Machine Learning - Lecture 8

Model Combination & Boosting

12.05.2011

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

leibe@umic.rwth-aachen.de

Many slides adapted from B. Schiele
Course Outline

- **Fundamentals (2 weeks)**
  - Bayes Decision Theory
  - Probability Density Estimation

- **Discriminative Approaches (5 weeks)**
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Randomized Trees, Forests & Ferns

- **Generative Models (4 weeks)**
  - Bayesian Networks
  - Markov Random Fields
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 $w$. 
Recap: SVM - New Dual Formulation

- New SVM Dual: Maximize

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

under the conditions

\[ 0 \leq a_n \leq C \]

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

- 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:

\[ \Phi: \mathbf{x} \rightarrow \phi(\mathbf{x}) \]
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: Kernels Fulfilling Mercer’s Condition

- **Polynomial kernel**
  
  \[ k(x, y) = (x^T y + 1)^p \]

- **Radial Basis Function kernel**
  
  \[ k(x, y) = \exp \left\{ -\frac{(x - y)^2}{2\sigma^2} \right\} \]  
  e.g. Gaussian

- **Hyperbolic tangent kernel**
  
  \[ k(x, y) = \tanh(\kappa x^T y + \delta) \]  
  e.g. Sigmoid

(and many, many more...)
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

\[ 0 \leq a_n \leq C \]

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

- **Classify new data points using**

\[ y(x) = \sum_{n=1}^{N} a_n t_n k(x_n, x) + b \]
Summary: SVMs

- Properties
  - Empirically, SVMs work very, very well.
  - SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
  - SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
  - SVM techniques have been applied to a variety of other tasks
    - e.g. SV Regression, One-class SVMs, ...
  - The kernel trick has been used for a wide variety of applications. It can be applied wherever dot products are in use
    - e.g. Kernel PCA, kernel FLD, ...
    - Good overview, software, and tutorials available on [http://www.kernel-machines.org/](http://www.kernel-machines.org/)
Summary: SVMs

- Limitations
  - How to select the right kernel?
    - Still something of a black art...
  - How to select the kernel parameters?
    - (Massive) cross-validation.
    - Usually, several parameters are optimized together in a grid search.
  - Solving the quadratic programming problem
    - Standard QP solvers do not perform too well on SVM task.
    - Dedicated methods have been developed for this, e.g. SMO.
  - Speed of evaluation
    - Evaluating $y(x)$ scales linearly in the number of SVs.
    - Too expensive if we have a large number of support vectors.
    - There are techniques to reduce the effective SV set.
  - Training for very large datasets (millions of data points)
    - Still problematic...
Example Application: Text Classification

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

  ![Document icon]

  ![Question mark]

• 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></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</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>84.2</td>
<td>85.1</td>
</tr>
</tbody>
</table>

Combined: **86.4**
Example Application: Text Classification

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

Diagram:
- Incoming email
- Dictionary
- Word activations
- SVM
- Mailbox
- Trash
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>$\approx 33000$</td>
<td>227</td>
<td>4.7</td>
</tr>
<tr>
<td>3</td>
<td>$\approx 1 \times 10^6$</td>
<td>274</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>$\approx 1 \times 10^9$</td>
<td>321</td>
<td>4.2</td>
</tr>
<tr>
<td>5</td>
<td>$\approx 1 \times 10^{12}$</td>
<td>374</td>
<td>4.3</td>
</tr>
<tr>
<td>6</td>
<td>$\approx 1 \times 10^{14}$</td>
<td>377</td>
<td>4.5</td>
</tr>
<tr>
<td>7</td>
<td>$\approx 1 \times 10^{16}$</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
Many Other Applications

- Lots of other applications in all fields of technology
  - OCR
  - Text classification
  - Computer vision
  - ... 
  - High-energy physics
  - Monitoring of household appliances
  - Protein secondary structure prediction
  - Design on decision feedback equalizers (DFE) in telephony

(Detailed references in Schoelkopf & Smola, 2002, pp. 221)
You Can Try It At Home...

- Lots of SVM software available, e.g.
  - svmlight (http://svmlight.joachims.org/)
    - Command-line based interface
    - Source code available (in C)
    - Interfaces to Python, MATLAB, Perl, Java, DLL,...

  - libsvm (http://www.csie.ntu.edu.tw/~cjlin/libsvm/)
    - Library for inclusion with own code
    - C++ and Java sources
    - Interfaces to Python, R, MATLAB, Perl, Ruby, Weka, C+ .NET,...

- Both include fast training and evaluation algorithms, support for multi-class SVMs, automated training and cross-validation, ...
  ⇒ Easy to apply to your own problems!
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?
Topics of This Lecture

- Ensembles of Classifiers
- Constructing Ensembles
  - Cross-validation
  - Bagging
- Combining Classifiers
  - Stacking
  - Bayesian model averaging
  - Boosting
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions
- Applications
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...
Ensembles of Classifiers

- Example
  - K classifiers with error probability $p = 0.3$.
  - Probability that exactly L classifiers make an error:
    \[ p^L (1 - p)^{K-L} \]
  - The probability that 11 or more classifiers make an error is 0.026.
Topics of This Lecture

- Ensembles of Classifiers
- Constructing Ensembles
  - Cross-validation
  - Bagging
- Combining Classifiers
  - Stacking
  - Bayesian Model Averaging
  - Boosting
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions
- Applications

Methods for obtaining a set of classifiers

Methods for combining different classifiers
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

- Cross-Validation
  - Split the available data into N disjunct subsets.
  - In each run, train on N-1 subsets for training a classifier.
  - Estimate the generalization error on the held-out validation set.

- E.g. 5-fold cross-validation

```
| train | train | train | train | test |
| train | train | train | test  | train |
| train | train | test  | train | train |
| train | test  | train | train | train |
| test  | train | train | train | train |
```

B. Leibe
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.
Topics of This Lecture

- Ensembles of Classifiers
- Constructing Ensembles
  - Cross-validation
  - Bagging
- Combining Classifiers
  - Stacking
  - Bayesian Model Averaging
  - Boosting
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions
- Applications

Methods for obtaining a set of classifiers

Methods for combining different classifiers

B. Leibe
Stacking

- **Idea**
  - Learn $L$ classifiers (based on the training data)
  - Find a meta-classifier that takes as input the output of the $L$ first-level classifiers.

- **Example**
  - Learn $L$ classifiers with leave-one-out cross-validation.
  - Interpret the prediction of the $L$ classifiers as $L$-dimensional feature vector.
  - Learn “level-2” classifier based on the examples generated this way.
Stacking

- Why can this be useful?
  - Simplicity
    - We may already have several existing classifiers available.
      ⇒ No need to retrain those, they can just be combined with the rest.
  - Correlation between classifiers
    - The combination classifier can learn the correlation.
      ⇒ Better results than simple Naïve Bayes combination.
  - Feature combination
    - E.g. combine information from different sensors or sources (vision, audio, acceleration, temperature, radar, etc.).
    - We can get good training data for each sensor individually, but data from all sensors together is rare.
      ⇒ Train each of the L classifiers on its own input data.
        Only combination classifier needs to be trained on combined input.
Recap: Model Combination

- **E.g. Mixture of Gaussians**
  - Several components are combined probabilistically.
  - Interpretation: different data points can be generated by different components.
  - We model the uncertainty which mixture component is responsible for generating the corresponding data point:
    \[
    p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x | \mu_k, \Sigma_k)
    \]
  - For iid data, we write the marginal probability of a data set \( X = \{x_1, \ldots, x_N\} \) in the form:
    \[
    p(X) = \prod_{n=1}^{N} p(x_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)
    \]
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
  - 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 average 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:
  \[ \mathbb{E}_{COM} = \frac{1}{M} \mathbb{E}_{AV} \]

B. Leibe
Model Averaging: Expected Error

- Average error of committee

\[ E_{COM} = \frac{1}{M} E_{AV} \]

- 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

\[ E_{COM} \leq E_{AV} \]
Boosting

- Simple technique with very interesting properties
  - Combination of multiple classifiers with the goal to improve classification accuracy.
  - 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.
    - Extreme case: Decision stumps
      \[
      y(x) = \begin{cases} 
      1, & x_i \geq \theta \\
      0, & \text{else} 
      \end{cases}
      \]

- Main idea
  - Train successive component classifiers on a subset of the training data that is *most informative* given the current set of classifiers.
    \[ \Rightarrow \text{Sequential classifier selection} \]

B. Leibe
Boosting (Schapire 1989)

- Algorithm: (3-component classifier)
  1. Sample $N_1 < N$ training examples (*without replacement*) from training set $\mathcal{D}$ to get set $\mathcal{D}_1$.
     - Train weak classifier $C_1$ on $\mathcal{D}_1$.
  2. Sample $N_2 < N$ training examples (*without replacement*), half of which were misclassified by $C_1$ to get set $\mathcal{D}_2$.
     - Train weak classifier $C_2$ on $\mathcal{D}_2$.
  3. Choose all data in $\mathcal{D}$ on which $C_1$ and $C_2$ disagree to get set $\mathcal{D}_3$.
     - Train weak classifier $C_3$ on $\mathcal{D}_3$.
  4. Get the final classifier output by majority voting of $C_1$, $C_2$, and $C_3$. 

B. Leibe

Image source: Duda, Hart, Stork, 2001
Applying Boosting

• How should we choose the number of samples $N_1$?
  - Ideally, the number of samples should be roughly equal in all 3 component classifiers.
  - Reasonable first guess: $N_1 \approx N/3$
  - However, if the problem is very simple
    - $C_1$ will explain most of the data.
      $\Rightarrow$ $N_2$ and $N_3$ will be very small.
      $\Rightarrow$ Not all of the data will be used effectively.
  - Similarly, if the problem is extremely hard
    - $C_1$ will explain only a small part of the data.
      $\Rightarrow$ $N_2$ may be unacceptably large.
  - In practice, may need to run the boosting procedure a few times and adjust $N_1$ in order to use the full training set.
  - Also, we can recursively apply the procedure on $C_1$ to $C_3$. 

B. Leibe
Discussion: Ensembles of Classifiers

• Set of simple methods for improving classification
  ➢ Often effective in practice.

• Apparent contradiction
  ➢ We have stressed before that a classifier should be trained on samples from the distribution on which it will be tested.
  ➢ Resampling seems to violate this recommendation.
  ➢ *Why can a classifier trained on a weighted data distribution do better than one trained on the i.i.d. sample?*

• Explanation
  ➢ We do not attempt to model the full category distribution here.
  ➢ Instead, try to find the decision boundary more directly.
  ➢ Also, increasing number of component classifiers broadens the class of implementable decision functions.
Topics of This Lecture

- Ensembles of Classifiers
- Constructing Ensembles
  - Cross-validation
  - Bagging
- Combining Classifiers
  - Stacking
  - Bayesian model averaging
  - Boosting
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions
- Applications
AdaBoost - “Adaptive Boosting”

- **Main idea** [Freund & Schapire, 1996]
  - Instead of resampling, 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:
    $$ H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right) $$

B. Leibe
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.

Slide credit: Kristen Grauman
AdaBoost: Intuition

Weights Increased

Weak Classifier 1

Weak Classifier 2

Slide credit: Kristen Grauman

B. Leibe

Figure adapted from Freund & Schapire
AdaBoost: Intuition

Final classifier is combination of the weak classifiers

Slide credit: Kristen Grauman

Figure adapted from Freund & Schapire
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
      \[
      J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)
      \]
      \( I(A) = \begin{cases} 
      1, & \text{if } A \text{ is true} \\
      0, & \text{else}
      \end{cases} \)

   b) Estimate the weighted error of this classifier on \( X \):
      \[
      \epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}
      \]

   c) Calculate a weighting coefficient for \( h_m(x) \):
      \[
      \alpha_m = \frac{1}{2} \ln \frac{1 - \epsilon_m}{\epsilon_m}
      \]

   d) Update the weighting coefficients:
      \[
      w_n^{(m+1)} = \frac{w_n^{(m)} e^{-\alpha_m I(h_m(x) \neq t_n)}}{Z_m}
      \]
      \( Z_m \) is a normalization term.

**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

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

- 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.

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

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

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

$$
= \text{const.}
$$

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

B. Leibe
AdaBoost - Minimizing Exponential Error

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

- **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**
  \[ E = e^{-\alpha_m/2} \sum_{n \in \mathcal{T}_m} w^{(m)}_n + e^{\alpha_m/2} \sum_{n \in \mathcal{F}_m} w^{(m)}_n \]

\[ = \left( e^{\alpha_m/2} \right)^{\sum_{n=1}^{N} w^{(m)}_n} I(h_m(x_n) \neq t_n) \]

---

B. Leibe
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

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

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

B. Leibe
AdaBoost - Minimizing Exponential Error

• Minimize with respect to \( h_m(x) \):
  \[
  \frac{\partial E}{\partial h_m(x_n)} = 0
  \]

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

  = const.

⇒ This is equivalent to minimizing

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

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

⇒ We’re on the right track. Let’s continue...
AdaBoost - Minimizing Exponential Error

- Minimize with respect to $\alpha_m$: \[ \frac{\partial E}{\partial \alpha_m} = 0 \]

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

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

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

\[ \epsilon_m = \frac{1}{e^{\alpha_m} + 1} \]

⇒ 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
    \[ E = \sum_{n=1}^{N} w^{(m)}_n \exp \left\{ -\frac{1}{2} t_n \alpha_m h_m(x_n) \right\} \]
    This becomes \( w^{(m+1)}_n \) in the next iteration.
  - Therefore
    \[ w^{(m+1)}_n = w^{(m)}_n \exp \left\{ -\frac{1}{2} t_n \alpha_m h_m(x_n) \right\} \]
    \[ = \ldots \]
    \[ = w^{(m)}_n \exp \{ \alpha_m I(h_m(x_n) \neq t_n) \} \]

⇒ Update for the weight coefficients.
AdaBoost - Final 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 \( h_m(x) \):
      \[
      \alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}
      \]
   
   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.
Comparing Error Functions

- Ideal misclassification error function (black)
  - This is what we want to approximate.
  - Unfortunately, it is not differentiable.
  ⇒ We cannot minimize it by gradient descent.
Comparing Error Functions

- Ideal misclassification error function
- “Hinge error” used in SVMs
  - Zero error for points outside the margin \((z>1)\).
  - Linearly increasing error for misclassified points \((z<1)\).

Image source: Bishop, 2006
Comparing Error Functions

- Ideal misclassification error function
- “Hinge error” used in SVMs
- Exponential error function
  - Continuous approximation to ideal misclassification function.
  - Sequential minimization leads to simple AdaBoost scheme.
  - Disadvantage: exponential penalty for large negative values!
    ⇒ Less robust to outliers or misclassified data points!

B. Leibe

Image source: Bishop, 2006
Comparing Error Functions

- Ideal misclassification error function
- “Hinge error” used in SVMs
- Exponential error function
- “Cross-entropy error” \( E = - \sum \{ t_n \ln y_n + (1 - t_n) \ln (1 - y_n) \} \)
  - Similar to exponential error for \( z > 0 \).
  - Only grows linearly with large negative values of \( z \).

⇒ Make AdaBoost more robust by switching ⇒ “GentleBoost”

B. Leibe

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
Topics of This Lecture

- Ensembles of Classifiers
- Constructing Ensembles
  - Cross-validation
  - Bagging
- Combining Classifiers
  - Stacking
  - Bayesian model averaging
  - Boosting
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions

- Applications
Example Application: Face Detection

- Frontal faces are a good example of a class where global appearance models + a sliding window detection approach fit well:
  - Regular 2D structure
  - Center of face almost shaped like a “patch”/window

- Now we’ll take AdaBoost and see how the Viola-Jones face detector works
Feature extraction

“Rectangular” filters

Feature output is difference between adjacent regions

Efficiently computable with integral image: any sum can be computed in constant time

Avoid scaling images \(\rightarrow\) scale features directly for same cost

\[ D = 1 + 4 - (2 + 3) = A + (A + B + C + D) - (A + C + A + B) = D \]

Slide credit: Kristen Grauman

[Viola & Jones, CVPR 2001]
Large Library of Filters

Considering all possible filter parameters: position, scale, and type:

180,000+ possible features associated with each 24 x 24 window

Use AdaBoost both to select the informative features and to form the classifier
AdaBoost for Feature+Classifier Selection

- Want to select the single rectangle feature and threshold that best separates positive (faces) and negative (non-faces) training examples, in terms of weighted error.

Resulting weak classifier:

\[ h_t(x) = \begin{cases} 
  +1 & \text{if } f_t(x) > \theta_t \\
  -1 & \text{otherwise} 
\end{cases} \]

For next round, reweight the examples according to errors, choose another filter/threshold combo.

Slide credit: Kristen Grauman
AdaBoost for Efficient Feature Selection

- Image features = weak classifiers
- For each round of boosting:
  - Evaluate each rectangle filter on each example
  - Sort examples by filter values
  - Select best threshold for each filter (min error)
    - Sorted list can be quickly scanned for the optimal threshold
  - Select best filter/threshold combination
  - Weight on this features is a simple function of error rate
  - Reweight examples

Viola-Jones Face Detector: Results
Viola-Jones Face Detector: Results
Viola-Jones Face Detector: Results
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: