AdaBoost

Freund and Schaipre:
Experiments with a new Boosting Algorithm

Schapire and Singer:
Improved Boosting Algorithms using Confidence-rated Predictions

Ian Fasel

October 23, 2001

Resampling for Classifier Design

- Arcing – "adaptive reweighting and combining"
  - reusing or selecting data in order to improve classification
- Two most popular:
  - Bagging (Breiman, 1994)
  - AdaBoost (Freund and Schapire, 1996)
- Combine the results of multiple "weak" classifiers into a single "strong" classifier.

The horse-track problem

Gambler wants to write a computer program to accurately predict winner of horse races.

- Gathers Historical horse-race data.
  - input vector:
    - Odds
    - dry or muddy
    - jockey
  - output label:
    - win or lose
- Discovers:
  - easy to find rules of thumb that are "often" correct.
  - hard to find a single rule that is very highly accurate.

The gambler's plan

The gambler has decided intuitively that he wants to use arcing, and he wants his final solution to be some simple combination of simply derived rules.

Repeat T times:

1. Examine small sample of races.
2. Derive rough rule-of-thumb:
   - e.g., "bet on horse with most favorable odds"
3. Select new sample, derive 2nd rule-of-thumb:
   - "If track is muddy, then bet on the lightest horse"
   - "otherwise, choose randomly"

end

Questions

- How to choose samples?
  - Select multiple random samples?
  - Concentrate only on the errors?
- How to combine rules-of-thumb into a single accurate rule?

More Formally:

Given: training data $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y} = \{-1, +1\}$

- For $t = 1, \ldots, T$:
  1. Train *Weak Learner* on the training set.
     Let $h_t : \mathcal{X} \to \{-1, +1\}$ represent the classifier obtained after training.
  2. Modify the training set somehow
- The final hypothesis $H(x)$ is some combination of all the weak hypotheses:

$$H(x) = f(h(x)) \tag{1}$$

The question is how to modify the training set, and how to combine the weak classifiers.

Bagging

The simplest algorithm is called Bagging, used by Breiman 1994

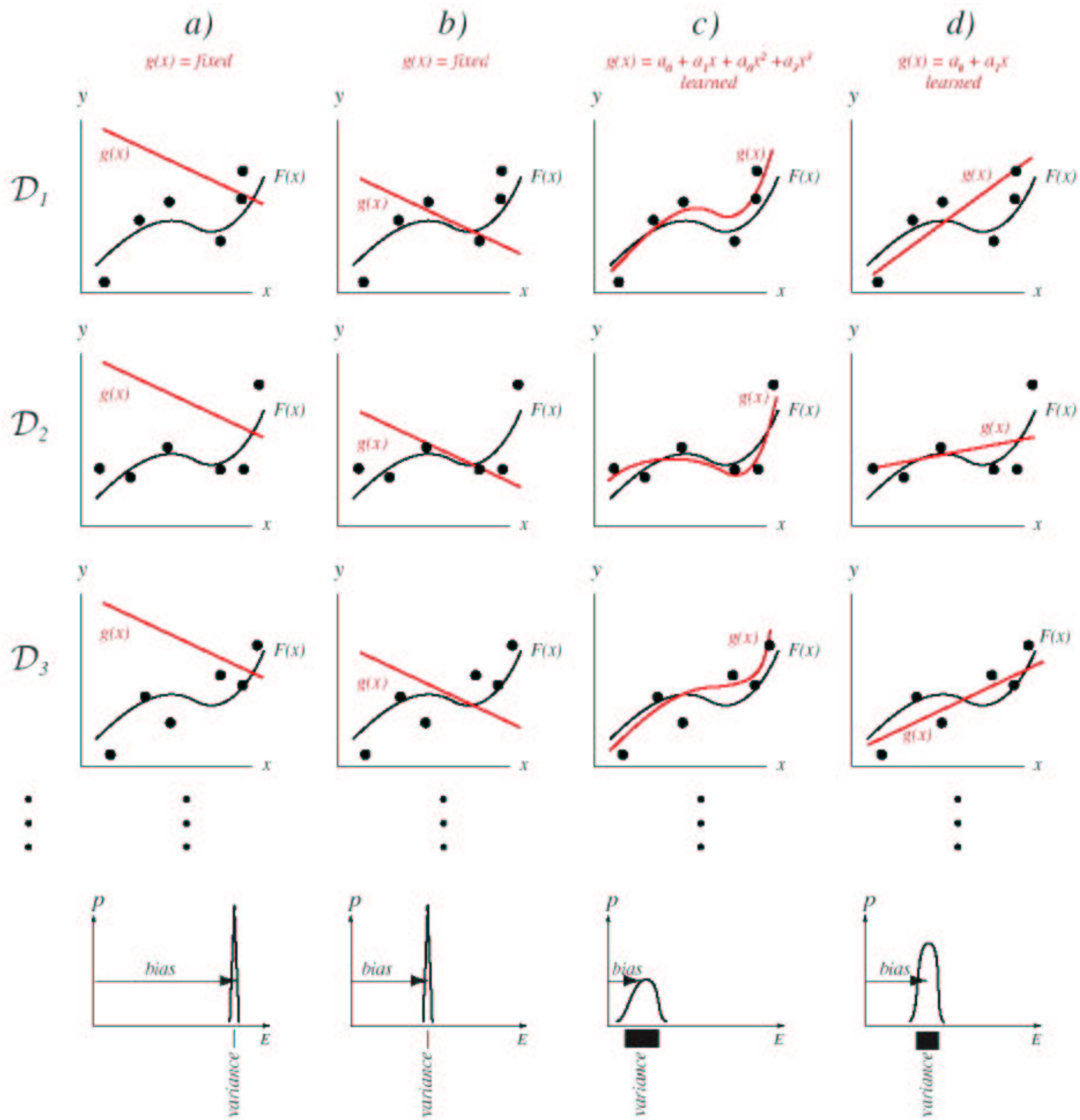
Algorithm:

Given m training examples, repeat for $t = 1 \ldots T$:

- Select, at random *with replacement*, m training examples.
- Train learning algorithm on selected examples to generate hypothesis h_t

Final hypothesis is simple vote: $H(x) = MAJ(h_1(x), \ldots, h_T(x))$.

Bias Variance Dilemma



Bagging: Pros and Cons

- Bagging reduces variance
  - Helps improve *unstable* classifiers: i.e., "small" changes in training data lead to significantly different classifiers and "large" changes in accuracy.
  - no proof for this, however
- does not reduce bias

Boosting

Two modifications

1. instead of a random sample of the training data, use a weighted sample to focus learning on most difficult examples.
2. instead of combining classifiers with equal vote, use a weighted vote.

Several previous methods (Schapire, 1990; Freund, 1995) were effective, but had limitations. We skip ahead to Freund and Schapire 1996.

AdaBoost (Freund and Schapire, 1996)

- Initialize distribution over the training set $D_1(i) = 1/m$
- For $t = 1, \ldots, T$:
  1. Train *Weak Learner* using distribution D_t.
  2. Choose a weight (or confidence value) $\alpha_t \in \mathbf{R}$.
  3. Update the distribution over the training set:

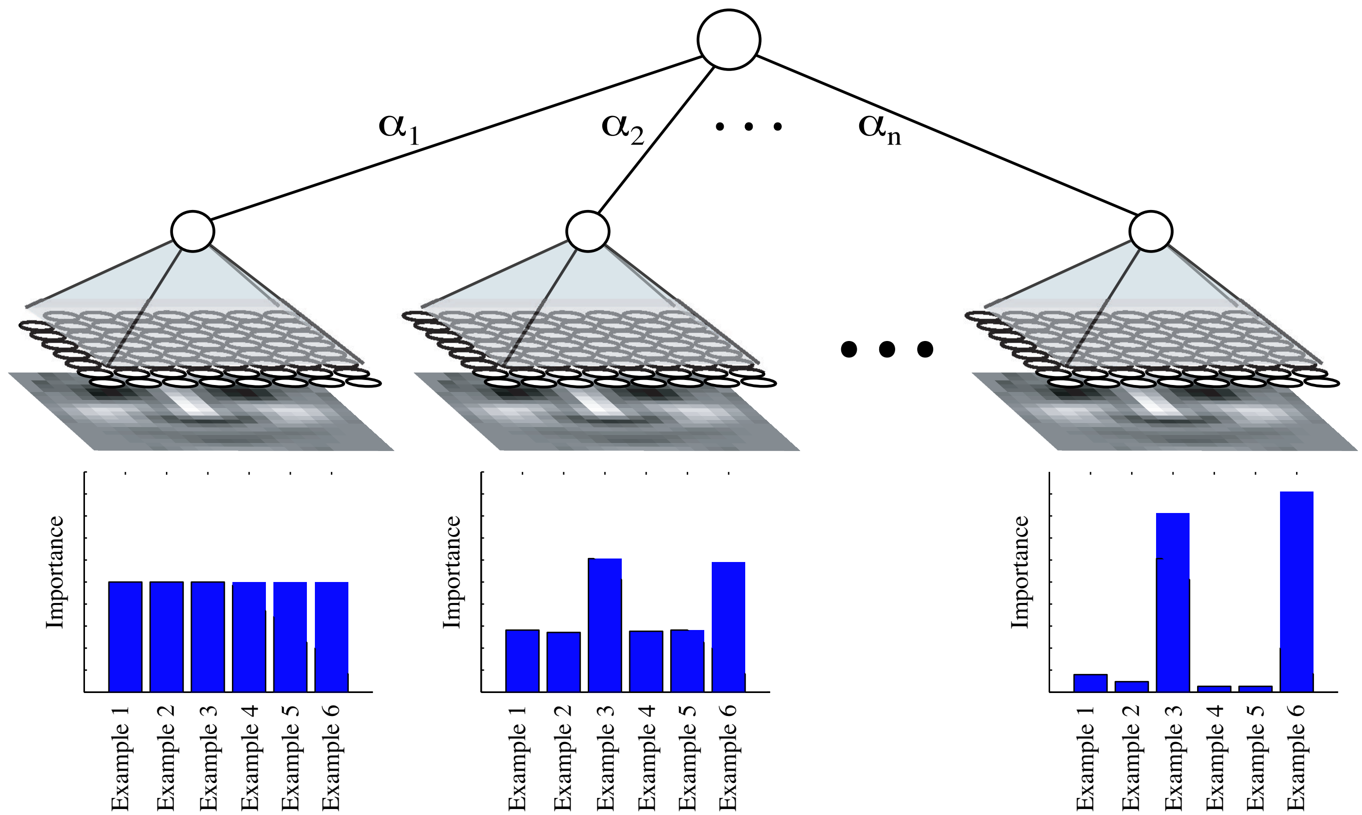
$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t} \tag{2}$$

     Where Z_t is a normalization factor chosen so that D_{t+1} will be a distribution

- Final vote $H(x)$ is a weighted sum:

$$H(x) = \text{sign}(f(x)) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{3}$$

If the underlying classifiers are linear networks, then AdaBoost builds multilayer perceptrons one node at a time.



However, underlying classifier can be anything. Decision trees, neural networks, hidden Markov models . . .

How do we pick α?

To decide how to pick the αs, we have to understand what the relationship is between the distribution, the α_t, and the training error. (Later, we can show that reducing training error this way should reduce test error as well).

Theorem 1: *Error is minimized by minimizing Z_t*

Proof:

$$\begin{aligned} D_{T+1}(i) &= \frac{1}{m} \cdot \frac{e^{-y_i \alpha_1 h_1(x_i)}}{Z_1} \cdot \ldots \cdot \frac{e^{-y_i \alpha_T h_T(x_i)}}{Z_T} \\ &= \frac{e^{\sum_t -y_i \alpha_t h_t(x_i)}}{m \prod_t Z_t} = \frac{e^{-y_i \sum_t \alpha_t h_t(x_i)}}{m \prod_t Z_t} \\ &= \frac{e^{-y_i f(x_i)}}{m \prod_t Z_t} \end{aligned} \tag{4}$$

But, if $H(x_i) \neq y_i$ then $y_i f(x_i)) \leq 0$, implying that $e^{-y_i f(x_i)} \geq 1$. Thus,

$$\begin{aligned} [\![H(x_i) \neq y_i]\!] &\leq e^{-y_i f(x_i)} \\ \frac{1}{m} \sum_i [\![H(x_i) \neq y_i]\!] &\leq \frac{1}{m} \sum_i e^{-y_i f(x_i)} \end{aligned} \tag{5}$$

Combining these results,

$$\begin{aligned}\frac{1}{m}\sum_i [\![H(x_i) \neq y_i]\!] &\leq \frac{1}{m}\sum_i e^{-y_i f(x_i)} \\ &= \sum_i \left(\prod_t Z_t\right) D_{T+1}(i) \qquad (6) \\ &= \prod_t Z_t \quad \text{(since } D_{T+1} \text{ sums to 1).}\end{aligned}$$

Thus, we can see that minimizing Z_t will minimize this error bound.

Consequences:

1. We should choose α_t to minimize Z_t
2. We should modify the "weak learner" to minimize Z_t (instead of, say, squared error).

Finding α_t analytically

Define:

$u_i = y_i h(x_i)$. (Then u_i is positive if $h(x_i)$ is correct, and negative if it is incorrect. Magnitude is confidence.)

$r = \sum_i D_i u_i$. ($r \in [-1, +1]$, this is a measure of the overall error rate.)

If we restrict $h(x_i) \in [-1, 1]$, we can approximate Z:

$$\begin{aligned} Z &= \sum_i D(i) e^{-\alpha u_i} \\ &\leq \sum_i D(i) \left(\frac{1+u_i}{2} e^{-\alpha u_i} + \frac{1-u_i}{2} e^{\alpha u_i} \right) \end{aligned} \tag{7}$$

Note: if $h(x_i) \in \{-1, 1\}$, this approximation is exact

We can find α to minimize Z analytically by finding $\frac{dZ(\alpha)}{d\alpha} = 0$,[a] which gives us

$$\alpha = \frac{1}{2} \ln \left(\frac{1+r}{1-r} \right)$$

Plugging this into equation 7, this α gives the upper bound

$$Z \leq \sqrt{1-r^2}$$

for a particular t, or, plugging into equation 6, we have an upper bound for the training error of H

$$\frac{1}{m} [\![H(x_i) \neq y_i]\!] \leq \prod_t Z_t \leq \prod_{t=1}^{t} \sqrt{1-r_t^2}$$

[a] For the careful, it can be shown that α chosen this way is guaranteed to minimize Z, and that it is unique.

Finding α numerically

When using real valued hypotheses $h(x_i) \in [-1, 1]$, the best choice of α to minimize Z must be found numerically.

If we do this, and find α such that $Z'(\alpha) = 0$, then

$$\begin{aligned} Z'(\alpha) = \frac{dZ(\alpha)}{d\alpha} &= \frac{d}{d\alpha} \sum_i D(i) e^{-\alpha u_i} \\ &= -\sum_i D(i) u_i e^{-\alpha u_i} \\ &= -Z \sum_i D_{t+1}(i) u_i = 0 \end{aligned}$$

So, either $Z = 0$ (the trivial case), or $\sum_i D_{t+1}(i) u_i = 0$.

In words, this means that with respect to the new distribution D_{T+1}, the current classifier h_t will perform at chance.

Relation to "Naive Bayes" rule

If we assume independence, then

$$p(y|h_1, h_2, \ldots h_t) \propto \prod_{i=1}^{T} p(h_i|y)$$

If we have $h_1 \ldots h_t$ independent classifiers, then Bayes optimal prediction $\hat{y} = \mathrm{sign}(\sum(h(x_i))$.

Since AdaBoost attempts to make the hypotheses independent, intuition is that this is the optimal combination.

Modifying the Weak Learner for AdaBoost

Just as we choose α_t to minimize Z_t, it is also sometimes possible to modify the weak learner h_t so that it minimizes Z explicitly.

This leads to several modifications of common weak learners

- A modified rule for branching in C4.5.
- For neural networks trained with backprop, simply multiply the weight change due to x_i by $D(i)$

Practical advantages of AdaBoost

- Simple and easy to program.
- No parameters to tune (except T).
- Provably effective, provided can consistently find rough rules of thumb
  - Goal is to find hypotheses barely better than guessing.
- Can combine with any (or many) classifiers to find weak hypotheses: neural networks, decision trees, simple rules of thumb, nearest-neighbor classifiers ...

Proof that it is easy to implement

```
[X, Y] = preprocessTrainingData(params);

% Initialize weight distribution vector
dist = ones(nImages,1) * (1/nImages);

for k = 1:nBoosts

    % do the ridge regression
    z = 1000;  % wild guess if this is good
    w(k) = pinv(X'*X + z*eye(nImages)) * X'*Y;

    % find alpha for this round of AdaBoost
    u = sign( I * w(k)) .* Y;
```

```
    % Find alpha numerically:
       % alpha_num = fminbnd(Z,-100,100,[],u,dist)
    % Or find alpha analytically
    r = sum(dist.*u);
    alpha(k) = .5*log((1+r)/(1-r));

    % make new distribution
    udist = dist .* exp(-1 * alpha(k) .* u);
    dist = udist/sum(udist);
end

[X, Y] = preprocessTestData(params);

H = sign(sum(alpha * sign( I * w )));
testErr = sum( H .* Y <= 0);
```

caveats

AdaBoost with decision trees has been referred to as "the best off-the-shelf classifier". However,

- If weak learners are actually quite strong (i.e., error gets small very quickly), boosting might not help
- If hypotheses too complex, test error might be much larger than training error

go to ppt slides