Machine Learning

10-701/15-781, Fall 2008

Ensemble methods
Boosting from Weak Learners

Lecture 9, October 6, 2008

Reading: Chap. 14.3 C.B book

Project proposal due this Wed

Mid term

> 10 \( \sim 10 \)

> 15 \( \sim 10 \)
The essence of kernel

- Feature mapping, but “without paying a cost”
  - E.g., polynomial kernel
    \[ K(x, z) = \left( x^T z + c \right)^d \]
    \[ O(n^d) \]
  - How many dimensions we’ve got in the new space?
  - How many operations it takes to compute \( K() \)?

- Kernel design, any principle?
  - \( K(x,z) \) can be thought of as a similarity function between \( x \) and \( z \)
  - This intuition can be well reflected in the following “Gaussian” function
    (Similarly one can easily come up with other \( K() \) in the same spirit)
    \[ K(x, z) = \exp \left( - \frac{\|x - z\|^2}{2\sigma^2} \right) \]
  - Is this necessarily lead to a “legal” kernel?
    (in the above particular case, \( K() \) is a legal one, do you know how many dimension \( \phi(x) \) is?)

(3) Structured Prediction

- Unstructured prediction

- Structured prediction
  - Part of speech tagging
    \[ x = \text{“Do you want sugar in it?”} \quad \Rightarrow \quad y = \langle \text{verb pron verb noun prep pron} \rangle \]
  - Image segmentation

\[ x = \begin{pmatrix} x_{11} & x_{12} & \cdots \\ x_{21} & x_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \quad y = \begin{pmatrix} y_{11} & y_{12} & \cdots \\ y_{21} & y_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \]
**OCR example**

Sequential structure: $f(x; x_{i+1})$ by $y$

$F(x, y) \downarrow F(x, f(y))$

**Inputs:**
- A set of training samples $D = \{(x_i, y_i)\}_{i=1}^N$, where $x_i = [x_1^i, x_2^i, \ldots, x_d^i]$ and $y_i \in C \subseteq \{e_1, e_2, \ldots, e_L\}$

**Outputs:**
- A predictive function $h(x)$: $y^* = h(x) \triangleq \arg\max_y F(x, y)$
  
  $F(x, y) = w^T f(x, y)$

**Examples:**
- SVM: $\max_w \frac{1}{2} w^T w + C \sum_{i=1}^N \xi_i$ s.t. $w^T \Delta f_i(y) \geq 1 - \xi_i$, $\forall i, \forall y_i$

- Logistic Regression: $\max_w \mathcal{L}(D; w) \triangleq \sum_{i=1}^N \log p(y_i|x_i)$

  where $p(y|x) = \frac{\exp{w^T f(x, y)}}{\sum_{y'} \exp{w^T f(x, y')}}$

**Classical Classification Models**

© Eric Xing @ CMU, 2006-2008
Structured Models

\[ h(x) = \arg \max_{y \in \mathcal{Y}(x)} F(x, y) \]

- **Assumptions:**
  - Linear combination of features
  - Sum of partial scores: index \( p \) represents a part in the structure
  - Random fields or Markov network features:

![Diagram showing a structured model with nodes and edges representing features and their interactions.]

Discriminative Learning Strategies

- **Max Conditional Likelihood**
  - We predict based on:
    \[ y^* | x = \arg \max_y p_y (y | x) = \frac{1}{Z(w, x)} \exp \left\{ \sum_w w_c f_c(x, y) \right\} \]
  - And we learn based on:
    \[ w^* | y, x = \arg \max_w \prod_i p_w (y_i | x_i) = \prod_i \frac{1}{Z(w, x_i)} \exp \left\{ \sum_w w_c f_c(x, y) \right\} \]

- **Max Margin**
  - We predict based on:
    \[ y^* | x = \arg \max_y \sum_w w f_c(x, y) = \arg \max_w \arg \max f(x, y) \]
  - And we learn based on:
    \[ w^* | y, x = \arg \max_w \min \left\{ w \right\} \min \left\{ f(y_i, x_i) - f(y_i, x) \right\} \]
E.g. Max-Margin Markov Networks

- Convex Optimization Problem:

\[
P_0 \text{(M^3N)} : \min_{w, \xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} \xi_i
\]

s.t. \forall i, \forall y \neq y_i : \quad w^T \Delta f_i(x, y) \geq \Delta \ell_i(y) - \xi_i, \quad \xi_i \geq 0,

- Feasible subspace of weights:

\[
F_0 = \{ w : w^T \Delta f_i(x, y) \geq \Delta \ell_i(y) - \xi_i ; \forall i, \forall y \neq y_i \}
\]

- Predictive Function:

\[
h_0(x; w) = \arg \max_{y \in \mathcal{Y}(x)} \mathcal{F}(x, y; w)
\]

OCR Example

- We want:

\[
\arg \max_{\text{word}} w^T f(\text{brace}, \text{word}) = \text{“brace”}
\]

- Equivalently:

\[
\begin{align*}
w^T f(\text{brace}, \text{“brace”}) &> w^T f(\text{brace}, \text{“aaaa”}) \\
w^T f(\text{brace}, \text{“brace”}) &> w^T f(\text{brace}, \text{“aaaab”}) \\
\vdots \\
w^T f(\text{brace}, \text{“brace”}) &> w^T f(\text{brace}, \text{“zzzz”})
\end{align*}
\]

\[\text{a lot!}\]
Min-max Formulation

- Brute force enumeration of constraints:
  \[
  \min \frac{1}{2}||w||^2 \\
  w^T f(x, y^*) \geq w^T f(x, y) + \ell(y^*, y), \quad \forall y
  \]
  - The constraints are exponential in the size of the structure

- Alternative: min-max formulation
  - add only the most violated constraint
  \[
  y' = \arg \max_{y \neq y^*} \{ w^T f(x_i, y) + \ell(y_i, y) \}
  \]
  - adds to QP:
    \[
    w^T f(x_i, y_i) \geq w^T f(x_i, y') + \ell(y_i, y')
    \]
  - Handles more general loss functions
  - Only polynomial # of constraints needed
  - Several algorithms exist ...

Results: Handwriting Recognition

Length: ~8 chars
Letter: 16x8 pixels
10-fold Train/Test
5000/50000 letters
600/6000 words

Models:
- Multiclass-SVMs*
- M^3 nets

* Crammer & Singer 01

Test error (average per-character)
raw pixels  quadratic kernel  cubic kernel
raw pixels  quadratic kernel  cubic kernel

better

33% error reduction over multiclass SVMs
Discriminative Learning Paradigms

SVM
\[ y = \text{sign}(w^T x + b) \]
\[
\min_{w, b} \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} \xi_i \\
y'(w^T x_i + b) \geq 1 - \xi_i, \quad \forall i
\]

MED
\[ y = \text{sign}(f(x, w)) \]
\[
\min_{w} \text{KL}(Q||Q_w) \\
y'(f(x')) \geq \xi_i, \quad \forall i
\]

M3N
\[ y = \arg \max_{y \in \mathbb{Y}} P(y | x, w) \]
\[
\min_{w} \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} \xi_i \\
f(x', y) \geq f(x', y') - \xi_i, \quad \forall i, y \neq y'
\]

MED-MN = SMED + Bayesian M3N

See [Zhu and Xing, 2008]

Summary

- Maximum margin nonlinear separator
  - Kernel trick
  - Project into linearly separable space (possibly high or infinite dimensional)
  - No need to know the explicit projection function

- Max-entropy discrimination
  - Average rule for prediction,
  - Average taken over a posterior distribution of \( w \) who defines the separation hyperplane
  - \( P(w) \) is obtained by max-entropy or min-KL principle, subject to expected marginal constraints on the training examples

- Max-margin Markov network
  - Multi-variate, rather than uni-variate output \( Y \)
  - Variable in the outputs are not independent of each other (structured input/output)
  - Margin constraint over every possible configuration of \( Y \) (exponentially many!)
Rationale: Combination of methods

- There is no algorithm that is always the most accurate
- We can select simple “weak” classification or regression methods and combine them into a single “strong” method
- Different learners use different
  - Algorithms
  - Parameters
  - Representations (Modalities)
  - Training sets
  - Subproblems
- The problem: how to combine them
Some early algorithms

- Boosting by filtering (Schapire 1990)
  - Run weak learner on differently filtered example sets
  - Combine weak hypotheses
  - Requires knowledge on the performance of weak learner
- Boosting by majority (Freund 1995)
  - Run weak learner on weighted example set
  - Combine weak hypotheses linearly
  - Requires knowledge on the performance of weak learner
- Bagging (Breiman 1996)
  - Run weak learner on bootstrap replicates of the training set
  - Average weak hypotheses
  - Reduces variance

Combination of classifiers

- Suppose we have a family of component classifiers (generating ±1 labels) such as decision stumps:

\[ h(x; \theta) = \text{sign}(w^T x_k + b) \]

where \( \theta = \{k,w,b\} \)

- Each decision stump pays attention to only a single component of the input vector
Combination of classifiers con’d

- We’d like to combine the simple classifiers additively so that the final classifier is the sign of

\[ \hat{h}(x) = \alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m) \]

where the “votes” \( \{\alpha_i\} \) emphasize component classifiers that make more reliable predictions than others.

- Important issues:
  - what is the criterion that we are optimizing? (measure of loss)
  - we would like to estimate each new component classifier in the same manner (modularity)

AdaBoost

- **Input:**
  - \( N \) examples \( S_N = \{(x_1, y_1), \ldots, (x_N, y_N)\} \)
  - a weak base learner \( h = h(x, \theta) \)

- **Initialize:** equal example weights \( w_i = 1/N \) for all \( i = 1..N \)

- **Iterate for \( t = 1..T \):**
  1. train base learner according to weighted example set \( (w_i, x) \) and obtain hypothesis \( h_t = h(x, \theta) \)
  2. compute hypothesis error \( \epsilon_t \)
  3. compute hypothesis weight \( \alpha_t \)
  4. update example weights for next iteration \( w_{i+1} \)

- **Output:** final hypothesis as a linear combination of \( h_t \)
AdaBoost

- At the $k$th iteration we find (any) classifier $h(x; \theta_k^*)$ for which the weighted classification error:

$$\epsilon_k = \frac{1}{n} \sum_{i=1}^{n} W_i^{k-1} I(y_i \neq h(x_i; \theta_k^*) / \sum_{i=1}^{n} W_i^{k-1}$$

is better than change.
- This is meant to be "easy" --- weak classifier
- Determine how many “votes” to assign to the new component classifier:

$$\alpha_k = 0.5 \log((1 - \epsilon_k) / \epsilon_k)$$
- stronger classifier gets more votes
- Update the weights on the training examples:

$$W_i^k = W_i^{k-1} \exp\{-y_i \alpha_k h(x_i; \theta_k)\}$$

AdaBoost: dataflow diagram

$$f_T(x) = \sum_{i=1}^{T} \frac{\alpha_i}{\sum_{i=1}^{T} \alpha_i} h_i(x)$$
What is the criterion that we are optimizing? (measure of loss)
Measurement of error

- Loss function:
  \[ \lambda(y, h(x)) \quad \text{(e.g. } I(y \neq h(x)) \text{)} \]

- Generalization error:
  \[ L(h) = E[\lambda(y, h(x))] \]

- Objective: find \( h \) with minimum generalization error

- Main boosting idea: minimize the empirical error:
  \[ \hat{L}(h) = \frac{1}{N} \sum_{i=1}^{N} \lambda(y_i, h(x_i)) \]

Exponential Loss

- Empirical loss:
  \[ \hat{L}(h) = \frac{1}{N} \sum_{i=1}^{N} \lambda(y_i, \hat{h}_m(x_i)) \]

- Another possible measure of empirical loss is
  \[ \hat{L}(h) = \sum_{i=1}^{N} \exp\left(-y_i \hat{h}_m(x_i)\right) \]
Exponential Loss

- One possible measure of empirical loss is

\[
\hat{L}(h) = \sum_{i=1}^{n} \exp\left\{ -y_i h_m(x_i) \right\}
\]

Recall that:
\[
\hat{h}_m(x) = \alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m)
\]

\[
= \sum_{i=1}^{n} \exp\left\{ -y_i \hat{h}_{m-1}(x_i) - y_i a_m h(x; \theta_m) \right\}
\]

\[
= \sum_{i=1}^{n} \exp\left\{ -y_i \hat{h}_{m-1}(x_i) \right\} \exp\left\{ -y_i a_m h(x; \theta_m) \right\}
\]

\[
= \sum_{i=1}^{n} W_{m-1} \exp\left\{ -y_i a_m h(x; \theta_m) \right\}
\]

- The combined classifier based on \( m - 1 \) iterations defines a weighted loss criterion for the next simple classifier to add.
- Each training sample is weighted by its "classifiability" (or difficulty) seen by the classifier we have built so far.
Linearization of loss function

- We can simplify a bit the estimation criterion for the new component classifiers (assuming $\alpha$ is small)

$$\exp\{-y, a_{m}h(x; \theta_{m})\} \approx 1 - y, a_{m}h(x; \theta_{m})$$

- Now our empirical loss criterion reduces to

$$\sum_{i=1}^{n} \exp\{-y_{i}, \hat{h}_{m}(x_{i})\}$$

$$\approx \sum_{i=1}^{n} W_{m-1}^{-1}(1 - y_{i}, a_{m}h(x; \theta_{m}))$$

$$\approx \sum_{i=1}^{n} W_{m-1}^{-1} - a_{m} \sum_{i=1}^{n} W_{m-1}^{-1} y_{i} h(x; \theta_{m})$$

- We could choose a new component classifier to optimize this weighted agreement

A possible algorithm

- At stage $m$ we find $\theta^{*}$ that maximize (or at least give a sufficiently high) weighted agreement:

$$\sum_{i=1}^{n} W_{m-1}^{-1} y_{i} h(x; \theta_{m}^{*})$$

  - each sample is weighted by its "difficulty" under the previously combined $m - 1$ classifiers,
  - more "difficult" samples received heavier attention as they dominates the total loss

- Then we go back and find the "votes" $\alpha_{m}^{*}$ associated with the new classifier by minimizing the (exponential) loss

$$\hat{L}(h) = \sum_{i=1}^{n} W_{m-1}^{-1} \exp\{-y_{i}, a_{m}h(x; \theta_{m})\}$$

$$= \sum_{i_{1} \in F} W_{m-1}^{-1} \exp\{-\omega_{i_{1}}\} + \sum_{i_{2} \in T} W_{m-1}^{-1} \exp\{-\omega_{i_{2}}\}$$
The AdaBoost algorithm

- At the $k$th iteration we find (any) classifier $h(x; \theta_k^*)$ for which the weighted classification error:

$$
\varepsilon_k = \frac{\sum_{i=1}^{n} W_i^k I(y_i \neq h(x_i; \theta_k^*))}{\sum_{i=1}^{n} W_i^k}
$$

is better than change.
- This is meant to be "easy" --- weak classifier
- Determine how many "votes" to assign to the new component classifier:

$$
\alpha_k = 0.5 \log((1 - \varepsilon_k) / \varepsilon_k)
$$

- stronger classifier gets more votes
- Update the weights on the training examples:

$$
W_i^k = W_i^{k-1} \exp\{-y_i h(x_i; \theta_k^*)\}
$$
The AdaBoost algorithm cont’d

- The final classifier after m boosting iterations is given by the sign of

$$h(x) = \frac{\alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m)}{\alpha_1 + \ldots + \alpha_m}$$

- The votes here are normalized for convenience

Boosting

- We have basically derived a Boosting algorithm that sequentially adds new component classifiers, each trained on reweighted training examples
  - Each component classifier is presented with a slightly different problem

- AdaBoost preliminaries:
  - We work with normalized weights $W_i$ on the training examples, initially uniform ($W_i = 1/n$)
  - The weight reflects the "degree of difficulty" of each datum on the latest classifier
AdaBoost: summary

- **Input:**
  - $N$ examples $S_N = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  - a weak base learner $h = h(x, \theta)$
- **Initialize:** equal example weights $w_i = 1/N$ for all $i = 1..N$
- **Iterate for $t = 1...T$:**
  1. train base learner according to weighted example set $(w_i, x_i)$ and obtain hypothesis $h_t = h(x, \theta_t)$
  2. compute hypothesis error $\epsilon_t$
  3. compute hypothesis weight $\alpha_t$
  4. update example weights for next iteration $w_{t+1}$
- **Output:** final hypothesis as a linear combination of $h_t$

Base Learners

- **Weak learners used in practice:**
  - Decision stumps (axis parallel splits)
  - Decision trees (e.g. C4.5 by Quinlan 1996)
  - Multi-layer neural networks
  - Radial basis function networks

- **Can base learners operate on weighted examples?**
  - In many cases they can be modified to accept weights along with the examples
  - In general, we can sample the examples (with replacement) according to the distribution defined by the weights
Boosting performance

- The error rate of component classifier (the decision stumps) does not improve much (if at all) over time.
- But both training and testing error improve over time!
- Even after the training error of the combined classifier goes to zero, boosting iterations can still improve the generalization error!!

Why it is working?

- You will need some learning theory (to be covered in the next two lectures) to understand this fully, but for now let's just go over some high level ideas.
- Generalization Error:

  With high probability, Generalization error is less than:

  \[
  \hat{\Pr}[H(x) \neq y] + \tilde{O}\left(\sqrt{\frac{td}{m}}\right)
  \]

  As \(T\) goes up, our bound becomes worse, Boosting should overfit!
Experiments

The Boosting Approach to Machine Learning, by Robert E. Schapire

Training Margins

- When a vote is taken, the more predictors agreeing, the more confident you are in your prediction.

- Margin for example:

\[
\text{margin}_i(x_i, y_i) = y_i \left[ \frac{\sum \alpha_i h(x_i; \theta_i) + \ldots + \alpha_m h(x_i; \theta_m)}{\alpha_1 + \ldots + \alpha_m} \right]
\]

The margin lies in \([-1, 1]\) and is negative for all misclassified examples.

- Successive boosting iterations improve the majority vote or margin for the training examples.
More Experiments

The Boosting Approach to Machine Learning, by Robert E. Schapire

A Margin Bound

- For any $\gamma$, the generalization error is less than:

$$\Pr(margin_k(x,y) \leq \gamma) + O\left(\frac{d}{\sqrt{m\gamma^2}}\right)$$


- It does not depend on $T$!!!
Summary

- Boosting takes a weak learner and converts it to a strong one.
- Works by asymptotically minimizing the empirical error.
- Effectively maximizes the margin of the combined hypothesis.