Ensemble Learning

CS534
Ensemble Learning

Traditional:

S → L₁ → h₁

(x, ?) ⇔ (x, y* = h₁(x))

Ensemble method:

S → L₁ → h₁
S → L₂ → h₂
S → ... → hₛ

h* = F(h₁, h₂, ..., hₛ)

(x, ?) ⇔ (x, y* = h*(x))

different training sets and/or learning algorithms
How to generate ensembles?

• There have been a wide range of methods developed

• We will study some popular approaches
  – Bagging (and Random Forest, a variant that builds de-correlated trees)
  – Boosting

• Both methods take a single (base) learning algorithm and generate ensembles
Base Learning Algorithm

- We are given a ‘black box’ learning algorithm \textit{Learn} referred to as the base learner.

Protocol to \textit{Learn}:

\begin{center}
\textbf{Input:} \\
\textbf{\hspace{1cm}} S \quad - \quad \text{set of labeled training instances.} \\

\textbf{Output:} \\
\textbf{\hspace{1cm}} h \quad - \quad \text{a hypothesis from hypothesis space } H.
\end{center}
Bootstrap Aggregating (Bagging)

- Create many different training sets by sampling from the original training set and learn a hypothesis for each training set.
  - Resulting hypotheses will vary due to using different training sets
  - Combine these hypotheses using majority vote
Bagging Algorithm

Given training set $S$, bagging works as follows:

1. Create $T$ bootstrap samples $\{S_1, ..., S_T\}$ of $S$ as follows:
   - For each $S_i$: Randomly drawing $|S|$ examples from $S$ with replacement
2. For each $i = 1, ..., T$, $h_i = \text{Learn}(S_i)$
3. Output $H = \langle \{h_1, ..., h_T\}, \text{majorityVote} \rangle$

With large $|S|$, each $S_i$ will contain $1 - \frac{1}{e} \approx 63.2\%$ unique examples
Target concept

Single decision tree

100 bagged decision tree
Stability of Learn

• A learning algorithm is **unstable** if small changes in the training data can produce large changes in the output hypothesis (otherwise **stable**) – high variance

• Bagging will have little benefit when used with stable learning algorithms (i.e., most ensemble members will be very similar).

• Bagging generally works best when used with unstable yet relatively accurate base learners
  – High variance and low bias classifiers
Bagging Decision Trees
(Freund & Schapire)
Random Forest

• An extension to bagging
• Builds an ensemble of de-correlated decision trees
• One of the most successful classifiers in current practice
  – Very fast
  – Easy to train
  – Many good implements available
Random Forest Classifier

- Each bootstrapped sample is used to build a tree
- When building the tree, each node only choose from $m < M$ randomly sampled features
- Gini index is used to select the test samples
Random Forest Classifier

N examples $\rightarrow$ M features $\rightarrow$ Take majority vote
Random forest learns trees that makes de-correlated errors.

**FIGURE 15.9.** Correlations between pairs of trees drawn by a random-forest regression algorithm, as a function of $m$. The boxplots represent the correlations at 600 randomly chosen prediction points $x$. 
Random forest

• Available package:
  • http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

• To read more:
  • http://www-stat.stanford.edu/~hastie/Papers/ESLII.pdf
Boosting

• Its iterative.
  – Bagging: Individual classifiers were independently learned
  – Boosting:
    • Look at errors from previous classifiers to decide what to focus on for the next iteration over data
    • Successive classifiers depends upon its predecessors.
    • Result: more weights on ‘hard’ examples. (the ones on which we committed mistakes in the previous iterations)
Some Boosting History

• The idea of boosting began with a learning theory question first asked in the late 80’s.
• The question was answered in 1989 by Robert Shapire resulting in the first theoretical boosting algorithm
• Shapire and Freund later developed a practical boosting algorithm called Adaboost
• Many empirical studies show that Adaboost is highly effective (very often they outperform ensembles produced by bagging)
Specifying Input Distributions

• AdaBoost works by invoking \textit{Learn} many times on different distributions over the training data set.
• Need to modify base learner protocol to accept a training set distribution as an input.

Protocol to \textit{Learn}:

\begin{center}
\begin{tabular}{|l|}
\hline
\textbf{Input:} \\
$S$ - Set of $N$ labelled training instances. \\
$D$ - Distribution over $S$ where $D(i)$ is the weight of the $i$'th training instance (interpreted as the probability of observing $i$'th instance). Where $\sum_{i=1}^{N} D(i) = 1$.
\hline
\textbf{Output:} \\
$h$ - a hypothesis from hypothesis space $H$
\hline
\end{tabular}
\end{center}

$D(i)$ can be viewed as indicating to base learner \textit{Learn} the importance of correctly classifying the $i$'th training instance.
AdaBoost (High level steps)

- AdaBoost performs $L$ boosting rounds, the operations in each boosting round $l$ are:

1. Call \textit{Learn} on data set $S$ with distribution $D_l$ to produce $l^{th}$ ensemble member $h_l$, where $D_l$ is the distribution of round $l$.
2. Compute the $l + 1$ $th$ round distribution $D_{l+1}$ by putting more weight on instances that $h_l$ makes mistakes on.
3. Compute a voting weight $\alpha_l$ for $h_l$

The ensemble hypothesis returned is:

$$H = \{h_1, \ldots, h_L\}, \text{weightedVote}(\alpha_1, \ldots, \alpha_L)$$
AdaBoost algorithm:

**Input:** Learn - Base learning algorithm.
S - Set of N labeled training instances.

**Output:** $H = \langle \{h_1, \ldots, h_L\}, WeightedVote(\alpha_1, \ldots, \alpha_L) \rangle$

**Initialize** $D_1(i) = 1/N$, for all $i$ from 1 to $N$. (uniform distribution)

**FOR** $l = 1, 2, \ldots, L$ **DO**

1. $h_l = Learn(S, D_l)$
2. $\varepsilon_l = error(h_l, S, D_l)$
3. $\alpha_l = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_l}{\varepsilon_l} \right)$ ;; if $\varepsilon_l < 0.5$ implies $\alpha_l > 0$
4. $D_{l+1}(i) = D_l(i) \times \begin{cases} e^{\alpha_l}, & h_l(x_i) \neq y_i \\ e^{-\alpha_l}, & h_l(x_i) = y_i \end{cases}$ for $i$ from 1 to $N$

**Normalize** $D_{l+1}$ ;; can show that $h_l$ has 0.5 error on $D_{l+1}$

Note that $\varepsilon_l < 0.5$ implies $\alpha_l > 0$ so weight is decreased for instances $h_t$ predicts correctly and increases for incorrect instances.
Learning with Weights

• It is often straightforward to convert a base learner to take into account an input distribution $D$.
  – Decision trees?
  – Neural nets?
  – Logistic regression?

• When it’s not straightforward, we can resample the training data according to $D$
AdaBoost (Example)

**Base Learner:** Decision Stump Learner (i.e. single test decision trees)

Original Training set: Equal weights to all training samples

\[ h_1 \]

\[ D_1 \]

\[ \epsilon_1 = 0.30 \]

\[ \alpha_1 = 0.42 \]

Taken from “A Tutorial on Boosting” by Yoav Freund and Rob Schapire
AdaBoost (Example)

ROUND 1

\[ h_1 \]

\[ \varepsilon_1 = 0.30 \]
\[ \alpha_1 = 0.42 \]

\[ D_2 \]
AdaBoost (Example)

ROUND 2

$\varepsilon_2 = 0.21$
$\alpha_2 = 0.65$

$D_3$
AdaBoost(Example)

ROUND 3

\[ h_3 \]

\[ \varepsilon_3 = 0.14 \]
\[ \alpha_3 = 0.92 \]
$H_{\text{final}}$ = sign(0.42 + 0.65 + 0.92)
Property of Adaboost

- Suppose $L$ is a weak learner
  - $\varepsilon_l < 0.5$ (slightly better than random guesses)
  - Training error goes to zero exponentially fast
Overfitting?

- Boosting drives training error to zero, will it overfit?
- Curious phenomenon

Boosting is often robust to overfitting (not always)
Test error continues to decrease even after training error goes to zero
Figure 1: Error curves and margin distribution graphs for bagging and boosting C4.5 on the letor dataset. Learning curves are shown directly above corresponding margin distribution graphs. Each learning-curve figure shows the training and test error curves (lower and upper curves, respectively) of the combined classifier as a function of the number of classifiers combined. Horizontal lines indicate the test error rate of the base classifier as well as the test error of the final combined classifier. The margin distribution graphs show the cumulative distribution of margins of the training instances after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed (mostly hidden) and solid curves, respectively.
Boosting Performance

- Comparing C4.5, boosting decision stumps, boosting C4.5 using 27 UCI data set
  - C4.5 is a popular decision tree learner
Boosting vs Bagging of Decision Trees
AdaBoost as an Additive Model

• We will now derive AdaBoost in a way that can be adapted in various ways
• This recipe will let you derive “boosting-style” algorithms for particular learning settings of interest
  – E.g., general mis-prediction cost, semi-supervised learning
• These boosting-style algorithms will not generally be boosting algorithms in the theoretical sense but they often work quite well
AdaBoost: Iterative Learning of Additive Models

- Consider the final hypothesis: it takes the sign of an additive expansion of a set of base classifiers

\[ H(x) = \text{sign} \left[ f(x) \right] = \text{sign} \left[ \sum_{l=1}^{L} \alpha_l h_l(x) \right] \]

- AdaBoost iteratively finds at each iteration an \( h(\cdot) \) to add to \( f(\cdot) \)

\[ f_i(x) = f_{i-1}(x) + \alpha_i h_i(x) \]

- The goal is to minimize a loss function on the training example:

\[ \sum_{i=1}^{N} L \left( y^i, \sum_{l=1}^{L} \alpha_l h_l(x^i) \right) \]
• We would like to minimize the error:

\[ L(y^i, f(x^i)) = [y^i \neq \text{sgn}(f(x^i))] \quad \text{Note } y \in \{ -1, 1 \} \]

• Instead, Adaboost can be viewed as minimizing an exponential loss function, which is a smooth upper bound on 0/1 error:

\[ L(y^i, f(x^i)) = e^{-y^i f(x^i)} \]

\[
\begin{align*}
\text{arg min}_f & \sum_{i=1}^{N} L(y^i, f(x^i)) \\
& = \text{arg min}_{\alpha, h_l} \sum_{i=1}^{N} e^{-y^i \cdot [f_{l-1}(x^i) + \alpha \cdot h_l(x^i)]} \\
& = \text{arg min}_{\alpha, h_l} \sum_{i=1}^{N} e^{-y^i f_{l-1}(x^i)} \cdot e^{-y^i \alpha \cdot h_l(x^i)}
\end{align*}
\]

at iteration \( l \), look for \( h_l \) and \( \alpha \)
Fix $\alpha$ and optimize $h_l$

$$\arg \min_{h_l} \sum_{i=1}^{N} e^{-y^i \cdot f_{l-1}(x^i)} \cdot e^{-y^i \cdot \alpha \cdot h_l(x^i)}$$

$$= \arg \min_{h_l} \sum_{i=1}^{N} w^i \cdot e^{-y^i \cdot \alpha \cdot h_l(x^i)}$$

$$= \arg \min_{h_l} \sum_{y^i = h_l(x^i)}^{N} w^i \cdot e^{-\alpha} + \sum_{y^i \neq h_l(x^i)}^{N} w^i \cdot e^\alpha$$

$$= \arg \min_{h_l} \sum_{i=1}^{N} w^i \cdot e^{-\alpha} - \sum_{y^i \neq h_l(x^i)}^{N} w^i \cdot e^{-\alpha} + \sum_{y^i \neq h_l(x^i)}^{N} w^i \cdot e^\alpha$$

$$= \arg \min_{h_l} \ e^{-\alpha} \sum_{i=1}^{N} w^i + (e^\alpha - e^{-\alpha}) \cdot \sum_{y^i \neq h_l(x^i)}^{N} w^i$$

$$= \arg \min_{h_l} \ e^{-\alpha} + (e^\alpha - e^{-\alpha}) \cdot \frac{\sum_{y^i \neq h_l(x^i)}^{N} w^i}{\sum_{i=1}^{N} w^i}$$

$$= \arg \min_{h_l} \ e^{-\alpha} + (e^\alpha - e^{-\alpha}) \cdot \frac{\sum_{i=1}^{N} w^i}{\sum_{i=1}^{N} w^i} \cdot [y^i \neq h_l(x^i)]$$
Fix $h_l(\cdot)$ and find $\alpha$

$$\arg\min_{\alpha} \ e^{-\alpha} + (e^\alpha - e^{-\alpha}) \cdot \sum_{i=1}^{N} \frac{w_i^i}{\sum_{i=1}^{N} w_i^i} \cdot I[y^i \neq h_l(x^i)]$$

$$= \arg\min_{\alpha} \ e^{-\alpha} + (e^\alpha - e^{-\alpha}) \cdot \epsilon_l$$

$$J(\alpha)$$

$$\frac{\partial J(\alpha)}{\partial \alpha} = -e^{-\alpha} + \epsilon_l \cdot e^\alpha + \epsilon_l \cdot e^{-\alpha}$$

$$= e^{-\alpha} \cdot (\epsilon_l - 1) + e^\alpha \cdot \epsilon_l = 0$$

$$\Rightarrow \ e^\alpha \cdot \epsilon_l = e^{-\alpha} \cdot (1 - \epsilon_l)$$

$$\Rightarrow \ e^{2\alpha} = \frac{1 - \epsilon_l}{\epsilon_l} \Rightarrow \alpha = \frac{1}{2} \ln \frac{1 - \epsilon_l}{\epsilon_l}$$
Pitfall of Boosting: sensitive to noise and outliers

**Good 🌸**: Can identify outliers since focuses on examples that are hard to categorize

**Bad 😞**: Too many outliers can degrade classification performance dramatically increase time to convergence
Summary: Bagging and Boosting

• Bagging
  – Resample data points
  – Weight of each classifier is the same
  – Only variance reduction
  – Robust to noise and outliers

• Boosting
  – Reweight data points (modify data distribution)
  – Weight of classifier vary depending on accuracy
  – Reduces both bias and variance
  – Can hurt performance with noise and outliers