Machine Learning - Lecture 10
Model Combination & Boosting
06.06.2016

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

Recap: SVM for Non-Separable Data

- Slack variables
  - One slack variable $\xi_i \geq 0$ for each training data point.
- Interpretation
  - $\xi_i = 0$ for points that are on the correct side of the margin.
  - $\xi_i = y_i - y(x_i)$ 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
  $$L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_n^T x_m)$$
  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
  $\Rightarrow$ Solve as before...

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 $\phi(x)^T \phi(y)$:
  $$y(x) = w^T \phi(x) + b = \sum_{n=1}^{N} a_n t_n \phi(x_n)^T \phi(x) + b$$
  - 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
  \[ L(y(a)) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \]
  under the conditions
  \[ \sum_{n=1}^{N} a_n t_n = 0 \]
  \[ \sum_{n=1}^{N} a_n = C \]
- Classify new data points using
  \[ y(x) = \sum_{n=1}^{N} a_n t_n k(x_n, x) + b \]

Recap: Error Functions

- \( t_n \in \{ -1, 1 \} \) Ideal misclassification error

  \[ E(t_n) = t_n y(x_n) \]

  - Not differentiable
  - 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.

Error Functions (Loss Functions)

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

SVM - Analysis

- Traditional soft-margin formulation
  \[ \min_{w \in \mathbb{R}^n, \xi \in \mathbb{R}^+} \frac{1}{2} ||w||^2 + C \sum_{n=1}^{N} \xi_n \]
  subject to the constraints
  \[ \sum_{n=1}^{N} a_n = 0 \]
  \[ \sum_{n=1}^{N} a_n t_n = 0 \]
  \[ t_n y(x_n) \geq 1 - \xi_n \]
- Different way of looking at it
  - We can reformulate the constraints into the objective function.
    \[ \min_{w \in \mathbb{R}^n} \frac{1}{2} ||w||^2 + C \sum_{n=1}^{N} (1 - t_n y(x_n))_+ \]
  where \( (x)_+ := \max(0, x) \).

Recap: Error Functions

- 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.

SVM - Discussion

- SVM optimization function
  \[ \min_{w \in \mathbb{R}^n} \frac{1}{2} ||w||^2 + C \sum_{n=1}^{N} (1 - t_n y(x_n))_+ \]
  \[ L_2 \text{ regularizer} \]
  \[ \text{Hinge loss} \]
  \[ \text{"Hinge loss enforces sparsity"} \]
  - 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
Applications of SVMs: 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:**

| Method         | USPS benchmark error | Decision tree (C4.5) error | 2-layer Neural Network error | LeNet 1 error
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9%</td>
<td>16.2%</td>
<td>5.1%</td>
<td>4.0%</td>
<td></td>
</tr>
</tbody>
</table>

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

Example Application: OCR

- **Results**

<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>≈ 3000</td>
<td>227</td>
<td>4.7</td>
</tr>
<tr>
<td>3</td>
<td>≈ 1 × 10^6</td>
<td>247</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>≈ 1 × 10^9</td>
<td>321</td>
<td>4.2</td>
</tr>
<tr>
<td>5</td>
<td>≈ 1 × 10^12</td>
<td>374</td>
<td>4.3</td>
</tr>
<tr>
<td>6</td>
<td>≈ 1 × 10^14</td>
<td>377</td>
<td>4.5</td>
</tr>
<tr>
<td>7</td>
<td>≈ 1 × 10^16</td>
<td>422</td>
<td>4.5</td>
</tr>
</tbody>
</table>

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: 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.

**Example Application: Pedestrian Detection**

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

---

**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...

---

**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
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.

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

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 Naive 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.
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 i.i.d. 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.
    \( \Rightarrow \) One latent variable \( x_i \) for each data point:
    \[ p(X) = \prod_{n=1}^{N} p(x_n) = \prod_{n=1}^{N} \sum_{m=1}^{M} p(x_n | h_m) \]

- Bayesian Model Averaging
  - The whole data set is generated by a single model.
  - Uncertainty is about which model was responsible.
    \( \Rightarrow \) One latent variable \( x \) for the entire data set:
    \[ p(X) = \sum_{x} p(X, x) \]

Model Averaging: Expected Error

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

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

- Assumptions
  - Errors have zero mean: \( E_x [\epsilon_m(x)] = 0 \)
  - Errors are uncorrelated: \( E_x [\epsilon_m(x)\epsilon_f(x)] = 0 \)

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

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
    \[ E_{COM} \neq E_{AV} \]
**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
  - 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:
    \[
    H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right)
    \]

**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.

**Final classifier is 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 \( w_1^{(1)} = \frac{1}{N} \) 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
      \[ J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n) \]
      \( I \) is \( 1, \) if \( m \) is true \( \) and \( 0, \) else.
   b) Estimate the weighted error of this classifier on \( X \):
      \[ e_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n) \]
   c) Calculate a weighting coefficient for \( h_m(x) \):
      \[ \alpha_m = \frac{e_m}{e_m + e_{m+1}} \]
   d) Update the weighting coefficients:
      \[ w_{n+1}^{(m+1)} = \frac{w_n^{(m)}}{\left( e_m + e_{m+1} \right)} \]

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

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

- Observation:
  - Correctly classified points: \( I(h_m(x_n) = +1) \Rightarrow \text{collect in } \mathcal{T}_m \)
  - Misclassified points: \( I(h_m(x_n) = -1) \Rightarrow \text{collect in } \mathcal{F}_m \)
- Rewrite the error function as
  \[ E = e^{-\alpha_m/2} \sum_{n \in \mathcal{T}_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in \mathcal{F}_m} w_n^{(m)} \]
  \[ = \left( e^{\alpha_m/2} \right) \sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n) \]
Recall that \( I + I = 1 \):

- Update the weighting coefficients:
  
  \[
  w^{(m+1)}_n = w^{(m)}_n \exp \left\{ -\frac{1}{2} \alpha_m h_m(x_n) \right\}
  \]

This allows us to analyze AdaBoost’s behavior in more detail.

- Estimate the weighted error of this classifier on \( X \):
  
  \[
  \epsilon_m = \frac{\sum_{n=1}^{N} w^{(m)}_n I(h_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w^{(m)}_n} = e^{-\epsilon_m/2}
  \]

- Update for the \( \alpha \) coefficients:
  
  \[
  \alpha_m = \ln \left( 1 - \frac{\epsilon_m}{\epsilon_m} \right)
  \]

**AdaBoost - Minimizing Exponential Error**

- Minimize with respect to \( h_m(x) \):
  
  \[
  E = \frac{1}{2} \frac{\partial E}{\partial h_m(x_n)} = 0
  \]

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

  \[
  = \frac{1}{2} e^{-\epsilon_m/2} + \frac{1}{2} e^{\epsilon_m/2} = \text{const.}
  \]

  \[
  \Rightarrow \frac{1}{2} e^{-\epsilon_m/2} + \frac{1}{2} e^{\epsilon_m/2} = \text{const.}
  \]

  \[
  \Rightarrow \text{This is equivalent to minimizing}
  \]

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

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

  \[
  \Rightarrow \text{We’re on the right track. Let’s continue…}
  \]

**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

- **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.

**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.

**“Hinge error” used in SVMs**

- Zero error for points outside the margin \( z > 1 \) ⇒ sparsity
- Linear penalty for misclassified points \( z < 1 \) ⇒ robustness
- Not differentiable around \( z = 1 \) ⇒ Cannot be optimized directly.

**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!

**“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”
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
    - 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

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:

\[ h(x) = \begin{cases} 
  +1 & \text{if } f(x) > \theta_i \\
  -1 & \text{otherwise} 
\end{cases} \]

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

(first version appeared at CVPR 2001)

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: