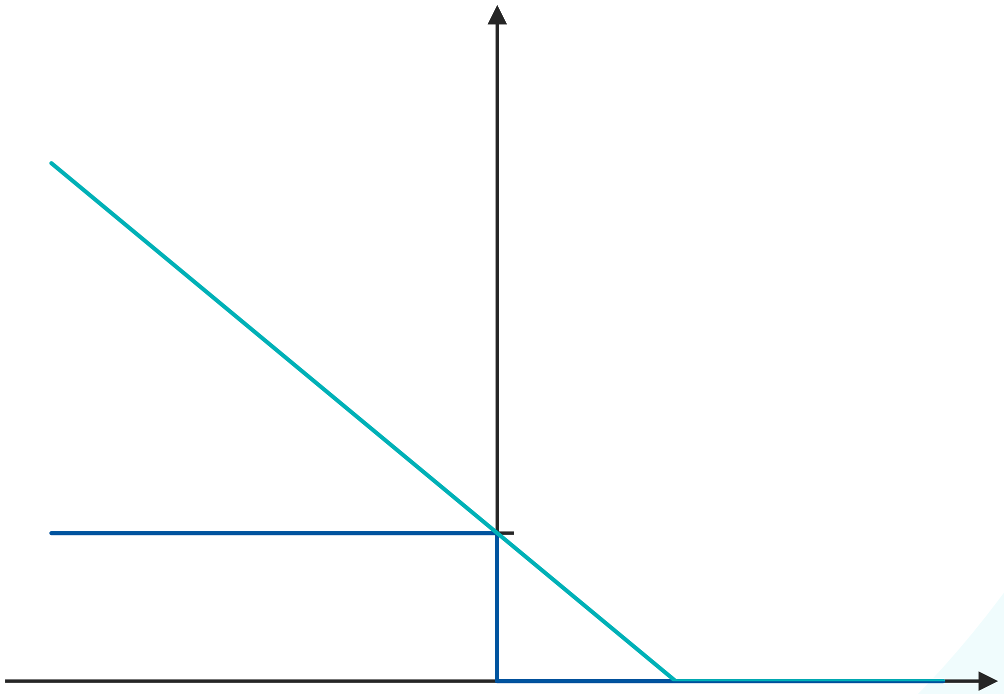
Limitations

- Which kernel to choose?
  - Model selection problem
- How to choose kernel parameters?
  - Hyperparameter optimization problem, usually solved by performing a grid search over the validation set
- Evaluation speed scales linearly with number of support vectors

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n, \mathbf{x}) + b$$

Support Vector Machines

1. Maximum Margin Classification
2. Primal Formulation
3. Dual Formulation
4. Soft-Margin SVMs
5. Non-linear SVMs
6. **Error Function Analysis**



Error Function Analysis

- We know how to formulate and optimize an SVM as a convex optimization problem:

$$\underset{\mathbf{w},\, b,\, \xi_n \in \mathbb{R}^+}{\arg\min} \; \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n$$

subject to the constraints

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n$$

Error Function Analysis

- We know how to formulate and optimize an SVM as a convex optimization problem:

$$\underset{\mathbf{w},\, b,\, \xi_n \in \mathbb{R}^+}{\arg\min} \; \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N} \xi_n$$

subject to the constraints

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n$$

But what error function does this correspond to?

- Integrate the constraints into the objective function:
  - Rewrite as $\xi_n \geq 1 - t_n y(\mathbf{x}_n)$
  - Thus, we obtain

$$\min_{\mathbf{w},\, b} E(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}[1 - t_n y(\mathbf{x}_n)]_+$$

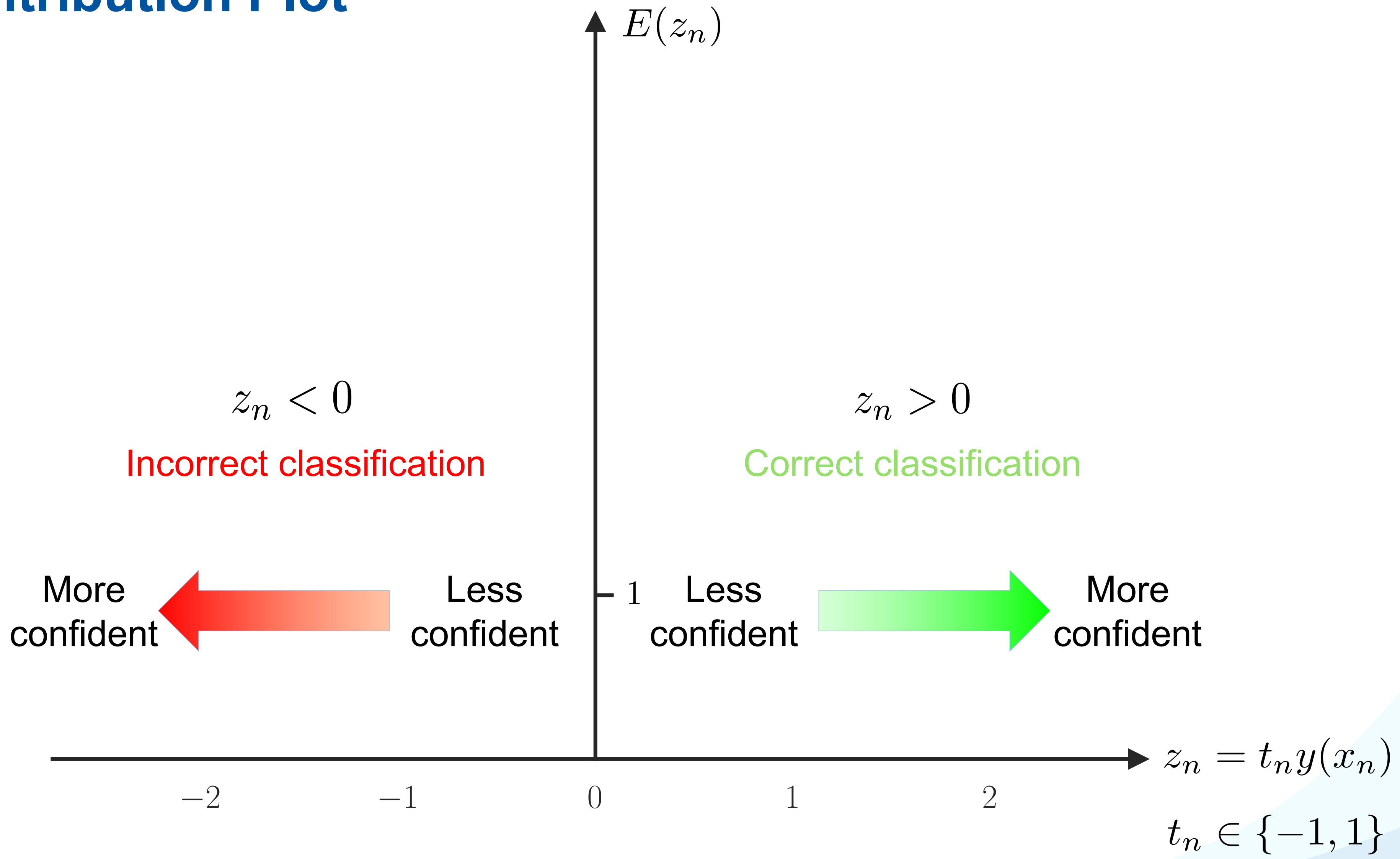
$$[x]_+ \equiv \max\{x, 0\}$$

The Hinge Loss

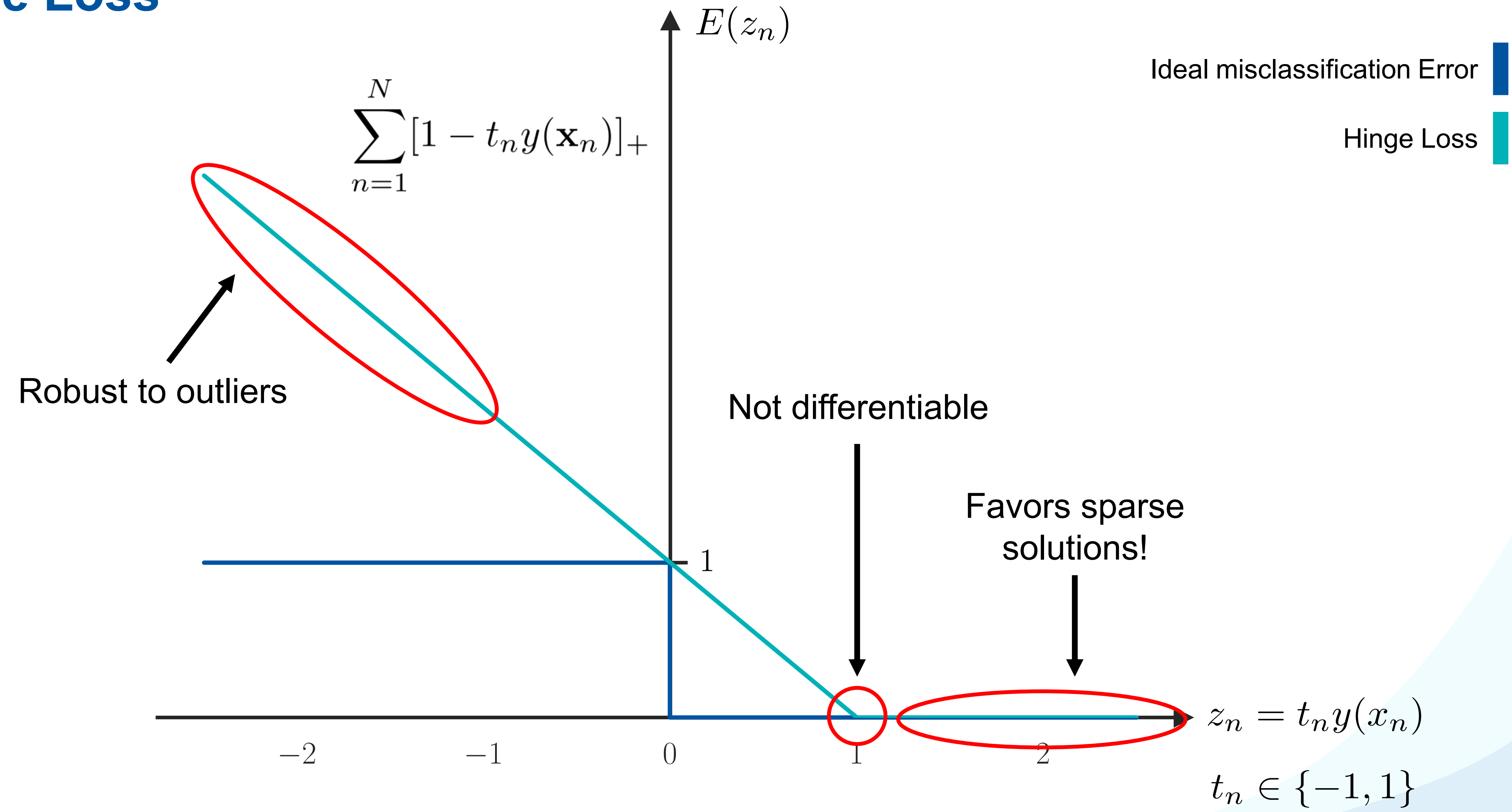
$$E(\mathbf{w}) = \underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{\text{L}_2 \text{ regularization}} + C \underbrace{\sum_{n=1}^{N} [1 - t_n y(\mathbf{x}_n)]_+}_{\text{Hinge loss}}$$

- Regularization bounds parameter size.
- Hinge Loss enforces sparsity:
  - Only a subset of training data points actually influences the decision boundary.
  - Still, all input dimensions are used.
- This formulation corresponds to an unconstrained optimization of a non-differentiable function.
  - Very efficient: stochastic (sub-)gradient descent.

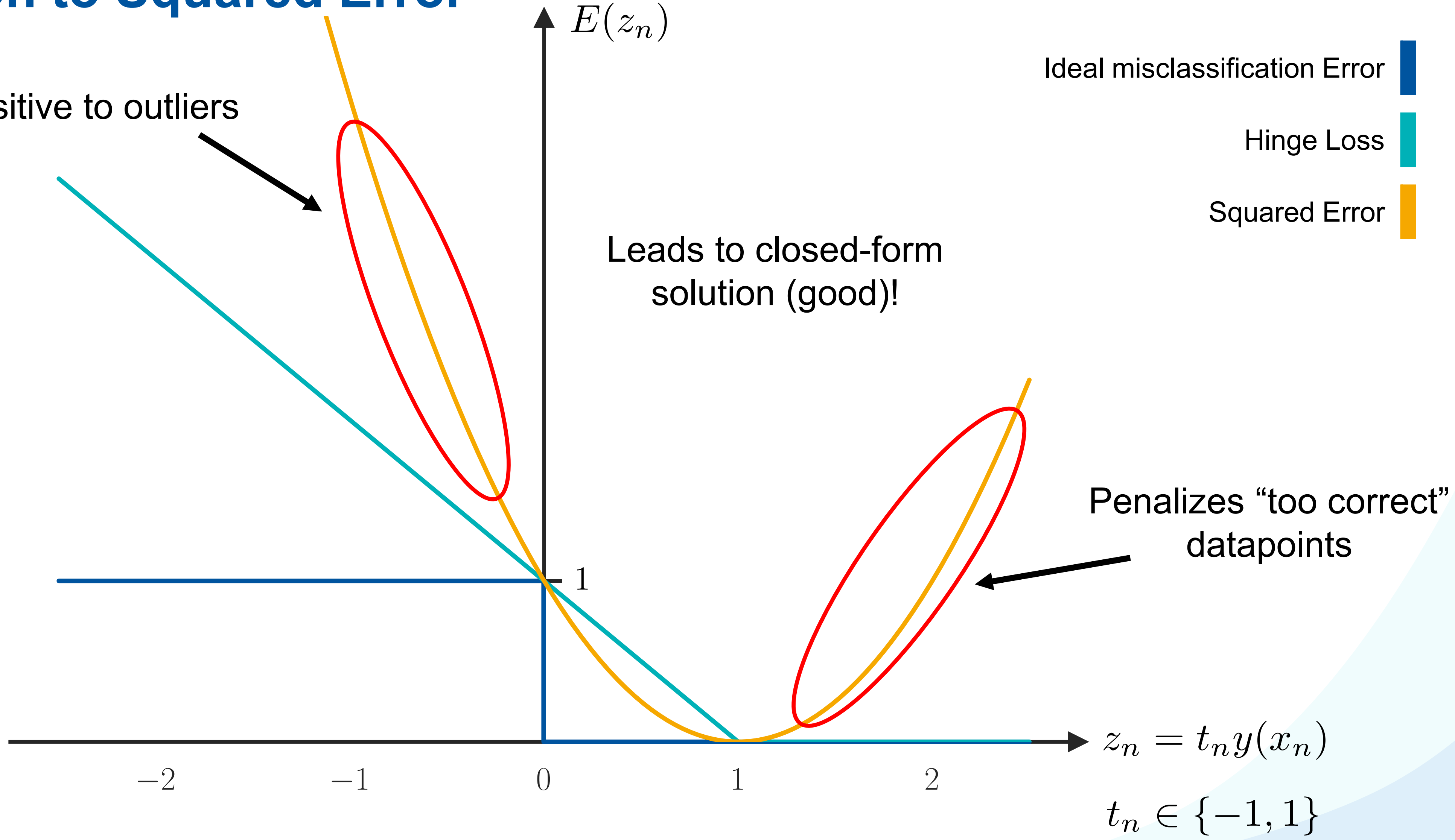
Error Contribution Plot



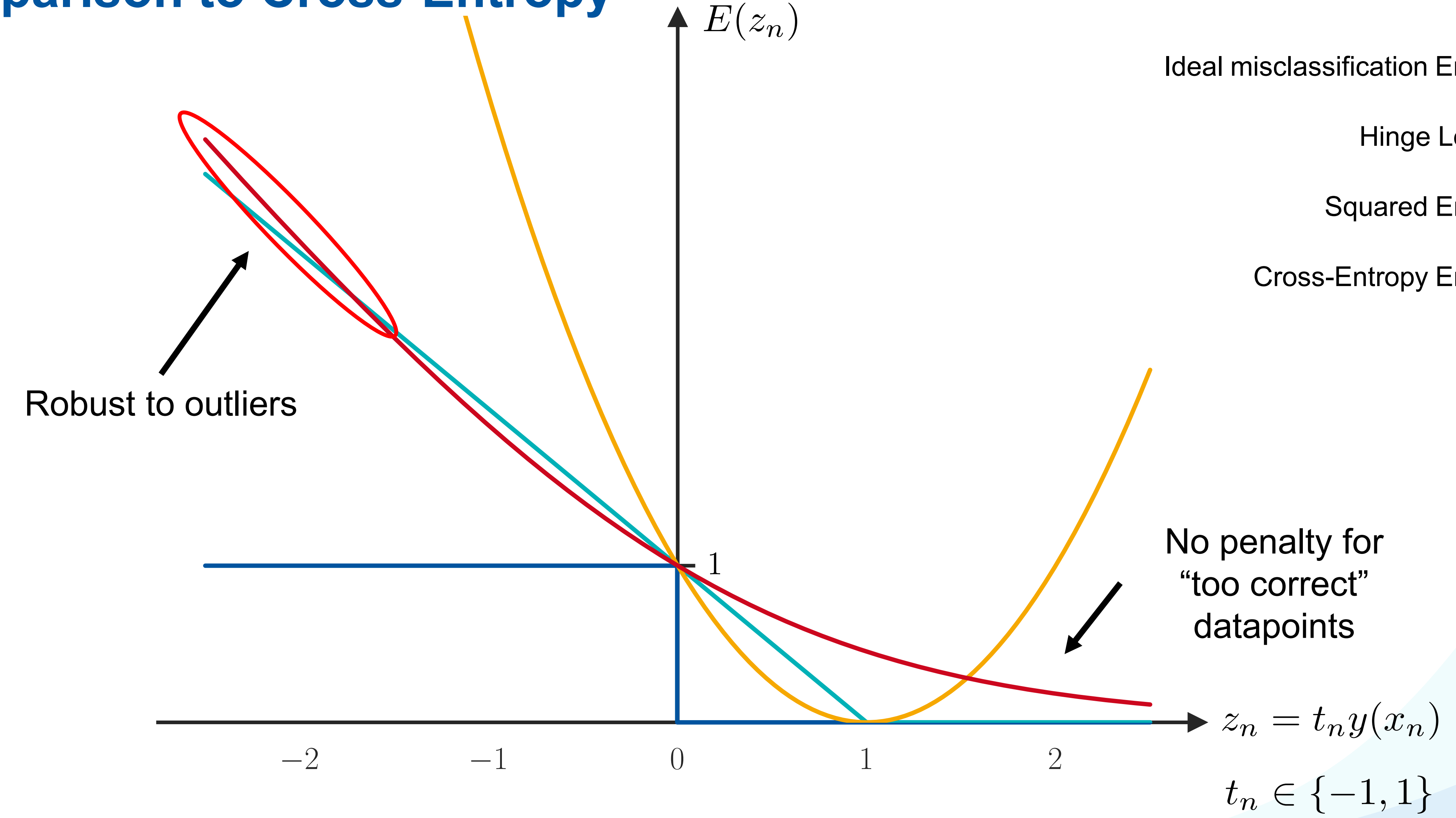
Hinge Loss


$E(z_n)$
Ideal misclassification Error
Hinge Loss
$\sum_{n=1}^{N}[1 - t_n y(\mathbf{x}_n)]_+$
Robust to outliers
Not differentiable
Favors sparse solutions!
1
-2
-1
0
1
2
$z_n = t_n y(x_n)$
$t_n \in \{-1, 1\}$

Comparison to Squared Error



Comparison to Cross-Entropy



Discussion: Hinge Loss

Advantages

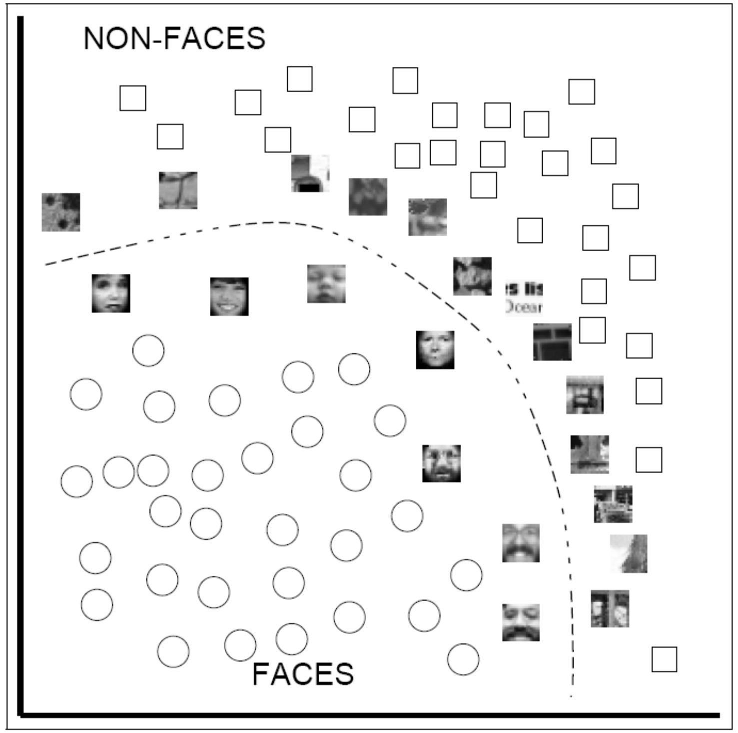
- Favors sparse solutions that only depend on a subset of training data points.
- Robust to outliers (only a linear penalty for misclassified points).
- Convex function, unique minimum exists.

Limitations

- Not differentiable (cannot minimize this loss using standard gradient descent, but need to use subgradient descent).

Support Vector Machines

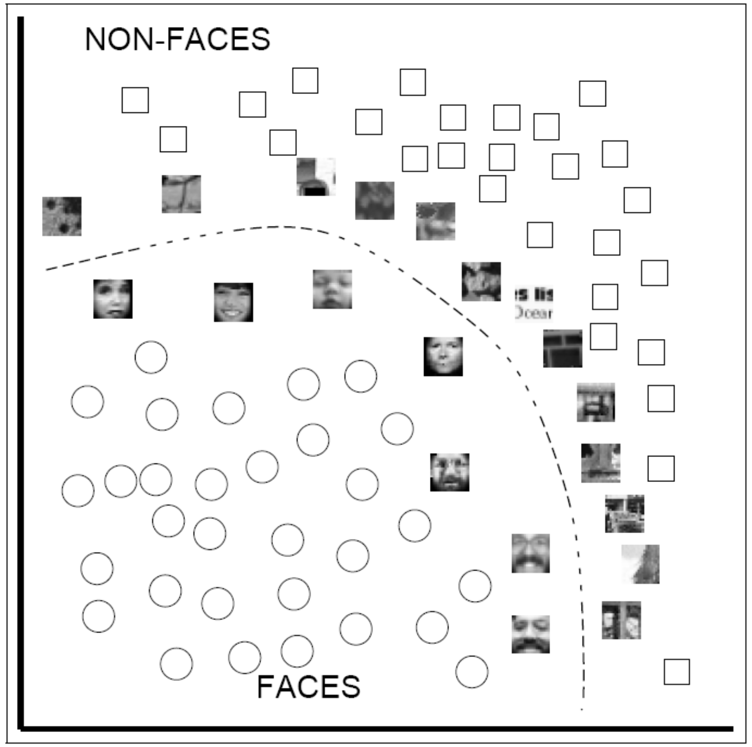
1. Maximum Margin Classification
2. Primal Formulation
3. Dual Formulation
4. Soft-Margin SVMs
5. Non-linear SVMs
6. Error Function Analysis
7. **Applications**



Interpretation of Support Vectors

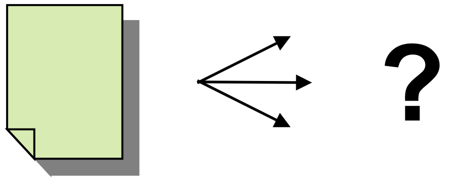
- Let's consider an image classification problem:
  - E.g., *faces* vs. *non-faces*

- The support vectors chosen by the SVM correspond to the hardest examples
  - The real faces closest to non-faces
  - The non-faces closest to real faces
- *We can visualize those support vectors again as images to get some insights into what the SVM reacts to…*

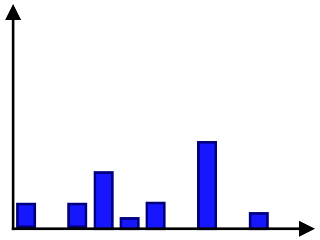


Example Application: Text Classification

- Problem:
  - Classify a document in a number of categories



- Representation:
  - “Bag-of-words” approach
  - Histogram of word counts (on learned dictionary)
    - Very high-dimensional feature space (~10.000 dimensions)
    - Few irrelevant features



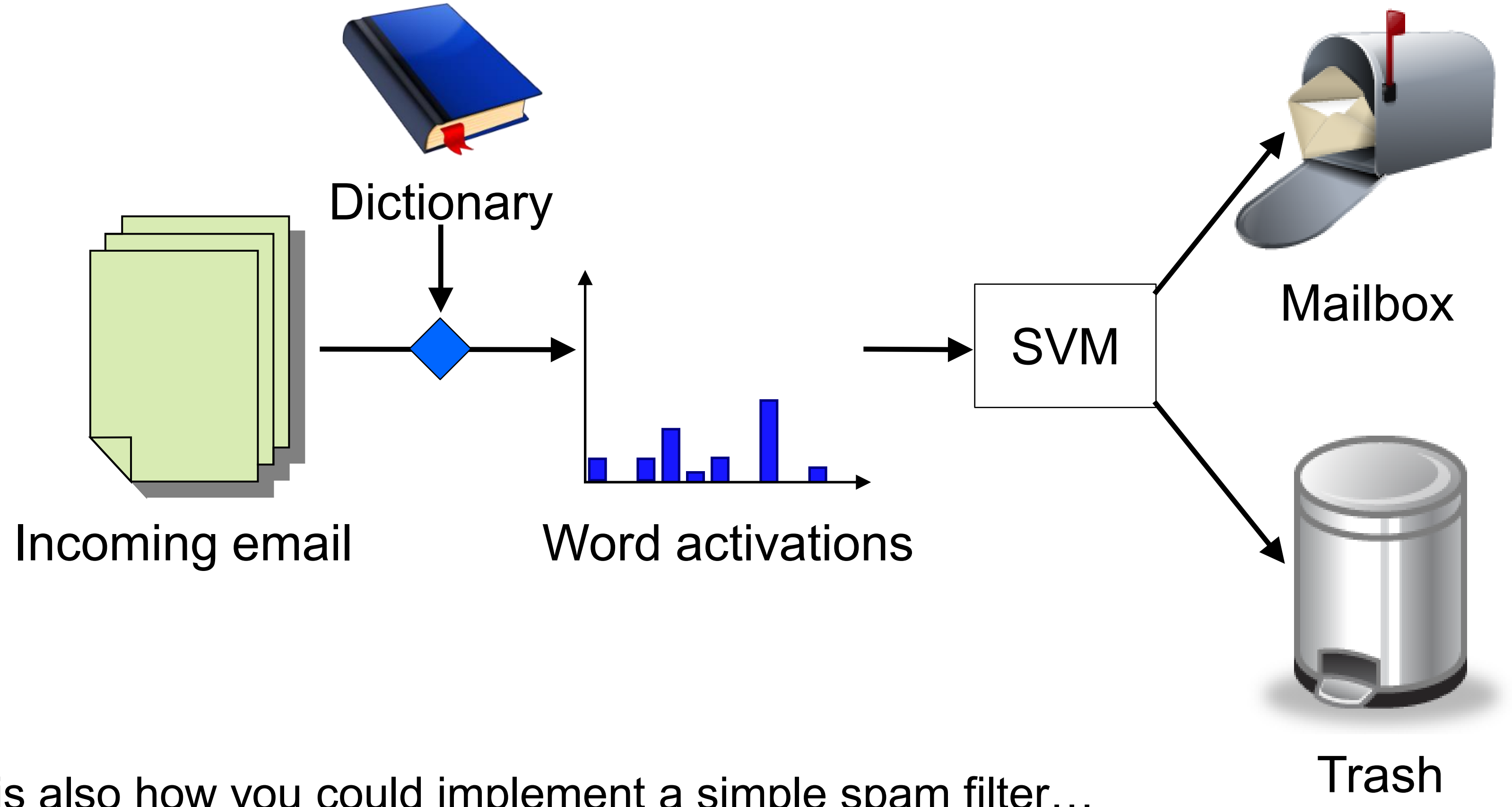
- This was one of the first applications of SVMs
  - T. Joachims (1997)

Example Application: Text Classification

					SVM (poly) degree $d =$					SVM (rbf) width $\gamma =$			
	Bayes	Rocchio	C4.5	k-NN	1	2	3	4	5	0.6	0.8	1.0	1.2
earn	95.9	96.1	96.1	97.3	98.2	98.4	**98.5**	98.4	98.3	**98.5**	98.5	98.4	98.3
acq	91.5	92.1	85.3	92.0	92.6	94.6	**95.2**	95.2	95.3	95.0	95.3	95.3	**95.4**
money-fx	62.9	67.6	69.4	78.2	66.9	72.5	75.4	74.9	**76.2**	74.0	75.4	**76.3**	75.9
grain	72.5	79.5	89.1	82.2	91.3	93.1	**92.4**	91.3	89.9	**93.1**	91.9	91.9	90.6
crude	81.0	81.5	75.5	85.7	86.0	87.3	88.6	**88.9**	87.8	**88.9**	89.0	88.9	88.2
trade	50.0	77.4	59.2	77.4	69.2	75.5	76.6	77.3	**77.1**	76.9	78.0	**77.8**	76.8
interest	58.0	72.5	49.1	74.0	69.8	63.3	67.9	73.1	**76.2**	74.4	75.0	**76.2**	76.1
ship	78.7	83.1	80.9	79.2	82.0	85.4	86.0	**86.5**	86.0	**85.4**	86.5	87.6	87.1
wheat	60.6	79.4	85.5	76.6	83.1	84.5	85.2	**85.9**	83.8	**85.2**	85.9	85.9	85.9
corn	47.3	62.2	87.7	77.9	86.0	86.5	85.3	**85.7**	83.9	**85.1**	85.7	85.7	84.5
microavg.	**72.0**	**79.9**	**79.4**	**82.3**	84.2	85.1	85.9	86.2	85.9	86.4	86.5	86.3	86.2
					combined: **86.0**					combined: **86.4**			

- Results by T. Joachims (1997)

Example Application: Text Classification

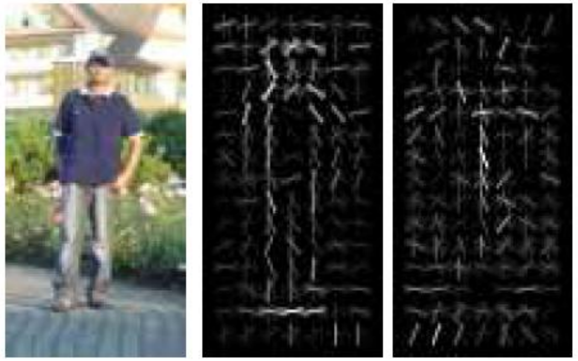


- This is also how you could implement a simple spam filter…

Example Application: Visual Object Detection

Sliding-Window Approach

- Densely scan an image with a sliding window
- Object detection by per-window classification
  - E.g., use a histogram of gradients (HOG) feature representation
  - Train a linear SVM for *object* vs. *non-object* classification.

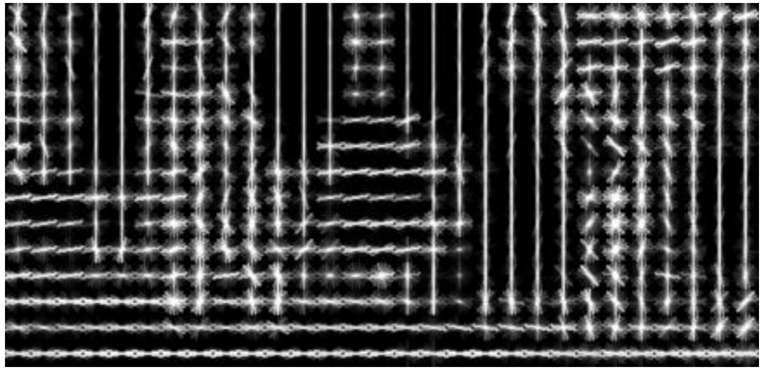


Example Application: Pedestrian Detection with SVMs

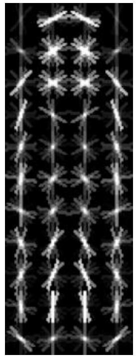
- Train a pedestrian template using a linear SVM
- At test time, convolve feature map with learned template $\mathbf{w}$
  - Linear SVM classification function ⇔ convolution with "template" $\mathbf{w}$

$$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

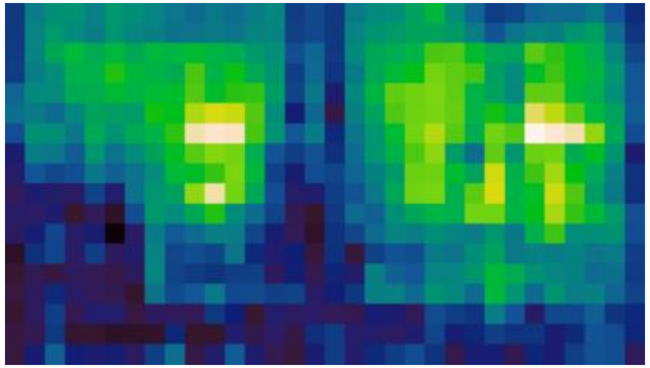
HOG feature map
$\mathbf{x}$



Template
$\mathbf{w}$



Detector response map
$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + \mathrm{b}$



N. Dalal, B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

Example Application: Pedestrian Detection with SVMs

N. Dalal, B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

References and Further Reading

- More information about SVMs is available in Chapter 7.1 of Bishop's book.

Christopher M. Bishop
Pattern Recognition and Machine Learning
Springer, 2006