Machine Learning – Lecture 9

AdaBoost

19.11.2017

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

• Analysis
  - Error function

• Applications

• 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

\[ \mathbf{w}^T \mathbf{x} + b = 0 \]

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

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

under the constraints

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

based on training data points \( \mathbf{x}_n \) and target values \( t_n \in \{-1, 1\} \)
Recap: SVM – Dual Formulation

• Maximize

\[ L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_m^T x_n) \]

under the conditions

\[ a_n \geq 0 \quad \forall n \]

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

• 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: SVM for Non-Separable Data

- Slack variables
  - One slack variable $\xi_n \geq 0$ for each training data point.

- Interpretation
  - $\xi_n = 0$ for points that are on the correct side of the margin.
  - $\xi_n = |t_n - y(x_n)|$ for all other points.
  - We do not have to set the slack variables ourselves!
    $\Rightarrow$ They are jointly optimized together with $w$. 

Point on decision boundary: $\xi_n = 1$

Misclassified point: $\xi_n > 1$
Recap: SVM – New Dual Formulation

• New SVM Dual: Maximize

\[ L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_na_mt_nt_m(x_m^T x_n) \]

under the conditions

\[ 0 \cdot a_n \cdot C \]

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

• This is again a quadratic programming problem

⇒ Solve as before…

This is all that changed!
Recap: Nonlinear SVMs

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

\[ \Phi: x \rightarrow \phi(x) \]
Recap: The Kernel Trick

• Important observation
  - $\phi(x)$ only appears in the form of dot products $\phi(x)^T \phi(y)$:
    $$ y(x) = w^T \phi(x) + b $$
    $$ = \sum_{n=1}^{N} a_n t_n \phi(x_n)^T \phi(x) + b $$
  - Define a so-called kernel function $k(x,y) = \phi(x)^T \phi(y)$.
  - Now, in place of the dot product, use the kernel instead:
    $$ y(x) = \sum_{n=1}^{N} a_n t_n k(x_n, x) + b $$
  - The kernel function *implicitly* maps the data to the higher-dimensional space (without having to compute $\phi(x)$ explicitly)!
Recap: Nonlinear SVM – Dual Formulation

• SVM Dual: Maximize

\[ L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_m, x_n) \]

under the conditions

\[ 0 \cdot a_n \cdot C \]

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

• Classify new data points using

\[ y(x) = \sum_{n=1}^{N} a_n t_n k(x_n, x) + b \]
Topics of This Lecture

• Recap: Nonlinear Support Vector Machines

• Analysis
  - Error function

• Applications

• Ensembles of classifiers
  - Bagging
  - Bayesian Model Averaging

• AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions
SVM – Analysis

• Traditional soft-margin formulation

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

subject to the constraints

$$t_n y(x_n) \geq 1 - \xi_n$$

“Maximize the margin”

“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.

$$\min_{w \in \mathbb{R}^D} \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{N} [1 - t_n y(x_n)]_+$$

L₂ regularizer

“Hinge loss”

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

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

⇒ We cannot minimize it by gradient descent.
Recap: Error Functions

\[ t_n \in \{-1, 1\} \]

- Sensitive to outliers!

- 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
  - Generally does not lead to good classifiers.

Ideal misclassification error

Squared error

\[ z_n = t_n y(x_n) \]
Error Functions (Loss Functions)

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

Ideal misclassification error

Squared error

Hinge error

Robust to outliers!

Not differentiable!

\[ E(z_n) \]

\[ z_n = t_n y(x_n) \]
SVM – Discussion

• SVM optimization function

\[
\min_{\mathbf{w} \in \mathbb{R}^D} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^{N} [1 - t_n y(\mathbf{x}_n)]_+
\]

L₂ regularizer \quad \text{Hinge loss}

• Hinge loss enforces sparsity
  
  • Only a subset of training data points actually influences the decision boundary.
  • This is different from sparsity obtained through the regularizer! There, only a subset of input dimensions are used.
  • Unconstrained optimization, but non-differentiable function.
  • Solve, e.g. by subgradient descent
  • Currently most efficient: stochastic gradient descent

Slide adapted from Christoph Lampert
Topics of This Lecture

• Recap: Nonlinear Support Vector Machines

• Analysis
  ➢ Error function

• Applications

• Ensembles of classifiers
  ➢ Bagging
  ➢ Bayesian Model Averaging

• AdaBoost
  ➢ Intuition
  ➢ Algorithm
  ➢ Analysis
  ➢ Extensions
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

- Results:

<table>
<thead>
<tr>
<th></th>
<th>Bayes</th>
<th>Rocchio</th>
<th>C4.5</th>
<th>k-NN</th>
<th>SVM (poly) degree $d =$</th>
<th>SVM (rbf) width $\gamma =$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0.6</td>
</tr>
<tr>
<td>earn</td>
<td>95.9</td>
<td>96.1</td>
<td>96.1</td>
<td>97.3</td>
<td>98.2</td>
<td>98.4</td>
</tr>
<tr>
<td>acq</td>
<td>91.5</td>
<td>92.1</td>
<td>85.3</td>
<td>92.0</td>
<td>92.6</td>
<td>94.6</td>
</tr>
<tr>
<td>money-fx</td>
<td>62.9</td>
<td>67.6</td>
<td>69.4</td>
<td>78.2</td>
<td>66.9</td>
<td>72.5</td>
</tr>
<tr>
<td>grain</td>
<td>72.5</td>
<td>79.5</td>
<td>89.1</td>
<td>82.2</td>
<td>91.3</td>
<td>93.1</td>
</tr>
<tr>
<td>crude</td>
<td>81.0</td>
<td>81.5</td>
<td>75.5</td>
<td>85.7</td>
<td>86.0</td>
<td>87.3</td>
</tr>
<tr>
<td>trade</td>
<td>50.0</td>
<td>77.4</td>
<td>59.2</td>
<td>77.4</td>
<td>69.2</td>
<td>75.5</td>
</tr>
<tr>
<td>interest</td>
<td>58.0</td>
<td>72.5</td>
<td>49.1</td>
<td>74.0</td>
<td>69.8</td>
<td>63.3</td>
</tr>
<tr>
<td>ship</td>
<td>78.7</td>
<td>83.1</td>
<td>80.9</td>
<td>79.2</td>
<td>82.0</td>
<td>85.4</td>
</tr>
<tr>
<td>wheat</td>
<td>60.6</td>
<td>79.4</td>
<td>85.5</td>
<td>76.6</td>
<td>83.1</td>
<td>84.5</td>
</tr>
<tr>
<td>corn</td>
<td>47.3</td>
<td>62.2</td>
<td>87.7</td>
<td>77.9</td>
<td>86.0</td>
<td>86.5</td>
</tr>
<tr>
<td>microavg.</td>
<td><strong>72.0</strong></td>
<td><strong>79.9</strong></td>
<td><strong>79.4</strong></td>
<td><strong>82.3</strong></td>
<td><strong>84.2</strong></td>
<td><strong>85.1</strong></td>
</tr>
</tbody>
</table>

B. Leibe
Example Application: Text Classification

- This is also how you could implement a simple spam filter...
Example Application: OCR

- Handwritten digit recognition
  - US Postal Service Database
  - Standard benchmark task for many learning algorithms
Historical Importance

- USPS benchmark
  - 2.5% error: human performance

- Different learning algorithms
  - 16.2% error: Decision tree (C4.5)
  - 5.9% error: (best) 2-layer Neural Network
  - 5.1% error: LeNet 1 – (massively hand-tuned) 5-layer network

- Different SVMs
  - 4.0% error: Polynomial kernel (p=3, 274 support vectors)
  - 4.1% error: Gaussian kernel (σ=0.3, 291 support vectors)
Example Application: OCR

- Results
  - Almost no overfitting with higher-degree kernels.

<table>
<thead>
<tr>
<th>degree of polynomial</th>
<th>dimensionality of feature space</th>
<th>support vectors</th>
<th>raw error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>256</td>
<td>282</td>
<td>8.9</td>
</tr>
<tr>
<td>2</td>
<td>≈ 33000</td>
<td>227</td>
<td>4.7</td>
</tr>
<tr>
<td>3</td>
<td>≈ 1 \times 10^6</td>
<td>274</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>≈ 1 \times 10^9</td>
<td>321</td>
<td>4.2</td>
</tr>
<tr>
<td>5</td>
<td>≈ 1 \times 10^{12}</td>
<td>374</td>
<td>4.3</td>
</tr>
<tr>
<td>6</td>
<td>≈ 1 \times 10^{14}</td>
<td>377</td>
<td>4.5</td>
</tr>
<tr>
<td>7</td>
<td>≈ 1 \times 10^{16}</td>
<td>422</td>
<td>4.5</td>
</tr>
</tbody>
</table>
Example Application: Object Detection

• Sliding-window approach

  - E.g. histogram representation (HOG)
    - Map each grid cell in the input window to a histogram of gradient orientations.
    - Train a linear SVM using training set of pedestrian vs. non-pedestrian windows.

[Dalal & Triggs, CVPR 2005]
Example Application: Pedestrian Detection

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

• Recap: Nonlinear Support Vector Machines

• Analysis
  ➢ Error function

• Applications

• 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 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 multiple 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
    \[
    p(X) = \sum_{h=1}^{H} p(X|h) p(h)
    \]

• 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 $p(X|h)$ 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.
  ⇒ One latent variable $z_n$ for each data point:

$$p(X) = \prod_{n=1}^{N} p(x_n) = \prod_{n=1}^{N} \sum_{z_n} p(x_n, z_n)$$

• Bayesian Model Averaging
  ➢ The whole data set is generated by a single model.
  ➢ Uncertainty is about which model was responsible.
  ⇒ One latent variable $z$ for the entire data set:

$$p(X) = \sum_{z} p(X, z)$$
Model Averaging: Expected Error

• Combine $M$ predictors $y_m(x)$ for target output $h(x)$.
  - E.g. each trained on a different bootstrap data set by bagging.
  - The committee prediction is given by
    \[
    y_{COM}(x) = \frac{1}{M} \sum_{m=1}^{M} y_m(x)
    \]
  - The output can be written as the true value plus some error.
    \[
    y(x) = h(x) + \epsilon(x)
    \]
  - Thus, the expected sum-of-squares error takes the form
    \[
    \mathbb{E}_x = \left[ \left\{ y_m(x) - h(x) \right\}^2 \right] = \mathbb{E}_x \left[ \epsilon_m(x)^2 \right]
    \]
Model Averaging: Expected Error

- Average error of individual models
  \[ \mathbb{E}_{AV} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_x [\epsilon_m(x)^2] \]

- Average error of committee
  \[ \mathbb{E}_{COM} = \mathbb{E}_x \left[ \left\{ \frac{1}{M} \sum_{m=1}^{M} y_m(x) - h(x) \right\}^2 \right] = \mathbb{E}_x \left[ \left\{ \frac{1}{M} \sum_{m=1}^{M} \epsilon_m(x) \right\}^2 \right] \]

- Assumptions
  - Errors have zero mean: \( \mathbb{E}_x [\epsilon_m(x)] = 0 \)
  - Errors are uncorrelated: \( \mathbb{E}_x [\epsilon_m(x)\epsilon_j(x)] = 0 \)

- Then:
  \[ \mathbb{E}_{COM} = \frac{1}{M} \mathbb{E}_{AV} \]
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**
  
  - 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**
  
  - $h_m(x)$: “weak” or base classifier
    - Condition: <50% training error over any distribution
  - $H(x)$: “strong” or final classifier

• **AdaBoost:**
  
  - Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:

  \[
  H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right)
  \]
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.
AdaBoost: Intuition

- Weak Classifier 1
- Weights Increased
- Weak Classifier 2

Slide credit: Kristen Grauman
Figure adapted from Freund & Schapire
AdaBoost: Intuition

The final classifier is a linear combination of the weak classifiers.
AdaBoost – Formalization

• 2-class classification problem
  ➢ Given: training set $X = \{x_1, \ldots, x_N\}$
    with target values $T = \{t_1, \ldots, t_N\}$, $t_n \in \{-1, 1\}$.
  ➢ Associated weights $W = \{w_1, \ldots, w_N\}$ for each training point.

• Basic steps
  ➢ In each iteration, AdaBoost trains a new weak classifier $h_m(x)$ based on the current weighting coefficients $W^{(m)}$.
  ➢ We then adapt the weighting coefficients for each point
    – Increase $w_n$ if $x_n$ was misclassified by $h_m(x)$.
    – Decrease $w_n$ if $x_n$ was classified correctly by $h_m(x)$.
  ➢ Make predictions using the final combined model
    $$H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right)$$
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 \( h_m(x) \) using the current weighting coefficients \( W^{(m)} \) by minimizing the weighted error function

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

   b) Estimate the weighted error of this classifier on \( X \):

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

   c) Calculate a weighting coefficient for \( h_m(x) \):

   \[
   \alpha_m = ?
   \]

   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

\[ E = \sum_{n=1}^{N} \exp \left\{ -t_n f_m(x_n) \right\} \]

where \( f_m(x) \) is a classifier defined as a linear combination of base classifiers \( h_l(x) \):

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

• Goal

  - Minimize \( E \) with respect to both the weighting coefficients \( \alpha_l \) and the parameters of the base classifiers \( h_l(x) \).
AdaBoost – Minimizing Exponential Error

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

  ⇒ Only minimize with respect to $\alpha_m$ and $h_m(x)$.

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

  $$
  = \sum_{n=1}^{N} \exp \left\{ -t_n f_{m-1}(x_n) - \frac{1}{2} t_n \alpha_m h_m(x_n) \right\}
  $$

  $$
  = \text{const.}
  $$

  $$
  = \sum_{n=1}^{N} w_n^{(m)} \exp \left\{ -\frac{1}{2} t_n \alpha_m h_m(x_n) \right\}
  $$
AdaBoost – Minimizing Exponential Error

\[ E = \sum_{n=1}^{N} w_{n}^{(m)} \exp \left\{ -\frac{1}{2} t_n \alpha_m h_m(x_n) \right\} \]

- Observation:
  - Correctly classified points: \( t_n h_m(x_n) = +1 \) \( \Rightarrow \) collect in \( \mathcal{T}_m \)
  - Misclassified points: \( t_n h_m(x_n) = -1 \) \( \Rightarrow \) collect in \( \mathcal{F}_m \)

- Rewrite the error function as
  \[
  E = e^{-\alpha_m/2} \sum_{n \in \mathcal{T}_m} w_{n}^{(m)} + e^{\alpha_m/2} \sum_{n \in \mathcal{F}_m} w_{n}^{(m)}
  \]

  \[= \left(e^{\alpha_m/2}\right)^{N} \sum_{n=1}^{N} w_{n}^{(m)} I(h_m(x_n) \neq t_n)\]
AdaBoost – Minimizing Exponential Error

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

- Observation:
  - Correctly classified points: $t_n h_m(x_n) = +1$ \(\Rightarrow\) collect in $T_m$
  - Misclassified points: $t_n h_m(x_n) = -1$ \(\Rightarrow\) collect in $F_m$

- Rewrite the error function as

$$E = e^{-\alpha_m/2} \sum_{n \in T_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in F_m} w_n^{(m)}$$

$$= \left( e^{\alpha_m/2} - e^{-\alpha_m/2} \right) \sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}$$
AdaBoost – Minimizing Exponential Error

- Minimize with respect to $h_m(x)$: 

$$
\frac{\partial E}{\partial h_m(x_n)} \neq 0
$$

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

= const.

⇒ This is equivalent to minimizing

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

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

⇒ We’re on the right track. Let’s continue…

B. Leibe
AdaBoost – Minimizing Exponential Error

• Minimize with respect to $\alpha_m$: $\frac{\partial E}{\partial \alpha_m} \neq 0$

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

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

weighted error $\epsilon_m := \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} = \frac{e^{-\alpha_m/2}}{e^{\alpha_m/2} + e^{-\alpha_m/2}}$

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

$\Rightarrow$ Update for the $\alpha$ coefficients:

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}$$
AdaBoost – Minimizing Exponential Error

• Remaining step: update the weights
  
  - Recall that
    \[ E = \sum_{n=1}^{N} w_n^{(m)} \exp \left\{ -\frac{1}{2} t_n \alpha_m h_m(x_n) \right\} \]

  This becomes
  \[ w_n^{(m+1)} \]
  in the next iteration.

  - Therefore
    \[ w_n^{(m+1)} = w_n^{(m)} \exp \left\{ -\frac{1}{2} t_n \alpha_m h_m(x_n) \right\} = \ldots \]
    \[ = w_n^{(m)} \exp \{ \alpha_m I(h_m(x_n) \neq t_n) \} \]

  \[ \Rightarrow \text{Update for the weight coefficients.} \]
AdaBoost – Final Algorithm

1. Initialization: Set \( w_n^{(1)} = \frac{1}{N} \) or \( n = 1, \ldots, N \).

2. For \( m = 1, \ldots, M \) iterations
   a) Train a new weak classifier \( h_m(x) \) using the current weighting coefficients \( W^{(m)} \) by minimizing the weighted error function
      \[
      J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)
      \]
   b) Estimate the weighted error of this classifier on \( X \):
      \[
      \epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}
      \]
   c) Calculate a weighting coefficient for \( h_m(x) \):
      \[
      \alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}
      \]
   d) Update the weighting coefficients:
      \[
      w_n^{(m+1)} = w_n^{(m)} \exp \{ \alpha_m I(h_m(x_n) \neq t_n) \}
      \]
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.
Recap: Error Functions

\[ t_n \in \{-1, 1\} \]

Sensitive to outliers!

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

⇒ Generally does not lead to good classifiers.
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.

\[ z_n = t_n y(x_n) \]

Ideal misclassification error
Squared error
Hinge error

Robust to outliers!
Not differentiable!
Favors sparse solutions!

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?

**Exponential error**

\[ z_n = t_n y(x_n) \]

- Ideal misclassification error
- Squared error
- Hinge error
- Exponential error

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!
  - Less robust to outliers or misclassified data points!

\[ z_n = t_n y(x_n) \]

Ideal misclassification error
- Squared error
- Hinge error
- Exponential error

Sensitive to outliers!
Discussion: Other Possible Error Functions

- "Cross-entropy error" used in Logistic Regression
  - Similar to exponential error for $z > 0$.
  - Only grows linearly with large negative values of $z$.
  - Make AdaBoost more robust by switching to this error function.
  - “GentleBoost”

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

\[ z_n = t_n y(x_n) \]

Ideal misclassification error
Square error
Hinge error
Exponential error
Cross-entropy error
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:
  