Machine Learning – Lecture 10

AdaBoost

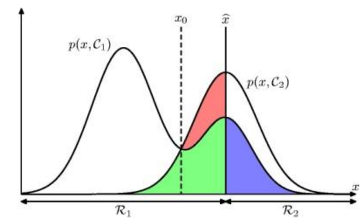
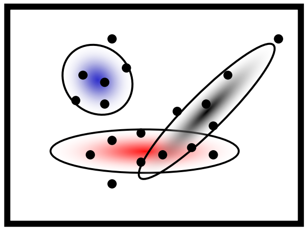
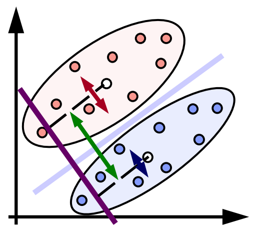
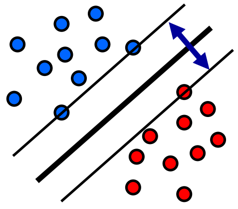
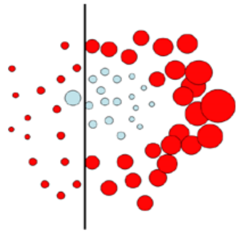
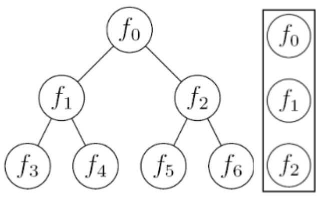
13.11.2019

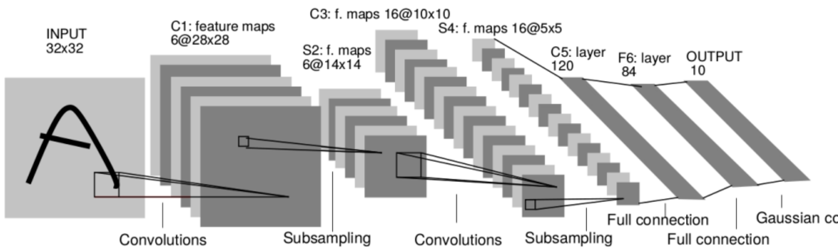
Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

leibe@vision.rwth-aachen.de

Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
  - Randomized Trees, Forests & Ferns
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks



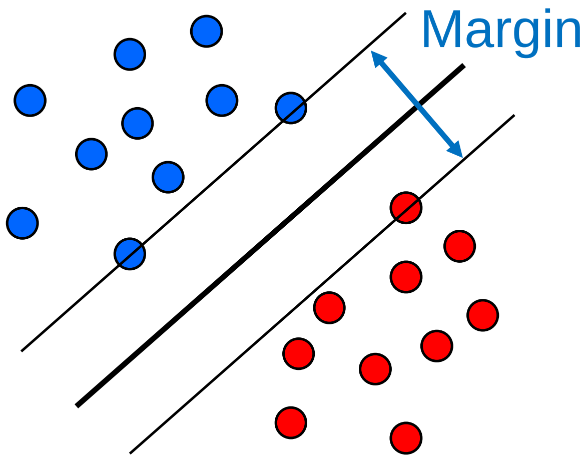
Topics of This Lecture

- Recap: Nonlinear Support Vector Machines
- Ensembles of classifiers
  - Bagging
  - Bayesian Model Averaging
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions

Recap: Support Vector Machine (SVM)

- Basic idea
  - The SVM tries to find a classifier which maximizes the margin between pos. and neg. data points.
  - Up to now: consider linear classifiers

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0$$



- Formulation as a convex optimization problem
  - Find the hyperplane satisfying

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

under the constraints

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

based on training data points $\mathbf{x}_n$ and target values $t_n \in \{-1, 1\}$

Recap: SVM – Dual Formulation

- Maximize

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

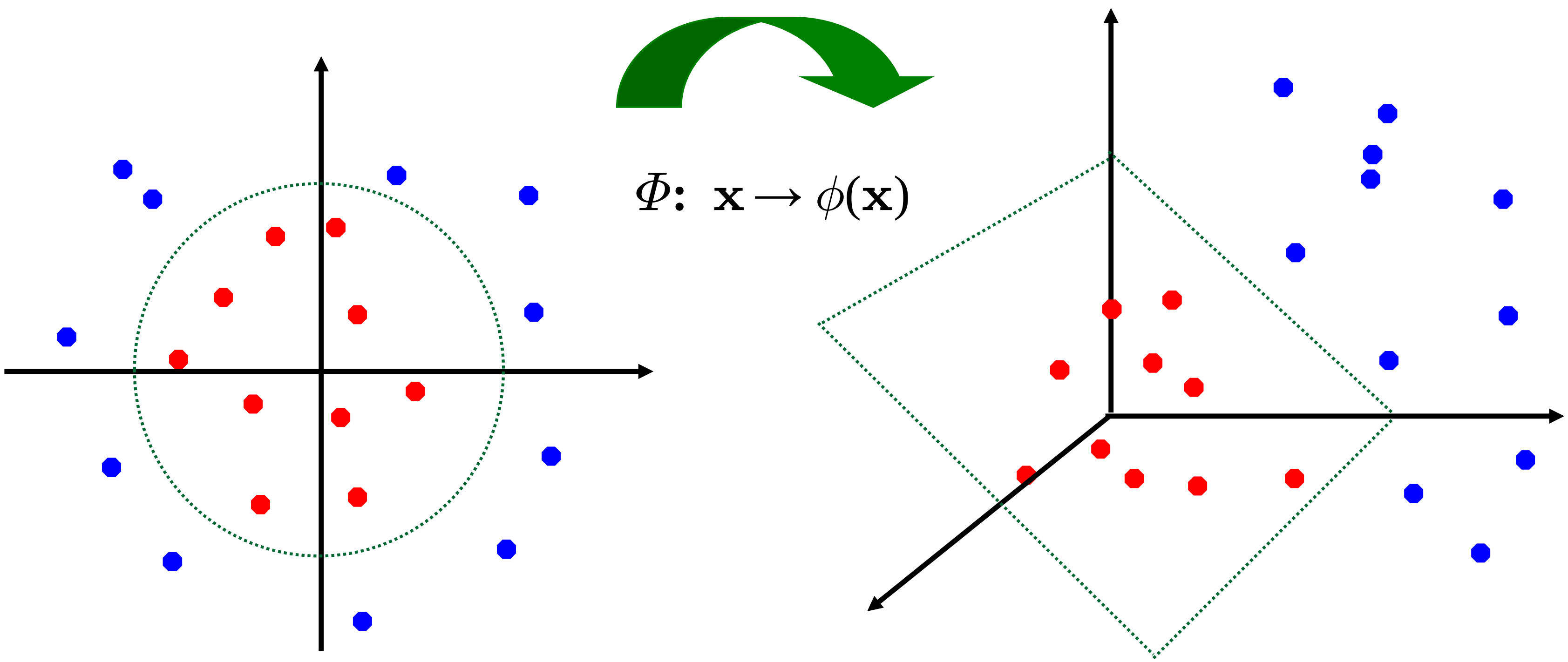
under the conditions

$$\begin{aligned} a_n &\geq 0 \quad \forall n \\ \sum_{n=1}^{N} a_n t_n &= 0 \end{aligned}$$

- Comparison
  - L_d is equivalent to the primal form L_p, but only depends on a_n.
  - L_p scales with $\mathcal{O}(D^3)$.
  - L_d scales with $\mathcal{O}(N^3)$ – in practice between $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$.

Recap: Nonlinear SVMs

- General idea: The original input space can be mapped to some higher-dimensional feature space where the training set is separable:



Slide credit: Raymond Mooney

Recap: The Kernel Trick

- Important observation
  - $\phi(\mathbf{x})$ only appears in the form of dot products $\phi(\mathbf{x})^{\mathsf{T}}\phi(\mathbf{y})$:

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

  - Define a so-called kernel function $k(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x})^{\mathsf{T}}\phi(\mathbf{y})$.
  - Now, in place of the dot product, use the kernel instead:

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

  - The kernel function *implicitly* maps the data to the higher-dimensional space (without having to compute $\phi(\mathbf{x})$ explicitly)!

Recap: Nonlinear SVM – Dual Formulation

- SVM Dual: 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 conditions

$$0 \cdot \quad a_n \cdot \quad C$$

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

- Classify new data points using

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

Recap: SVM Loss Function

- Traditional soft-margin formulation

$$\min_{\mathbf{w}\in\mathbb{R}^D,\, \xi_n\in\mathbb{R}^+} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n$$

"Maximize the margin"

subject to the constraints

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

"Most points should be on the correct side of the margin"

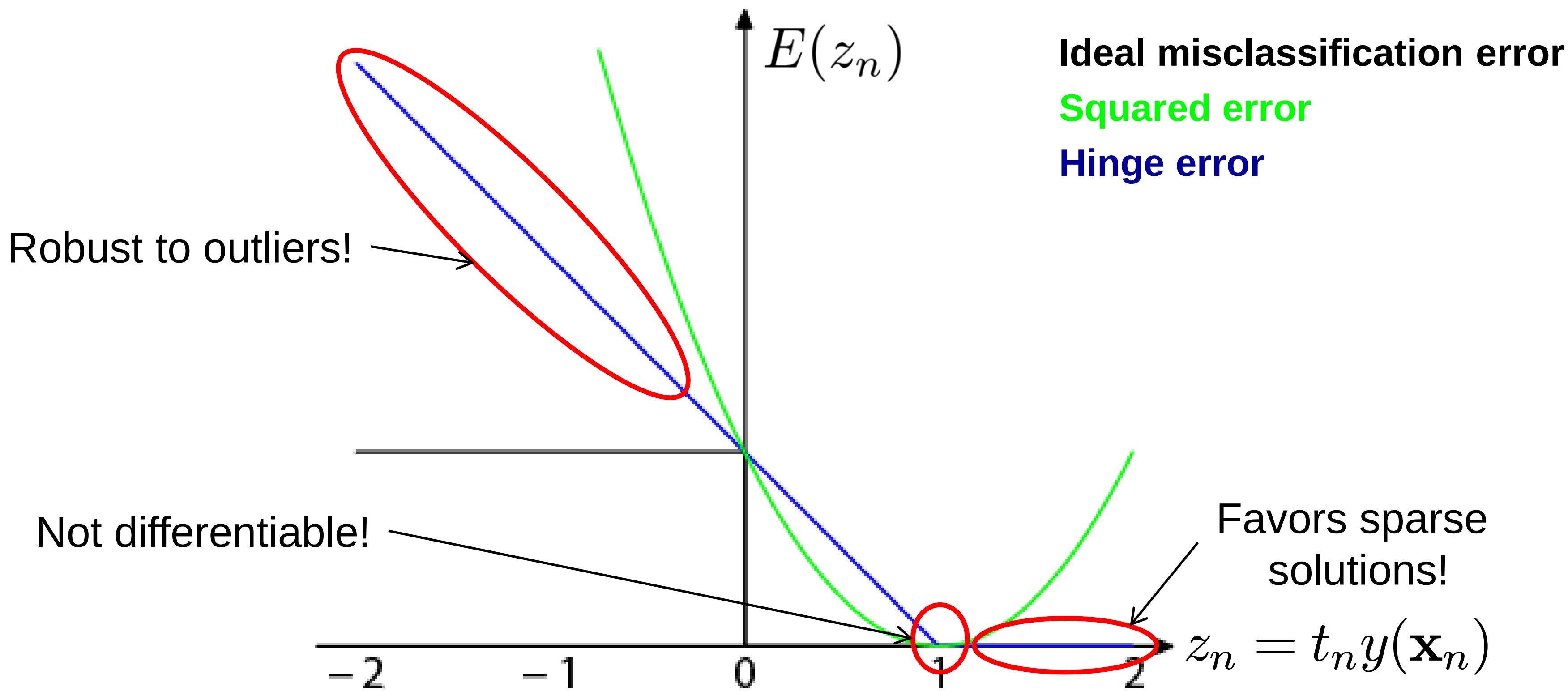
- Different way of looking at it
  - We can reformulate the constraints into the objective function.

$$\min_{\mathbf{w}\in\mathbb{R}^D} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{L_2 \text{ regularizer}} + \underbrace{C\sum_{n=1}^{N}[1 - t_n y(\mathbf{x}_n)]_+}_{\text{“Hinge loss”}}$$

where $[x]_+ := \max\{0, x\}$.

RWTH AACHEN UNIVERSITY

Recap: Hinge Loss Analysis



- "Hinge error" used in SVMs
  - Zero error for points outside the margin ($z_n > 1$) $\Rightarrow$ sparsity
  - Linear penalty for misclassified points ($z_n < 1$) $\Rightarrow$ robustness
  - Not differentiable around $z_n = 1$ $\Rightarrow$ Cannot be optimized directly.

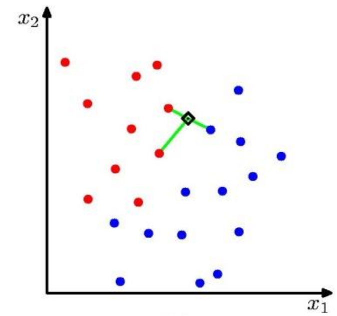
Image source: Bishop, 2006

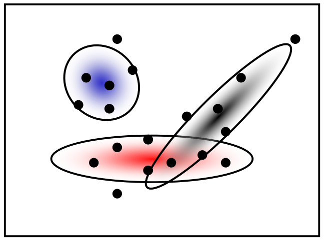
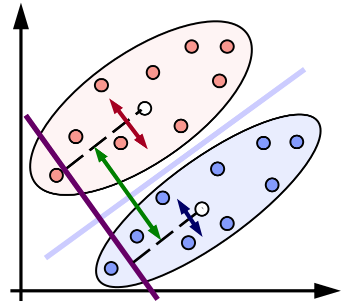
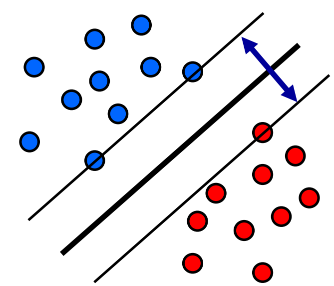
Topics of This Lecture

- Recap: Nonlinear Support Vector Machines
- Ensembles of classifiers
  - Bagging
  - Bayesian Model Averaging
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions

So Far…

- We’ve seen already a variety of different classifiers
  - k-NN
  - Bayes classifiers
  - Linear discriminants
  - SVMs



- Each of them has their strengths and weaknesses…
  - Can we improve performance by combining them?

Ensembles of Classifiers

- Intuition
  - Assume we have K classifiers.
  - They are independent (i.e., their errors are uncorrelated).
  - Each of them has an error probability $p < 0.5$ on training data.
    - Why can we assume that p won't be larger than 0.5?
  - Then a simple majority vote of all classifiers should have a lower error than each individual classifier…

Constructing Ensembles

- How do we get different classifiers?
  - Simplest case: train same classifier on different data.
  - But… where shall we get this additional data from?
    - Recall: training data is very expensive!
- Idea: Subsample the training data
  - Reuse the same training algorithm several times on different subsets of the training data.
- Well-suited for "unstable" learning algorithms
  - Unstable: small differences in training data can produce very different classifiers
    - E.g., Decision trees, neural networks, rule learning algorithms,…
  - Stable learning algorithms
    - E.g., Nearest neighbor, linear regression, SVMs,…

Constructing Ensembles

- Bagging = "Bootstrap aggregation" (Breiman 1996)
  - In each run of the training algorithm, randomly select M samples with replacement from the full set of N training data points.
  - If $M = N$, then on average, 63.2% of the training points will be represented. The rest are duplicates.

- Injecting randomness
  - Many (iterative) learning algorithms need a random initialization (e.g. k-means, EM)
  - Perform mutliple runs of the learning algorithm with different random initializations.

Bayesian Model Averaging

- Model Averaging
  - Suppose we have H different models $h = 1,\ldots,H$ with prior probabilities $p(h)$.
  - Construct the marginal distribution over the data set

$$p(\mathbf{X}) = \sum_{h=1}^{H} p(\mathbf{X}|h)p(h)$$

- Interpretation
  - Just one model is responsible for generating the entire data set.
  - The probability distribution over h just reflects our uncertainty which model that is.
  - As the size of the data set increases, this uncertainty reduces, and $p(\mathbf{X}|h)$ becomes focused on just one of the models.

Note the Different Interpretations!

- Model Combination (e.g., Mixtures of Gaussians)
  - Different data points *generated by different model components*.
  - Uncertainty is about which component created which data point.

  $\Rightarrow$ One latent variable $\mathbf{z}_n$ for each data point:

$$p(\mathbf{X}) = \prod_{n=1}^{N} p(\mathbf{x}_n) = \prod_{n=1}^{N} \sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n)$$

- Bayesian Model Averaging
  - The whole data set is *generated by a single model*.
  - Uncertainty is about which model was responsible.

  $\Rightarrow$ One latent variable $\mathbf{z}$ for the entire data set:

$$p(\mathbf{X}) = \sum_{\mathbf{z}} p(\mathbf{X}, \mathbf{z})$$

Model Averaging: Expected Error

- Combine M predictors $y_m(\mathbf{x})$ for target output $h(\mathbf{x})$.
  - E.g. each trained on a different bootstrap data set by bagging.
  - The committee prediction is given by

$$y_{COM}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x})$$

  - The output can be written as the true value plus some error.

$$y(\mathbf{x}) = h(\mathbf{x}) + \epsilon(\mathbf{x})$$

  - Thus, the expected sum-of-squares error takes the form

$$\mathbb{E}_{\mathbf{x}} = \left[\{y_m(\mathbf{x}) - h(\mathbf{x})\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

Model Averaging: Expected Error

- Average error of individual models

$$\mathbb{E}_{AV} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

$$\boxed{y_m(\mathbf{x}) = h(\mathbf{x}) + \epsilon_m(\mathbf{x})}$$

- Average error of committee

$$\mathbb{E}_{COM} = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}) - h(\mathbf{x})\right\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right]$$

- Assumptions
  - Errors have zero mean: $\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\right] = 0$
  - Errors are uncorrelated: $\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\epsilon_j(\mathbf{x})\right] = 0$

- Then:

$$\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{AV}$$

Isn't this spectacular?

Model Averaging: Expected Error

- Average error of committee

$$\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{AV}$$

  - This suggests that the average error of a model can be reduced by a factor of M simply by averaging M versions of the model!
  - Spectacular indeed…
  - This sounds almost too good to be true…

- And it is… Can you see where the problem is?
  - Unfortunately, this result depends on the assumption that the errors are all uncorrelated.
  - In practice, they will typically be highly correlated.
  - Still, it can be shown that

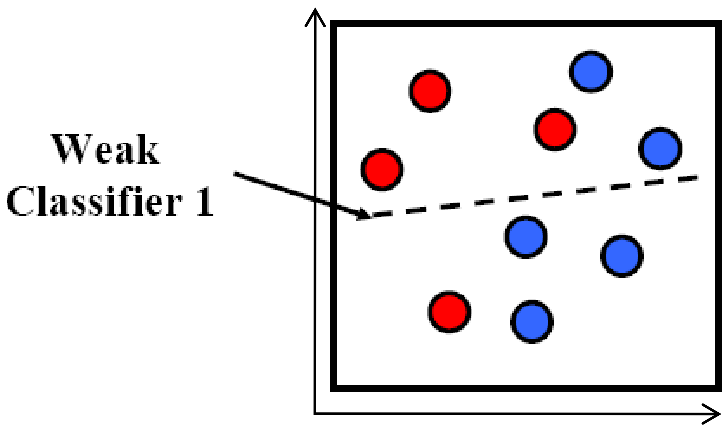
$$\mathbb{E}_{COM} \cdot \mathbb{E}_{AV}$$

AdaBoost – “Adaptive Boosting”

- Main idea [Freund & Schapire, 1996]
  - Iteratively select an ensemble of component classifiers
  - After each iteration, reweight misclassified training examples.
    - Increase the chance of being selected in a sampled training set.
    - Or increase the misclassification cost when training on the full set.
- Components
  - $h_m(\mathbf{x})$: “weak” or base classifier
    - Condition: <50% training error over any distribution
  - $H(\mathbf{x})$: “strong” or final classifier
- AdaBoost:
  - Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:

$$H(\mathbf{x}) = sign\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$$

AdaBoost: Intuition



Consider a 2D feature space with positive and negative examples.

Each weak classifier splits the training examples with at least 50% accuracy.

Examples misclassified by a previous weak learner are given more emphasis at future rounds.

Figure adapted from Freund & Schapire

AdaBoost: Intuition

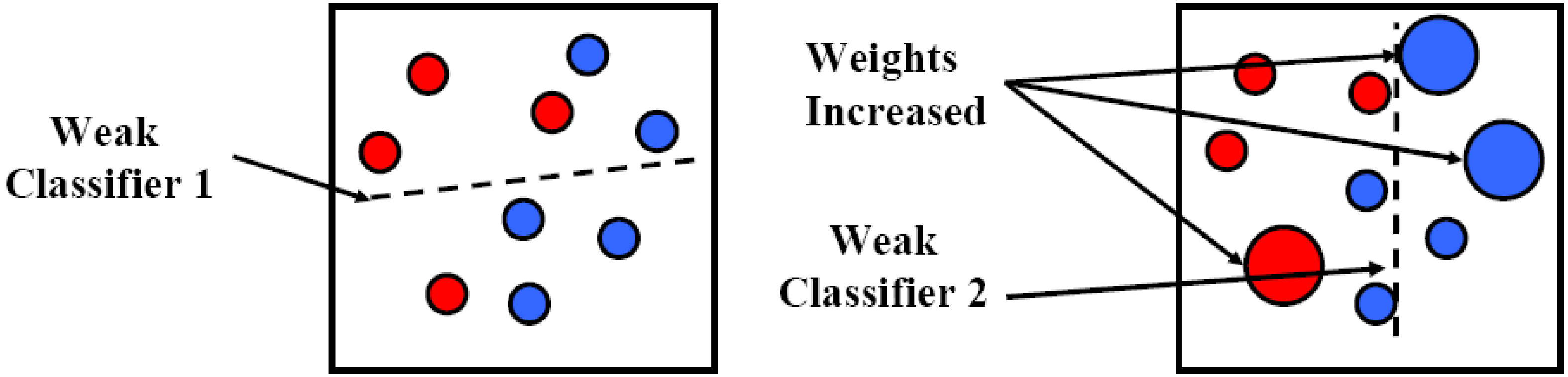
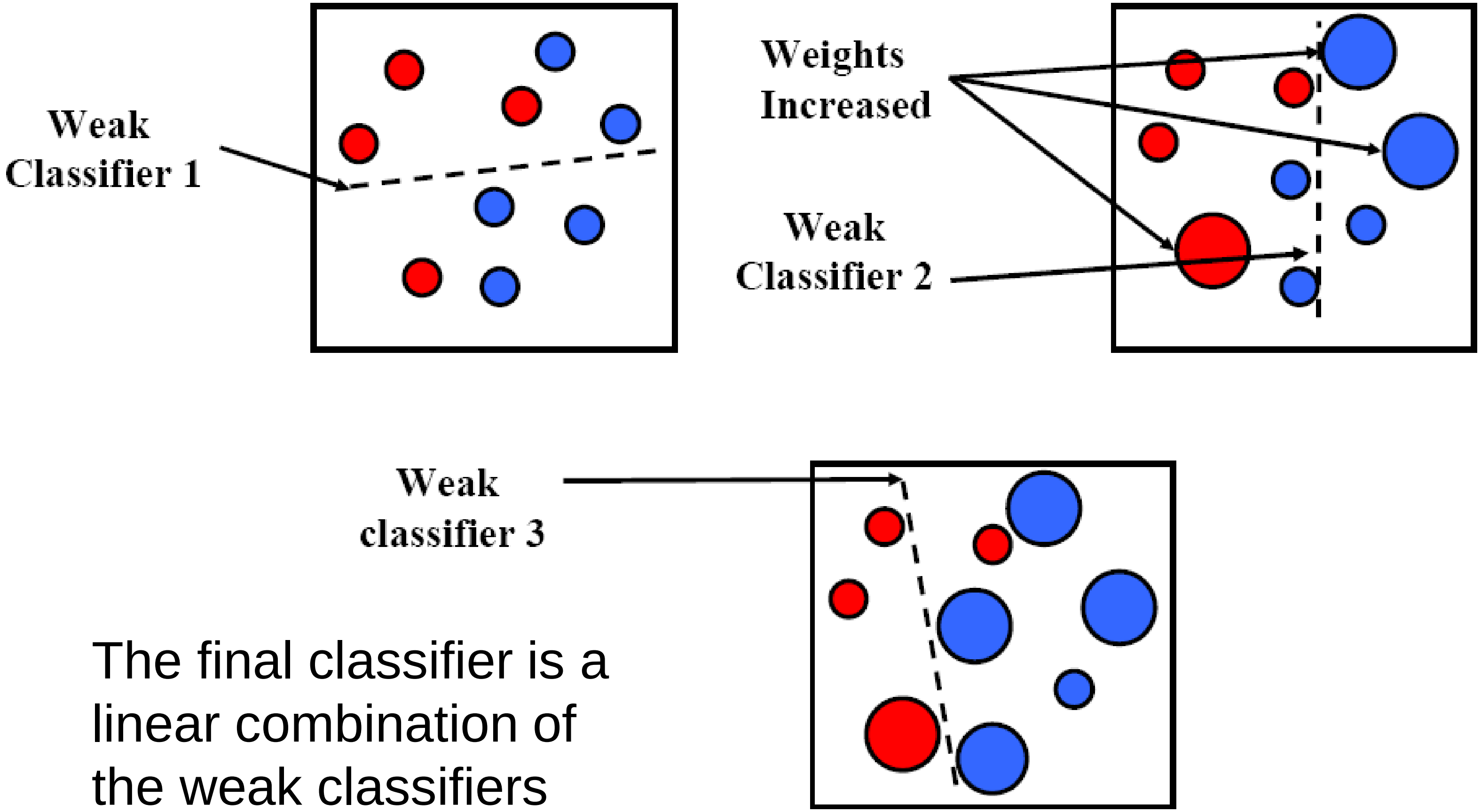


Figure adapted from Freund & Schapire

RWTH AACHEN UNIVERSITY

AdaBoost: Intuition



The final classifier is a linear combination of the weak classifiers

Figure adapted from Freund & Schapire

AdaBoost – Formalization

- 2-class classification problem
  - Given: training set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with target values $\mathbf{T} = \{t_1, \ldots, t_N\}$, $t_n \in \{-1,1\}$.
  - Associated weights $\mathbf{W}=\{w_1, \ldots, w_N\}$ for each training point.

- Basic steps
  - In each iteration, AdaBoost trains a new weak classifier $h_m(\mathbf{x})$ based on the current weighting coefficients $\mathbf{W}^{(m)}$.
  - We then adapt the weighting coefficients for each point
    - Increase w_n if $\mathbf{x}_n$ was misclassified by $h_m(\mathbf{x})$.
    - Decrease w_n if $\mathbf{x}_n$ was classified correctly by $h_m(\mathbf{x})$.
  - Make predictions using the final combined model

$$H(\mathbf{x}) = sign\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$$

AdaBoost – Algorithm

1. Initialization: Set $w_n^{(1)} = \frac{1}{N}$ for $n = 1,\ldots,N$.
2. For $m = 1,\ldots,M$ iterations
   a) Train a new weak classifier $h_m(\mathbf{x})$ using the current weighting coefficients $\mathbf{W}^{(m)}$ by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n)$$

$$I(A) = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{else} \end{cases}$$

   b) Estimate the weighted error of this classifier on $\mathbf{X}$:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

   c) Calculate a weighting coefficient for $h_m(\mathbf{x})$:

$$\alpha_m = \ ?$$

   d) Update the weighting coefficients:

$$w_n^{(m+1)} = \ ?$$

How should we do this exactly?

AdaBoost – Historical Development

- Originally motivated by Statistical Learning Theory
  - AdaBoost was introduced in 1996 by Freund & Schapire.
  - It was empirically observed that AdaBoost often tends not to overfit. (Breiman 96, Cortes & Drucker 97, etc.)
  - As a result, the margin theory (Schapire et al. 98) developed, which is based on loose generalization bounds.
    - Note: margin for boosting is *not* the same as margin for SVM.
    - A bit like retrofitting the theory…
  - However, those bounds are too loose to be of practical value.
- Different explanation (Friedman, Hastie, Tibshirani, 2000)
  - Interpretation as sequential minimization of an exponential error function ("Forward Stagewise Additive Modeling").
  - Explains why boosting works well.
  - Improvements possible by altering the error function.

AdaBoost – Minimizing Exponential Error

- Exponential error function

$$E = \sum_{n=1}^{N} \exp\left\{-t_n f_m(\mathbf{x}_n)\right\}$$

  - where $f_m(\mathbf{x})$ is a classifier defined as a linear combination of base classifiers $h_l(\mathbf{x})$:

$$f_m(\mathbf{x}) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l h_l(\mathbf{x})$$

- Goal
  - Minimize E with respect to both the weighting coefficients α_l and the parameters of the base classifiers $h_l(\mathbf{x})$.

AdaBoost – Minimizing Exponential Error

- Sequential Minimization
  - Suppose that the base classifiers $h_1(\mathbf{x}),\ldots, h_{m-1}(\mathbf{x})$ and their coefficients $\alpha_1,\ldots,\alpha_{m-1}$ are fixed.

$\Rightarrow$ Only minimize with respect to α_m and $h_m(\mathbf{x})$.

$$E = \sum_{n=1}^{N} \exp\left\{-t_n f_m(\mathbf{x}_n)\right\} \quad \text{with} \quad f_m(\mathbf{x}) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l h_l(\mathbf{x})$$

$$= \sum_{n=1}^{N} \exp\left\{\underbrace{-t_n f_{m-1}(\mathbf{x}_n)}_{= \textit{const.}} - \frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\}$$

$$= \sum_{n=1}^{N} w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\}$$

AdaBoost – Minimizing Exponential Error

$$E = \sum_{n=1}^{N} w_n^{(m)} \exp\left\{ -\frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n) \right\}$$

- Observation:
  - Correctly classified points: $t_n h_m(\mathbf{x}_n) = +1$ $\Rightarrow$ collect in $\mathcal{T}_m$
  - Misclassified points: $t_n h_m(\mathbf{x}_n) = -1$ $\Rightarrow$ collect in $\mathcal{F}_m$
- Rewrite the error function as

$$E = e^{-\alpha_m/2} \sum_{n \in \mathcal{T}_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in \mathcal{F}_m} w_n^{(m)}$$

$$= \left(e^{\alpha_m/2} \qquad\qquad \right) \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n)$$

Machine Learning Winter '19

AdaBoost – Minimizing Exponential Error

$$E = \sum_{n=1}^{N} w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\}$$

- Observation:
  - Correctly classified points: $t_n h_m(\mathbf{x}_n) = +1$ $\Rightarrow$ collect in $\mathcal{T}_m$
  - Misclassified points: $t_n h_m(\mathbf{x}_n) = -1$ $\Rightarrow$ collect in $\mathcal{F}_m$
- Rewrite the error function as

$$E = e^{-\alpha_m/2} \sum_{n \in \mathcal{T}_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in \mathcal{F}_m} w_n^{(m)}$$

$$= \left(e^{\alpha_m/2} - e^{-\alpha_m/2}\right) \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}$$

AdaBoost – Minimizing Exponential Error

- Minimize with respect to $h_m(\mathbf{x})$: $\frac{\partial E}{\partial h_m(\mathbf{x}_n)} \stackrel{!}{=} 0$

$$E = \underbrace{\left(e^{\alpha_m/2} - e^{-\alpha_m/2}\right)}_{= \textit{const.}} \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) + \underbrace{e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}}_{= \textit{const.}}$$

$\Rightarrow$ This is equivalent to minimizing

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n)$$

(our weighted error function from step 2a) of the algorithm)

$\Rightarrow$ *We're on the right track. Let's continue…*

AdaBoost – Minimizing Exponential Error

- Minimize with respect to α_m: $\frac{\partial E}{\partial \alpha_m} \stackrel{!}{=} 0$

$$E = \left(e^{\alpha_m/2} - e^{-\alpha_m/2}\right) \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}$$

$$\left(\frac{1}{\cancel{2}} e^{\alpha_m/2} + \frac{1}{\cancel{2}} e^{-\alpha_m/2}\right) \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) \stackrel{!}{=} \frac{1}{\cancel{2}} e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}$$

weighted error $\epsilon_m :=$

$$\frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} = \frac{e^{-\alpha_m/2}}{e^{\alpha_m/2} + e^{-\alpha_m/2}}$$

$$\epsilon_m = \frac{1}{e^{\alpha_m} + 1}$$

$\Rightarrow$ *Update for the α coefficients:*

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}$$

AdaBoost – Minimizing Exponential Error

- Remaining step: update the weights
  - Recall that

$$E = \sum_{n=1}^{N} \underbrace{w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\}}_{\text{This becomes } w_n^{(m+1)} \text{ in the next iteration.}}$$

  - Therefore

$$\begin{aligned} w_n^{(m+1)} &= w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\} \\ &= \ldots \\ &= w_n^{(m)} \exp\left\{\alpha_m I(h_m(\mathbf{x}_n) \neq t_n)\right\} \end{aligned}$$

$\Rightarrow$ *Update for the weight coefficients.*

AdaBoost – Final Algorithm

1. Initialization: Set $w_n^{(1)} = \frac{1}{N}$ for $n = 1,\dots,N$.
2. For $m = 1,\dots,M$ iterations
   a) Train a new weak classifier $h_m(\mathbf{x})$ using the current weighting coefficients $\mathbf{W}^{(m)}$ by minimizing the weighted error function

   $$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n)$$

   b) Estimate the weighted error of this classifier on $\mathbf{X}$:

   $$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

   c) Calculate a weighting coefficient for $h_m(\mathbf{x})$:

   $$\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}$$

   d) Update the weighting coefficients:

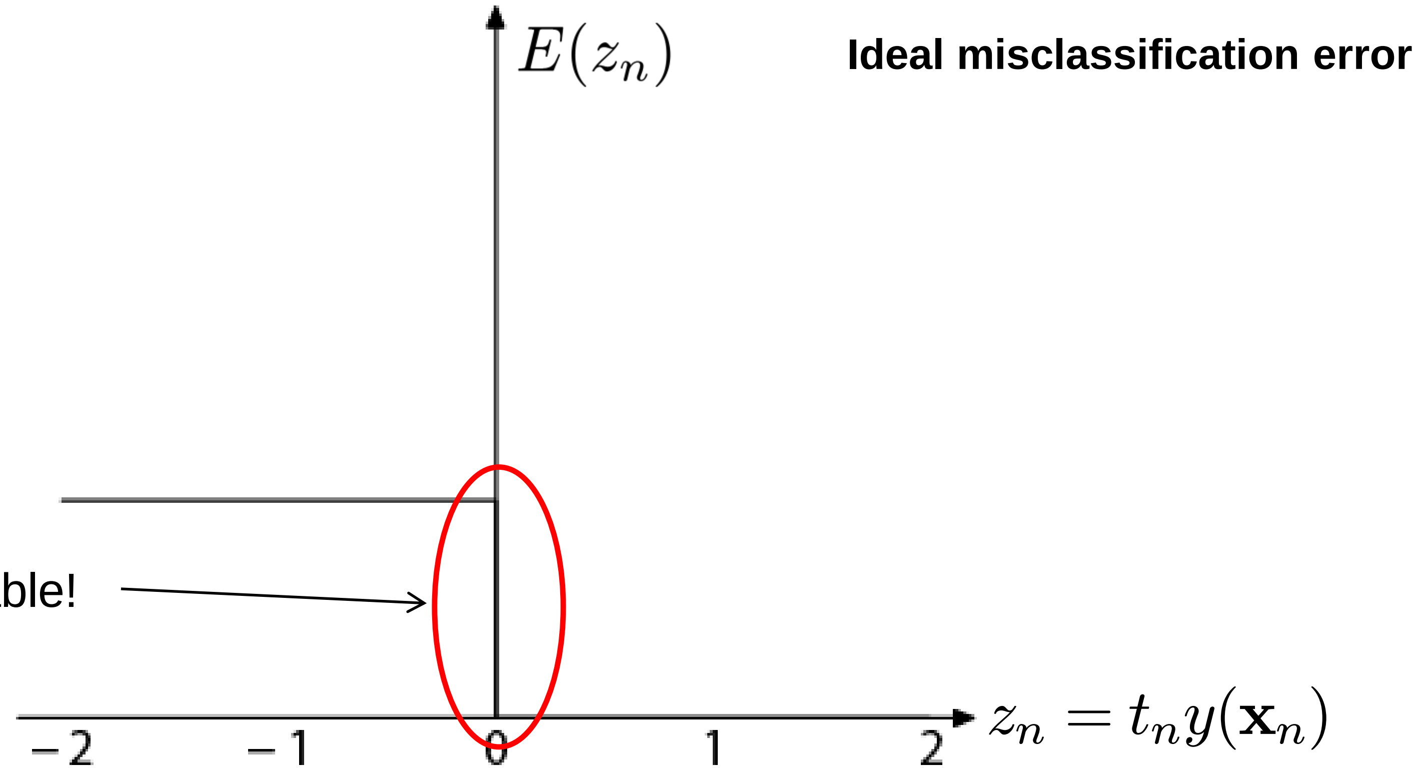
   $$w_n^{(m+1)} = w_n^{(m)} \exp\left\{\alpha_m I(h_m(\mathbf{x}_n) \neq t_n)\right\}$$

AdaBoost – Analysis

- Result of this derivation
  - We now know that AdaBoost minimizes an exponential error function in a sequential fashion.
  - This allows us to analyze AdaBoost's behavior in more detail.
  - In particular, we can see how robust it is to outlier data points.

Recap: Error Functions

$t_n \in \{-1, 1\}$



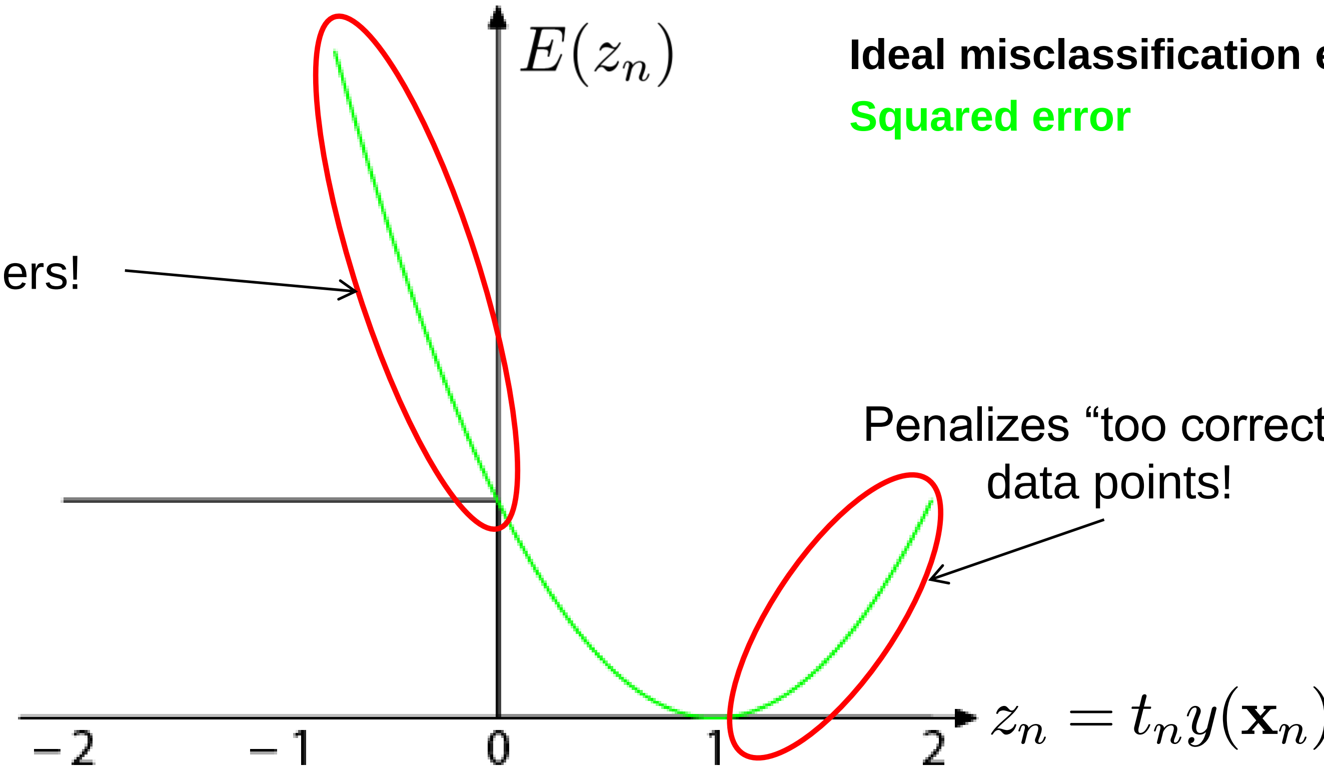
- Ideal misclassification error function (black)
  - This is what we want to approximate,
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.

  $\Rightarrow$ We cannot minimize it by gradient descent.

Image source: Bishop, 2006

Recap: Error Functions

$t_n \in \{-1, 1\}$

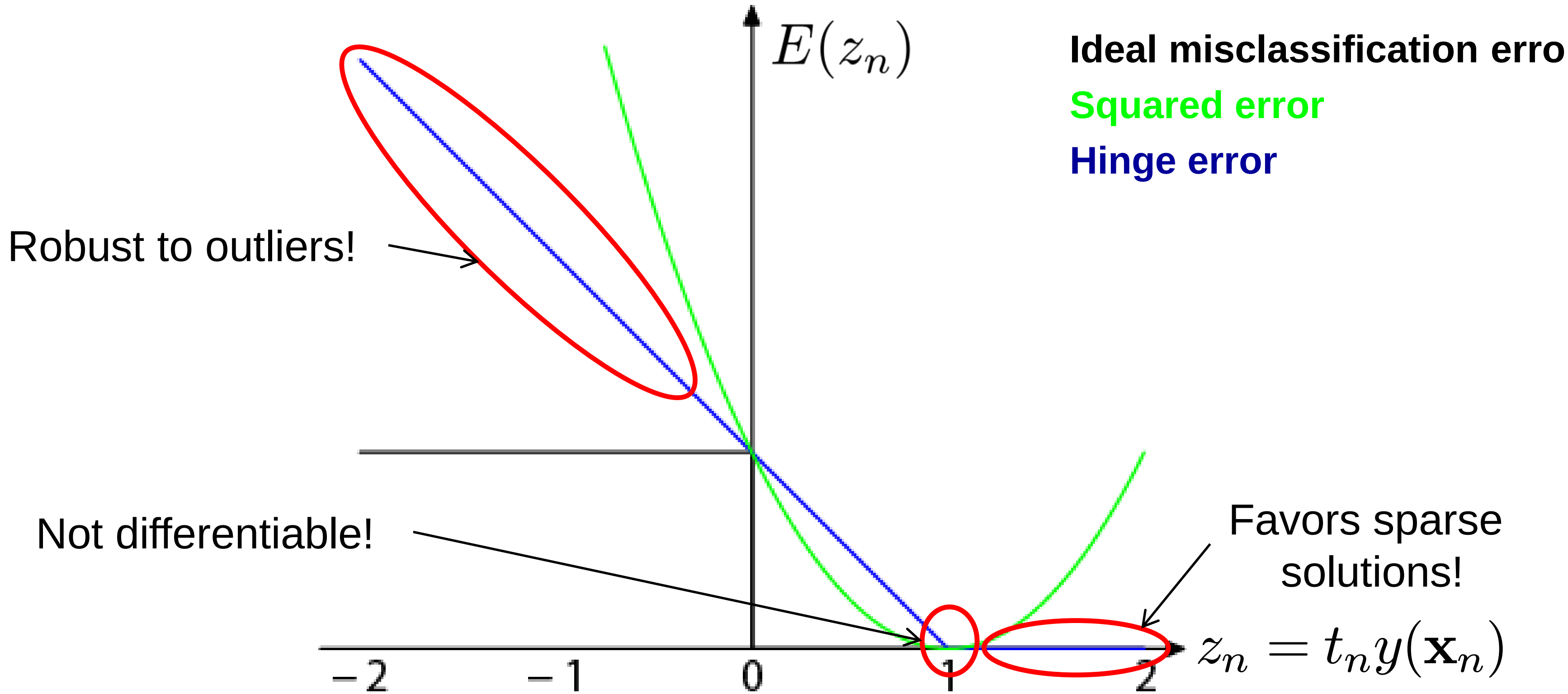


- Squared error used in Least-Squares Classification
  - Very popular, leads to closed-form solutions.
  - However, sensitive to outliers due to squared penalty.
  - Penalizes "too correct" data points

  $\Rightarrow$ Generally does not lead to good classifiers.

Image source: Bishop, 2006

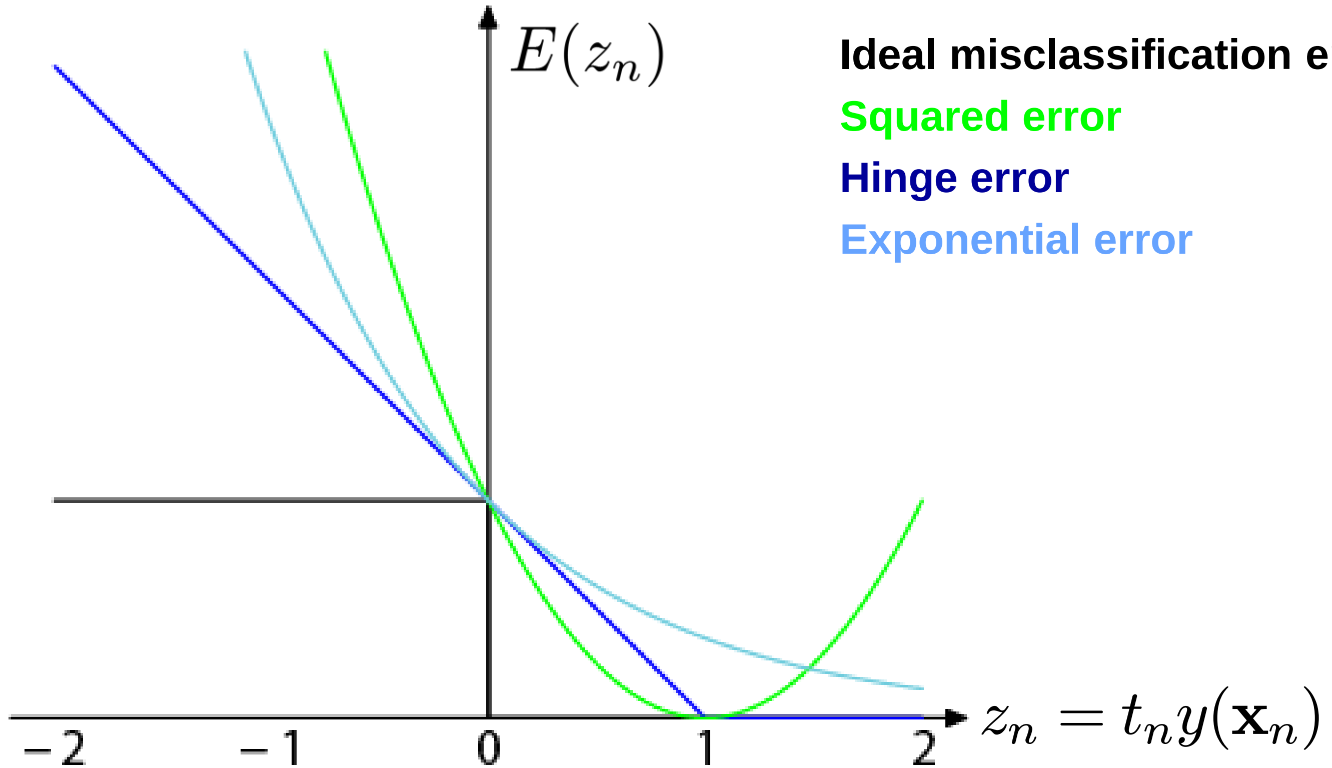
Recap: Error Functions



- "Hinge error" used in SVMs
  - Zero error for points outside the margin ($z_n > 1$) $\Rightarrow$ sparsity
  - Linear penalty for misclassified points ($z_n < 1$) $\Rightarrow$ robustness
  - Not differentiable around $z_n = 1$ $\Rightarrow$ Cannot be optimized directly.

Image source: Bishop, 2006

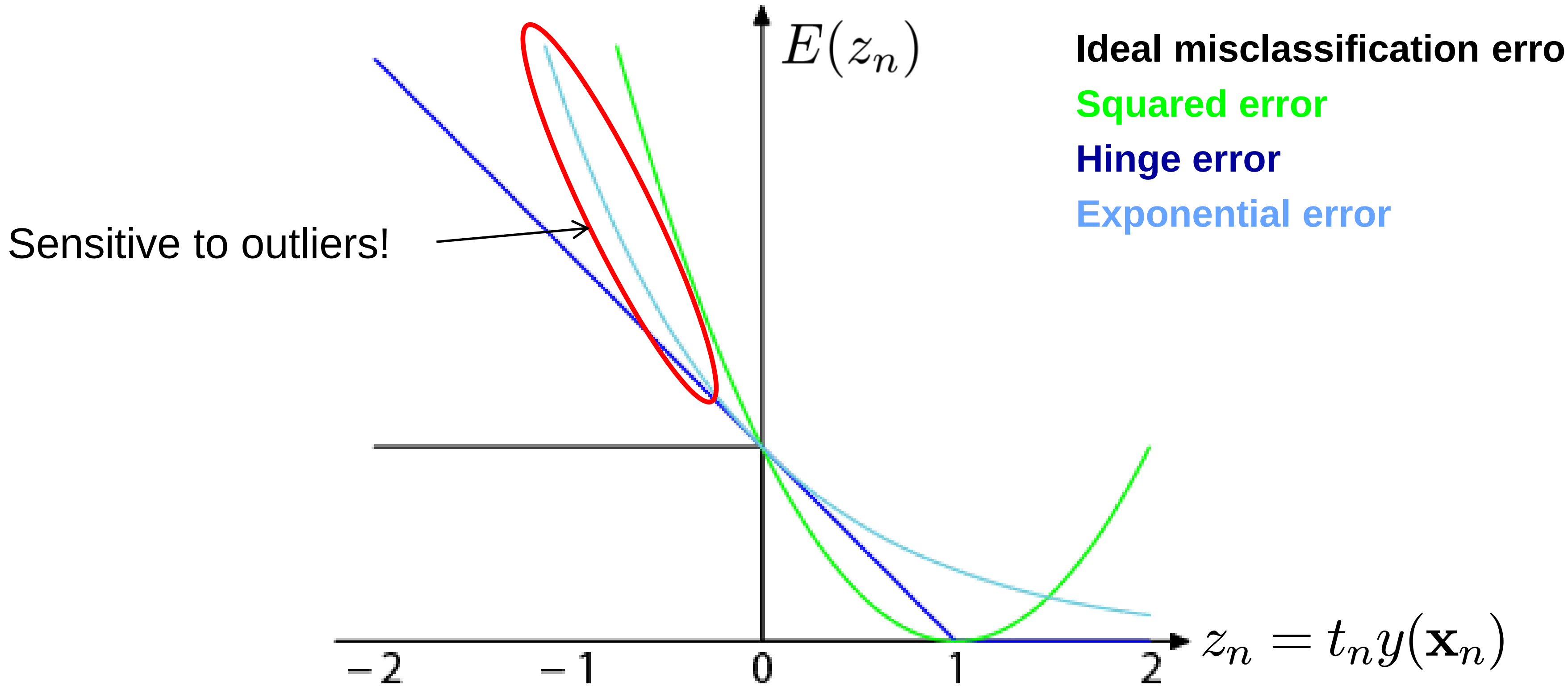
Discussion: AdaBoost Error Function



- Exponential error used in AdaBoost
  - Continuous approximation to ideal misclassification function.
  - Sequential minimization leads to simple AdaBoost scheme.
  - Properties?

Image source: Bishop, 2006

Discussion: AdaBoost Error Function

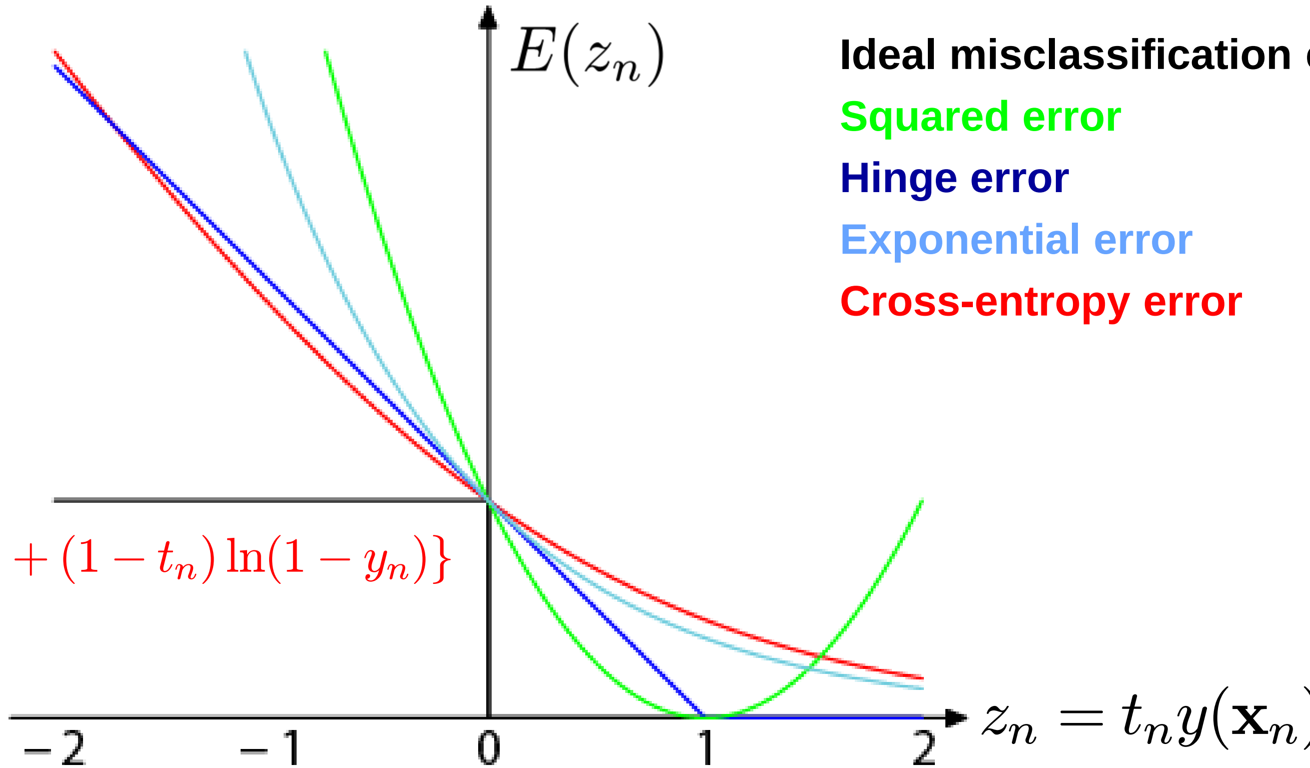


- Exponential error used in AdaBoost
  - No penalty for too correct data points, fast convergence.
  - Disadvantage: exponential penalty for large negative values!

  $\Rightarrow$ Less robust to outliers or misclassified data points!

Image source: Bishop, 2006

Discussion: Other Possible Error Functions



$$E = -\sum \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

- “Cross-entropy error” used in Logistic Regression
  - Similar to exponential error for $z>0$.
  - Only grows linearly with large negative values of z.

$\Rightarrow$ Make AdaBoost more robust by switching to this error function.

$\Rightarrow$ “GentleBoost”

Image source: Bishop, 2006

Summary: AdaBoost

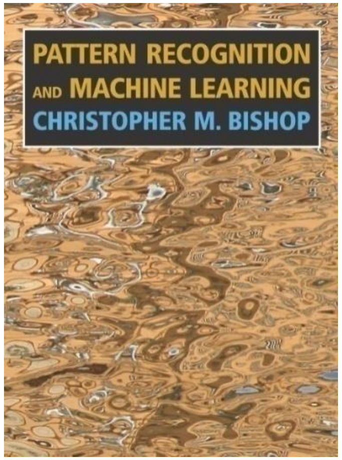
- Properties
  - Simple combination of multiple classifiers.
  - Easy to implement.
  - Can be used with many different types of classifiers.
    - None of them needs to be too good on its own.
    - In fact, they only have to be slightly better than chance.
  - Commonly used in many areas.
  - Empirically good generalization capabilities.

- Limitations
  - Original AdaBoost sensitive to misclassified training data points.
    - Because of exponential error function.
    - Improvement by GentleBoost
  - Single-class classifier
    - Multiclass extensions available

References and Further Reading

- More information on Classifier Combination and Boosting can be found in Chapters 14.1-14.3 of Bishop's book.

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



- A more in-depth discussion of the statistical interpretation of AdaBoost is available in the following paper:
  - J. Friedman, T. Hastie, R. Tibshirani, Additive Logistic Regression: a Statistical View of Boosting, *The Annals of Statistics*, Vol. 38(2), pages 337-374, 2000.