Recap: SVM for Non-Separable Data
- Slack variables
  - One slack variable \( \zeta_i \geq 0 \) for each training data point.
- Interpretation
  - \( \zeta_i = 0 \) for points that are on the correct side of the margin.
  - \( \zeta_i > 0 \) for all other points.
  - We do not have to set the slack variables ourselves! They are jointly optimized together with \( w \).

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

Recap: The Kernel Trick
- Important observation
  - \( \phi(x) \) only appears in the form of dot products \( \phi(x)^T \phi(y) \):
    \[
    y(x) = w^T \phi(x) + b = \sum_{n=1}^{N} a_n f_n \phi(x_n)^T \phi(x) + b
    \]
  - 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:
    \[
    y(x) = \sum_{n=1}^{N} a_n l_n k(x_n, x) + b
    \]
  - The kernel function implicitly maps the data to the higher-dimensional space (without having to compute \( \phi(x) \) explicitly)!
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)

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/)

- **Limitations**
  - How to select the right kernel?
  - How to select the kernel parameters?
  - 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 $\langle x, y \rangle$ 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...

- **Evaluating** $\langle x, y \rangle$ $= \sum_{i=1}^{N} a_i a_j y_i y_j k(x_i, x_j)$

- **SVM Dual: Maximize**
  $L_d(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j y_i y_j k(x_i, x_j)$

  under the conditions
  $0 \leq a_i \leq C$
  $\sum_{i=1}^{N} a_i = 0$

  - **Classify new data points using**

  $y(x) = \sum_{i=1}^{N} a_i y_i k(x_i, x) + b$

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

Example Application: OCR
- Handwritten digit recognition
  - 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 ($\sigma=0.3$, 291 support vectors)

Example Application: Pedestrian 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.

Example Application: Object Detection
- Example Application: Pedestrian Detection
  - N. Dalal, B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

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 ($\sigma=0.3$, 291 support vectors)
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...

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,…

Methods for obtaining a set of classifiers

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 |

Methods for combining different classifiers

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.

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.

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

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:
    \[
    p(X) = \prod_{n=1}^{N} p(x_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)
    \]

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

- Average error of individual models
  \[
  E_{AV} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{\epsilon} \left[ \epsilon_m(x)^2 \right]
  \]
- Average error of committee
  \[
  E_{COM} = \mathbb{E}_{\epsilon} \left[ \frac{1}{M} \sum_{m=1}^{M} \epsilon_m(x)^2 \right] = \mathbb{E}_{\epsilon} \left[ \frac{1}{M} \sum_{m=1}^{M} \epsilon_m(x)^2 \right]
  \]
- Assumptions
  - Errors have zero mean: \( \mathbb{E}_{\epsilon} \left[ \epsilon_m(x) \right] = 0 \)
  - Errors are uncorrelated: \( \mathbb{E}_{\epsilon} \left[ \epsilon_m(x)\epsilon_j(x) \right] = 0 \)
- Then:
  \[
  E_{COM} = \frac{1}{M} E_{AV}
  \]
Model Averaging: Expected Error

- Average error of committee
  \[ E_{COM} = \frac{1}{3} E_{AV} \]
  - This suggests that the average error of a model can be reduced by a factor of 3 simply by averaging 3 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 (Schapire 1989)

- Algorithm: (3-component classifier)
  1. Sample \( N_1 < N \) training examples (without replacement) from training set \( D \) to get set \( D_1 \).
  2. Sample \( N_2 < N \) training examples (without replacement), half of which were misclassified by \( C_1 \) to get set \( D_2 \).
     - Train weak classifier \( C_1 \) on \( D_2 \).
  3. Choose all data in \( D \) on which \( C_1 \) and \( C_2 \) disagree to get set \( D_3 \).
     - Train weak classifier \( C_1 \) on \( D_3 \).
  4. Get the final classifier output by majority voting of \( C_1 \), \( C_2 \), and \( C_3 \).

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.
  - **Sequential classifier selection**

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.

Applying Boosting

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

Topics of This Lecture

- **Ensembles of Classifiers**
  - Construction Ensembles
    - 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
  - \( D(x) \): “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 - Formalization

- **2-class classification problem**
  - Given: training set \( X = \{x_1, \ldots, x_N\} \) with target values \( T = \{t_1, \ldots, t_N\}, t_i \in \{-1,1\} \).
  - 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 \( w_i \) if \( x_i \) was misclassified by \( h_m(x) \).
    - Decrease \( w_i \) if \( x_i \) 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 \( w_1^{(1)} = \frac{1}{N} \) 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
      \[
      J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n) \quad I(a) = \begin{cases} 1 & \text{if } a \text{ is true} \\ 0 & \text{otherwise} \end{cases}
      \]
   b) Estimate the weighted error of this classifier on \( X \):
      \[
      \epsilon_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n) \quad \text{such that} \quad \epsilon_m \leq 1/2
      \]
   c) Calculate a weighting coefficient for \( h_m(x) \):
      \[
      \alpha_m = \frac{1}{2} \log \left( \frac{1-\epsilon_m}{\epsilon_m} \right)
      \]
   d) Update the weighting coefficients:
      \[
      w_n^{(m+1)} = w_n^{(m)} \exp \left( -\alpha_m I(h_m(x) \neq t_n) \right)
      \]

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 retrotitting 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 \( h_l(x) \):
    \[ f_m(x) = \frac{1}{m} \sum_{l=1}^{m} \alpha_l h_l(x) \]
  - Goal
    - Minimize \( E \) with respect to both the weighting coefficients \( \alpha_l \) and the parameters of the base classifiers \( h_l(x) \).

- Sequential Minimization
  - Suppose that the base classifiers \( h_1(x), \ldots, h_m(x) \) and their coefficients \( \alpha_1, \ldots, \alpha_m \) are fixed.
    - Only minimize with respect to \( \alpha_n \) and \( h_n(x) \).
    \[
    E = \sum_{n=1}^{N} \exp \left( -t_n f_m(x_n) \right) \quad \text{with} \quad f_m(x) = \frac{1}{m} \sum_{l=1}^{m} \alpha_l h_l(x) \]
    \[ = \sum_{n=1}^{N} \exp \left( -t_n f_m(x_n) - \frac{1}{m} \sum_{l=1}^{m} \alpha_l h_l(x_n) \right) \quad \Rightarrow \text{const.} \]
    \[ = \sum_{n=1}^{N} w_n^{(m)} \exp \left( -\frac{1}{2} \sum_{l=1}^{m} \alpha_l h_l(x_n) \right) \]

- Rewriting the error
  \[
  E = \sum_{n=1}^{N} w_n^{(m)} \exp \left( -\frac{1}{2} \sum_{l=1}^{m} \alpha_l h_l(x_n) \right) \]
  \[
  = \exp \left( -\frac{1}{2} \sum_{m=1}^{M} \sum_{n=1}^{N} w_n^{(m)} h_m(x_n) \right) \]
  \[
  = \exp \left( -\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} w_n^{(m)} \right) \]
  \[
  = \exp \left( -\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} w_n^{(m)} I(h_m(x_n) \neq t_n) \right) \]
  \[
  = \exp \left( -\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} w_n^{(m)} I(h_m(x_n) \neq t_n) \right) \]

- Minimize with respect to \( h_m(x) \):
  \[
  \frac{\partial E}{\partial h_m(x_n)} = 0 \]
  \[
  E = \left( e^{-\alpha_m/2} - e^{-\alpha_m/2} \right) \sum_{m=1}^{M} \sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n) + e^{-\alpha_m/2} \sum_{m=1}^{M} \sum_{n=1}^{N} w_n^{(m)} \]
  \[
  = \text{const.} \]
  \[
  \Rightarrow \text{This is equivalent to minimizing} \]
  \[
  J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_m) \]
  \[
  \text{(our weighted error function from step 2a) of the algorithm} \]
  \[
  \Rightarrow \text{We're on the right track. Let's continue...} \]
**AdaBoost - Minimizing Exponential Error**

- Minimize with respect to $\alpha_n$: $\frac{\partial E}{\partial \alpha_n} = 0$

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

weighted error $\epsilon_n = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_n(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$

$\Rightarrow$ Update for the $\alpha$ coefficients: $\alpha_n = \ln \left(\frac{1}{\epsilon_n}\right)$

**AdaBoost - Final Algorithm**

1. Initialization: Set $w_n^{(1)} = \frac{1}{N}$ for $n = 1,...,N$.
2. For $m = 1,...,M$ iterations
   a) Train a new weak classifier $h_m(x)$ using the current weighting coefficients $W^{(m)}$ by minimizing the weighted error function $J_m = \sum w_n^{(m)} I(h_m(x_n) \neq t_n)$
   b) Estimate the weighted error of this classifier on $X$: $\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$
   c) Calculate a weighting coefficient for $h_m(x)$: $\alpha_m = \ln \left(\frac{1}{\epsilon_m}\right)$
   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.

- "Hinge error" used in SVMs
  - Zero error for points outside the margin ($z>1$).
  - Linearly increasing error for misclassified points ($z<1$).
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!
  ⇒ More robust to outliers or misclassified data points!

Exp N = -∑\{tn \ln y_n + (1 - t_n) \ln(1 - y_n)\}

Comparison error function

- Ideal misclassification error function
- "Hinge error" used in SVMs
- Exponential error function
- "Cross-entropy error" E = -∑\{tn \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"

Summary: AdaBoost

- Properties
  - Simple combination of multiple classifiers.
  - Easy to implement.
  - Can be used with many different types of classifiers.
  - 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

Value at (x,y) is sum of pixels above and to the left of (x,y)
Efficiently computable with integral image: any sum can be computed in constant time
Avoid scaling images ⇒ scale features directly for same cost
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:
\[ h_i(x) = \begin{cases} +1 & \text{if } f_i(x) > 0, \\ -1 & \text{otherwise} \end{cases} \]

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

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, CVPR 2001

B. Leibe
Machine Learning, Summer'11
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: