Recap: 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.
  - Interpret the prediction of the $L$ classifiers as $L$-dimensional feature vector.
  - Learn "level-2" classifier based on the examples generated this way.

Recap: Bayesian Model Averaging

- **Model Averaging**
  - Suppose we have $H$ different models $h = 1, ..., H$ with prior probabilities $p(h)$.
  - Construct the marginal distribution over the data set
    $$ p(X) = \sum_{h=1}^{H} p(X|h) p(h) $$

- **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!
  - Unfortunately, this assumes that the errors are all uncorrelated. In practice, they will typically be highly correlated.

Recap: Boosting (Schapire 1989)

- **Algorithm:** (3-component classifier)
  1. Sample $N < N$ training examples (without replacement) from training set $D$ to get set $D_1$.
     - Train weak classifier $C_1$ on $D_1$.
  2. Sample $N < N$ training examples (without replacement), half of which were misclassified by $C_1$, to get set $D_2$.
     - Train weak classifier $C_2$ 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 $C_3$ on $D_3$.
  4. Get the final classifier output by majority voting of $C_1$, $C_2$, and $C_3$.
     (Recursively apply the procedure on $C_i$ to $C_i$.)

Topics of This Lecture

- **AdaBoost**
  - Intuition
  - Algorithm
  - Analysis
  - Extensions

- **Comparing Error Functions**
- **Applications**
  - AdaBoost for face detection
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: Formalization

- 2-class classification problem
  - Given: training set $X = \{x_1, ..., x_N\}$ with target values $T = \{t_1, ..., t_N\}$, $t_j \in \{-1, 1\}$.
  - Associated weights $W = \{w_1, ..., w_N\}$ 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: 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 - Algorithm

1. Initialization: Set $w^{(1)}_n = \frac{1}{N}$ 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
      $$J_m = \sum_{n=1}^{N} w^{(m)}_n I(h_m(x) \neq t_n)$$
      $$\hat{I}(A) = \begin{cases} 0 & \text{if } A \text{ is true} \\ 1 & \text{else} \end{cases}$$
   b) Estimate the weighted error of this classifier on $X$:
      $$\epsilon_m = \frac{\sum_{n=1}^{N} w^{(m)}_n I(h_m(x) \neq t_n)}{\sum_{n=1}^{N} w^{(m)}_n}$$
   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:
      $$w^{(m+1)}_n = \frac{w^{(m)}_n}{Z_m}$$
      $$Z_m = \sum_{n=1}^{N} \frac{w^{(m)}_n}{Z_m}$$
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 \( h_i(x) \):
    \[ f_m(x) = \frac{1}{m} \sum_{i=1}^{m} \alpha_i h_i(x) \]

- Goal
  - Minimize \( E \) with respect to both the weighting coefficients \( \alpha_i \)
    and the parameters of the base classifiers \( h_i(x) \).

### Details

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

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

**Rewrite the error function as**
\[ E \ni \left( e^{\alpha_m} - e^{-\alpha_m} \right) \sum_{n \in T_m} w_n^{(m)} I(h_m(x_n) \neq t_n) + \left( e^{\alpha_m} - e^{-\alpha_m} \right) \sum_{n \in F_m} w_n^{(m)} I(h_m(x_n) = t_n) \]

**Goal**
- Minimize with respect to \( h_m(x) \): 
  \[ \frac{\partial E}{\partial h_m(x)} \]
  \[ E \ni \left( e^{\alpha_m} - e^{-\alpha_m} \right) \sum_{n \in T_m} w_n^{(m)} I(h_m(x_n) \neq t_n) + \left( e^{\alpha_m} - e^{-\alpha_m} \right) \sum_{n \in F_m} w_n^{(m)} I(h_m(x_n) = t_n) \ni \text{const.} \]

\[ J_m \ni \sum_{n \in T_m} w_n^{(m)} I(h_m(x_n) \neq t_n) \ni \text{const.} \]

This is equivalent to minimizing
\[ J_m = \sum_{n \in T_m} w_n^{(m)} I(h_m(x_n) \neq t_n) \ni \text{the weighted error function from step 2a of the algorithm} \]

\[ \Rightarrow \text{We're on the right track. Let's continue...} \]
AdaBoost - Minimizing Exponential Error

- Minimize with respect to \( \alpha_m \): \( \frac{\partial E}{\partial \alpha_m} = 0 \)

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

- Remaining step: update the weights
  Recall that:

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

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

\[
w^{(m+1)} = w^{(m)} \exp \left\{ -\frac{1}{2} \alpha_m h_m(x_n) \right\}
\]

- Update for the weight coefficients:

\[
\alpha_m = \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right)
\]

AdaBoost - Final Algorithm

1. Initialization: Set \( w^{(1)}(n) = \frac{1}{N} \) 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
   \[
   J_m = \sum_{n=1}^{N} w^{(m)}(n) I(h_m(x_n) \neq t_n)
   \]
   b) Estimate the weighted error of this classifier on \( X \):
   \[
   \epsilon_m = \frac{\sum_{n=1}^{N} w^{(m)}(n) I(h_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w^{(m)}(n)}
   \]
   c) Calculate a weighting coefficient for \( h_m(x) \):
   \[
   \alpha_m = \ln \left( \frac{1}{\epsilon_m} \right)
   \]
   d) Update the weighting coefficients:
   \[
   w^{(m+1)}(n) = w^{(m)}(n) \exp \{ \alpha_m I(h_m(x_n) \neq t_n) \}
   \]

AdaBoost - Analysis

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

Comparing Error Functions (Loss Functions)

- Ideal classification 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.

- Not differentiable!
Comparing Error Functions (Loss Functions)

- Squared error used in Least-Squares Classification
  - Very popular, leads to closed-form solutions.
  - However, sensitive to outliers due to squared penalty.
  - Penalties “too correct” data points
  ⇒ Generally does not lead to good classifiers.

- Hinge error used in SVMs
  - Zero error for points outside the margin ($z>1$).
  - Linearly increasing error for misclassified points ($z<1$).
  ⇒ Leads to sparse solutions, not sensitive to outliers.
  - Not differentiable around $z=1$ ⇒ Cannot be optimized directly.

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

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

- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions
- Comparing Error Functions
- Applications
  - AdaBoost for face detection

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

Value at \((x,y)\) is sum of pixels above and to the left of \((x,y)\)

Integral Image

\[
I(x,y) = \sum_{(a,b) \leq (x,y)} I(a,b)
\]

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:

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

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

AdaBoost for Efficient Feature Selection

- Image features \Rightarrow 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

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: