Bagging and Boosting

Gautam Kamath

Bagging

Bagging: Bootstrap Aggregating

- Bootstrap sampling + aggregation
- Example: estimate μ given $X_1, \dots, X_n \sim N(\mu, \sigma^2)$
- Simple solution: use empirical mean $\hat{\mu} = \frac{1}{n}\sum X_i$
  - Note $E[\hat{\mu}] = \mu, \mathrm{Var}[\hat{\mu}] = \mathrm{Var}\left[\frac{1}{n}\sum X_i\right] = \frac{1}{n^2}\mathrm{Var}[\sum X_i] = \frac{1}{n^2} \cdot n \cdot \mathrm{Var}[X_1] = \frac{\sigma^2}{n}$.
- Variance may be very large…
- If we have Bn points from $N(\mu, \sigma^2)$, can form $S_1 = \{X_1, \dots, X_n\}, S_2 = \{X_{n+1}, \dots, X_{2n}\}, \dots, S_B = \{X_{(B-1)n+1}, \dots, X_{Bn}\}$
- $\hat{\mu}^{(j)} = \frac{1}{n}\sum_{z \in S_j} z$ and $\hat{\mu}^{(avg)} = \frac{1}{B}\sum_{j \in [B]} \hat{\mu}^{(j)}$
- $E\left[\hat{\mu}^{(avg)}\right] = \mu, \mathrm{Var}\left[\frac{1}{B}\sum_{j \in [B]} \hat{\mu}^{(j)}\right] = \frac{1}{B^2} \cdot B \cdot \frac{\sigma^2}{n} = \frac{\sigma^2}{nB}$

Bootstrap Sampling

- Averaging over B independent datasets reduces variance by factor B
  - But needs B times more data…
- Idea: cheat and just reuse parts of the same dataset!
  - Not independent, but still seems to work
- Bootstrap sampling $\approx$ sampling with replacement

Bootstrap Example

- Given dataset of size n, create B datasets of size n, where each is constructed by drawing n samples (with replacement) from original
- Example: Given dataset X_1, X_2, X_3, X_4, X_5
- $S_1 = \{X_3, X_4, X_1, X_1, X_4\}, S_2 = \{X_5, X_5, X_3, X_1, X_2\}, \ldots, S_B = \cdots$
- Again, use $\hat{\mu}^{(j)} = \frac{1}{n}\sum_{z \in S_j} z$ and $\hat{\mu}^{(avg)} = \frac{1}{B}\sum_{j \in [B]} \hat{\mu}^{(j)}$
- $E\left[\hat{\mu}^{(avg)}\right] = \mu, \mathrm{Var}\left[\frac{1}{B}\sum_{j \in [B]} \hat{\mu}^{(j)}\right] = ??$
- Can't compute variance as before, since we lost independence
- Still works in practice by reducing variance anyway!

Bagging in ML

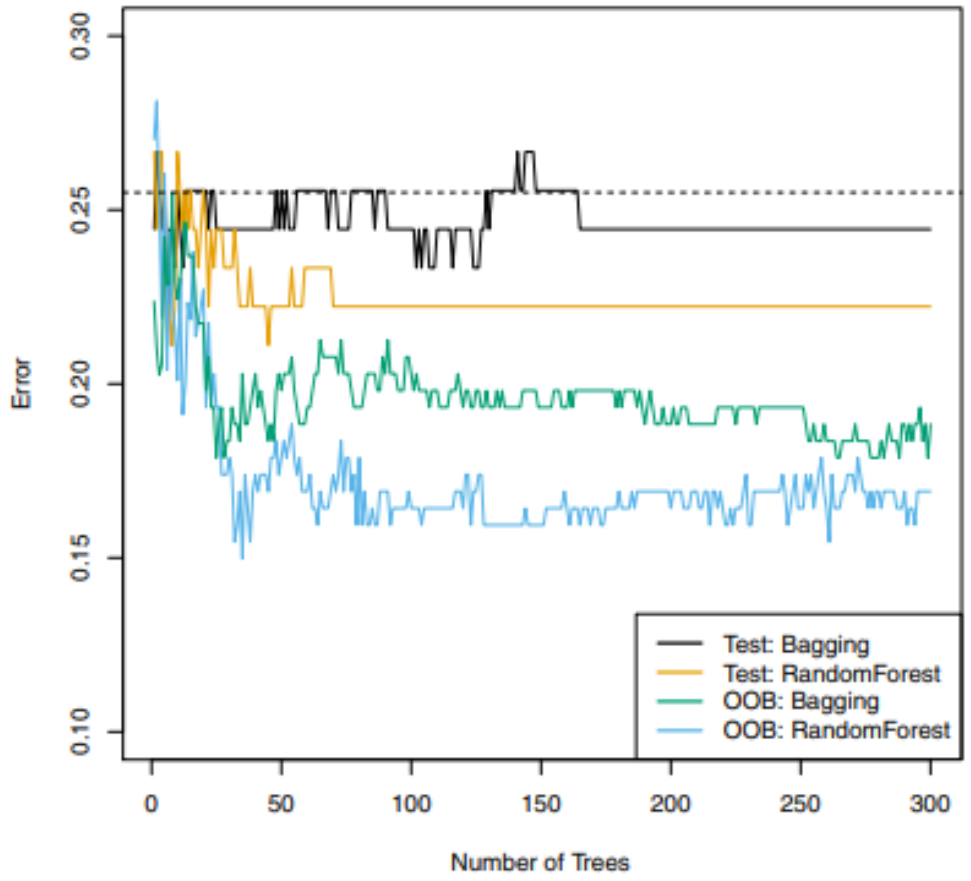
- Some methods are inherently high variance
- Decision trees
  - Learn decision tree on 2 halves of the same dataset → (very?) different trees
- Use bootstrap aggregating to reduce variance

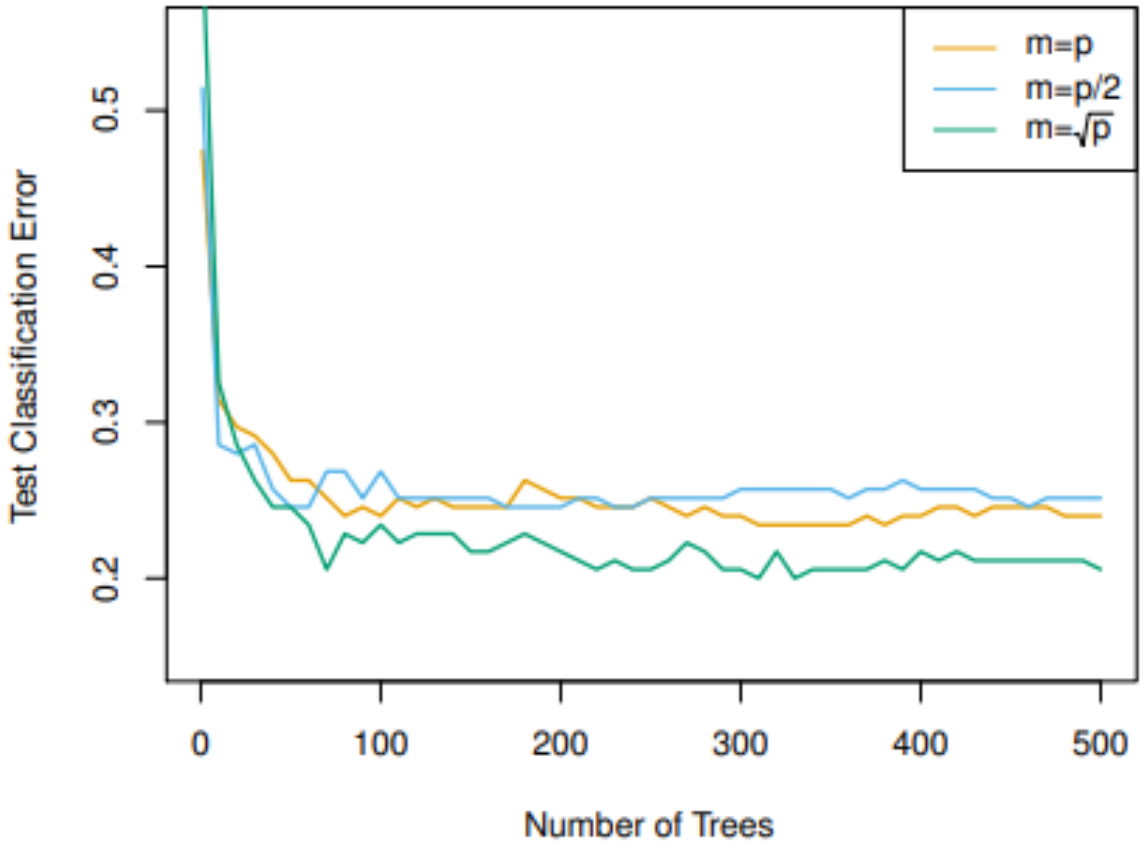
1. Bootstrap sample B datasets of size n
2. Run some learning algorithm on each, get classifiers $\hat{f}^{(1)}, \ldots, \hat{f}^{(B)}$
3. Aggregate $\hat{f}^{(1)}, \ldots, \hat{f}^{(B)}$
   - How? Regression $\hat{f}(x) = \frac{1}{B}\sum \hat{f}^{(j)}(x)$.
   - Classification $\hat{f}(x) =$ majority vote of $\hat{f}^{(j)}(x)$

Random Forests

- Bagging on decision trees
  - Twist to add randomness/make bootstrap samples "look" more independent
- Standard decision trees: When choosing which feature to split on, look at all d features and pick the "best" one
  - Downside: if one feature is very informative, will be used in all B datasets
- Random forests: When choosing which feature to split on, look at a random subsample of $m \ll d$ features and pick the "best" one
  - Say, $m = \sqrt{d}$
  - Resample for each split

Random Forests





Boosting

Boosting

- Given several “weak learners,” can we combine them into a “strong learner”?
- Weak learner: “55% accurate” (slightly better than random guess)
  - Focus on binary classification today
- Strong learner: “90%+ accurate” (a good classifier)
- Iterative process. Train a classifier. “Downweight” points it gets right, “upweight” points it gets wrong. Train classifier on new weighted dataset (draw)
- Bit of a diversion until we get to that…

“Online Learning with Experts”

- Example: Horse racing, with n horses and T races. How to choose which horse to bet on? How to update your bet after each race?
- More general setting, T rounds of the following:
  1. At round t, algorithm specifies weights $p_1^{(t)}, \dots, p_n^{(t)}$ such that $\sum_i p_i^{(t)} = 1$
     - Choose a distribution over the different “experts”
  2. Algorithm experiences “loss” at time t of $\langle p^{(t)}, \ell^{(t)} \rangle$
     - $\ell^{(t)} \in [0,1]^n$ is an adversarially picked loss vector
- Goal: Minimize $\sum_{t=1}^{T} \langle p^{(t)}, \ell^{(t)} \rangle$
- Try to compete with “best single expert in hindsight”
  - $\min_i \sum_{t=1}^{T} \ell_i^{(t)}$ -- same as goal when $p_i^{(t)} = 1$ for all t

Hedge Algorithm

Hedge(β), where $\beta \in [0,1]$

1. Initialize $w^{(1)} = [1/n, \ldots, 1/n] \in \mathbf{R}^n$
2. For $t = 1, \ldots, T$
    1. Set $p^{(t)} = \frac{w^{(t)}}{\sum_i w_i^{(t)}}$ (normalize w into a distribution)
    2. Receive loss $\langle p^{(t)}, \ell^{(t)} \rangle$
    3. Update $w_i^{(t+1)} = w_i^{(t)} \beta^{\ell_i^{(t)}}$ (downweight experts based on loss)

Guarantee: $\sum_{t=1}^{T} \langle p^{(t)}, \ell^{(t)} \rangle \leq \frac{1}{1-\beta} \left(\log n + \log(1/\beta) \min_i \sum_{t=1}^{T} \ell_i^{(t)} \right)$

- Must choose β to balance the two costs

Hedge Guarantees

- $\frac{1}{T}\sum_{t=1}^{T}\langle p^{(t)}, \ell^{(t)}\rangle \leq \frac{1}{T}\min_i \sum_{t=1}^{T} \ell_i^{(t)} + O\left(\sqrt{\frac{\log n}{T}}\right)$
- LHS: average loss at each step
- RHS: average loss of best expert in hindsight, plus “regret”
- Regret goes to 0 as $T \to \infty$
- We can be very competitive with choosing the best expert in hindsight!
- Very powerful framework! Useful in linear programming, game theory, etc.
- Now, how do we use this for standard ML classification…?

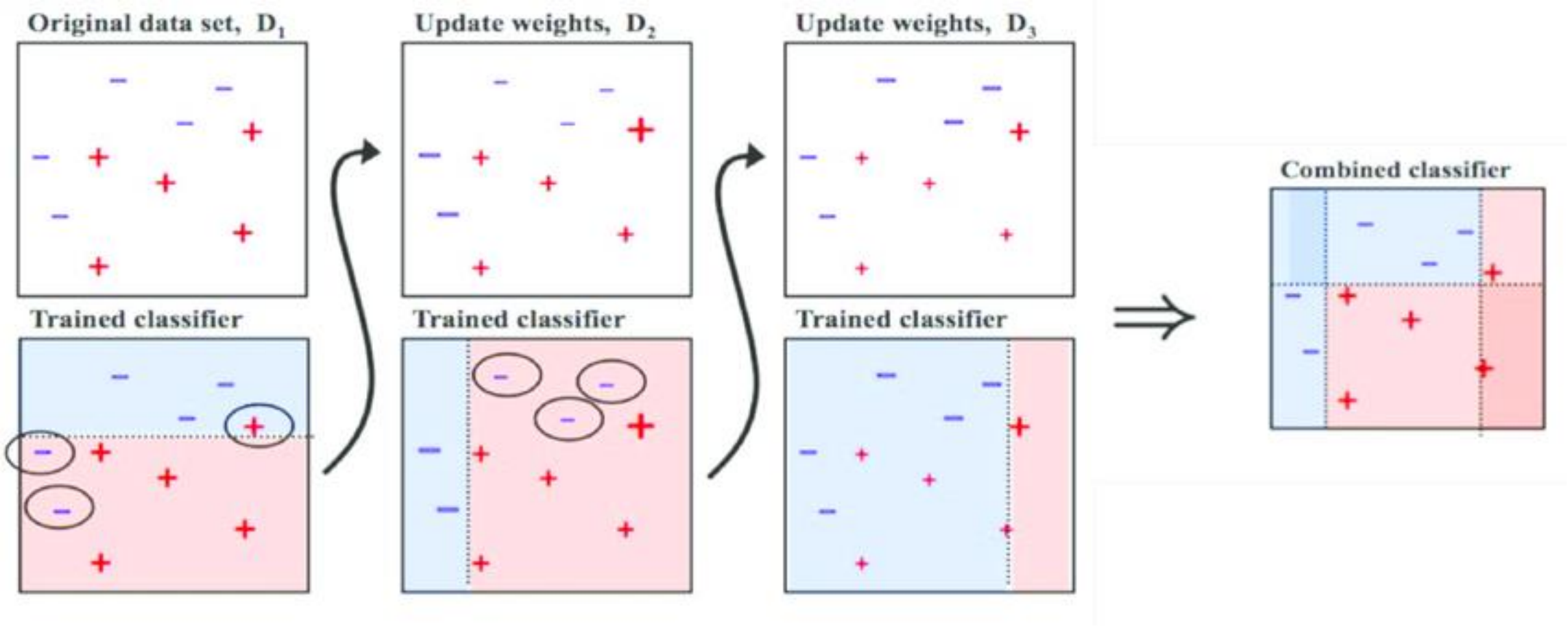
AdaBoost

- Given algorithm WeakLearn that gets 55% accuracy on a training set. Can we boost this to high probability?
- Wrong way: Run the algorithm many times on the dataset, treat resulting classifiers as “experts.” “Put large weight on good classifiers”
- Right way: Treat the *datapoints* as experts. Put large weight on points that haven’t been learned yet.

AdaBoost

1. Initialize $w^{(1)} = [1/n, \dots, 1/n] \in \mathbf{R}^n$
2. For $t = 1, \dots, T$
   1. Set $p^{(t)} = \frac{w^{(t)}}{\sum_i w_i^{(t)}}$ (normalize w into a distribution)
   2. Run WeakLearn on training set (with weights $p^{(t)}$)
      - Obtain classifier $h^{(t)}$ which maps (x, y) datapoints to $[0,1]$ (confidence in classification)
   3. Calculate error $\varepsilon_t = \sum_i p_i^{(t)} |h^{(t)}(x_i) - y_i|$ (should be < 0.5 by WeakLearn guarantees)
   4. Define $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$, if $\varepsilon_t \leq 1/2$ set $w_i^{(t+1)} = w_i^{(t)} \beta_t^{1-|h^{(t)}(x_i)-y_i|}$
      - Note: If ε_t big, then β_t is big. Many errors, so don't downweight points!
3. $h(x) = 1$ if $\sum_{t=1}^{T} \log\left(\frac{1}{\beta_t}\right) h^{(t)}(x) \geq \frac{1}{2} \sum_{t=1}^{T} \log\left(\frac{1}{\beta_t}\right)$, 0 else

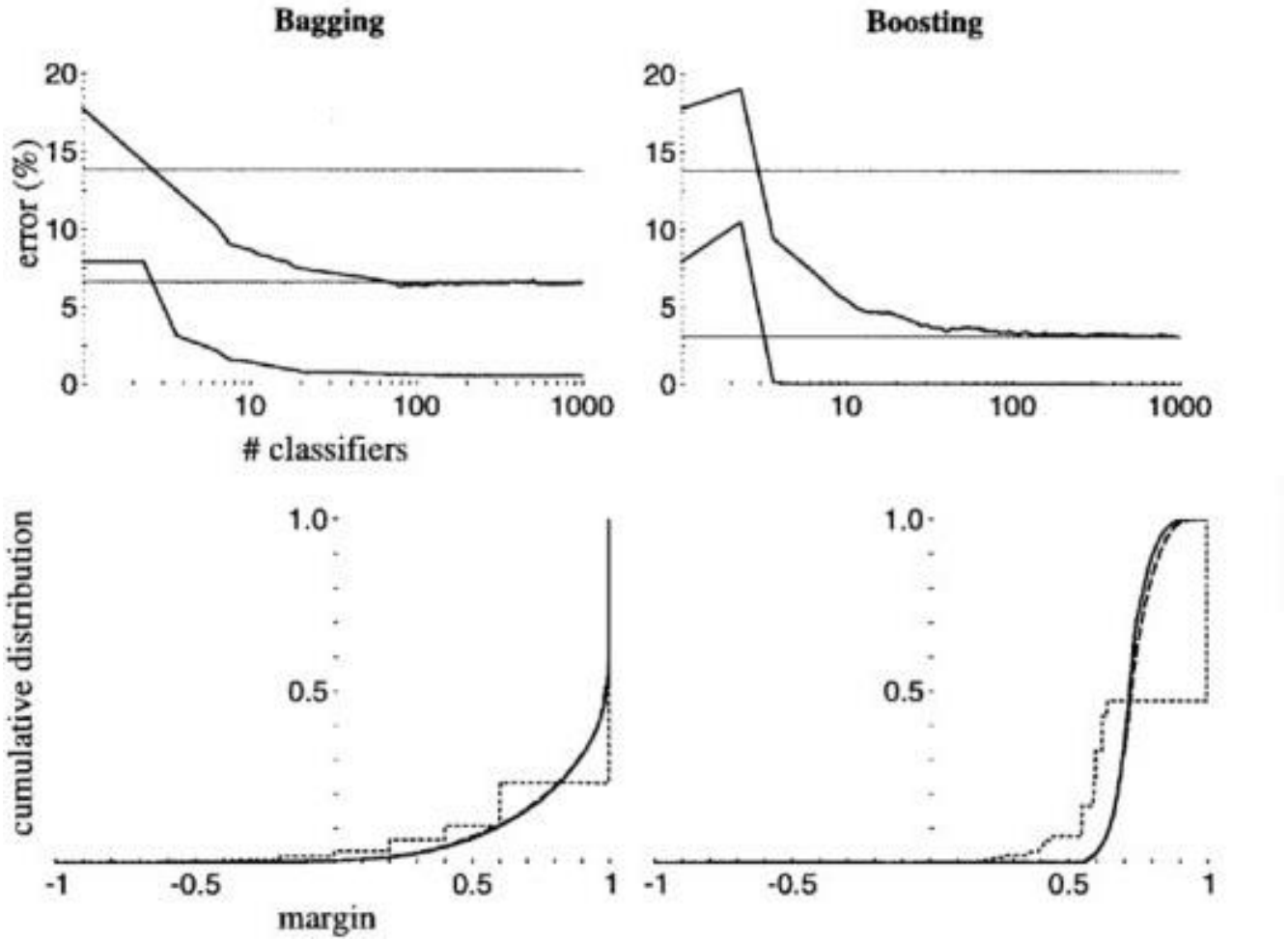
Illustration of AdaBoost



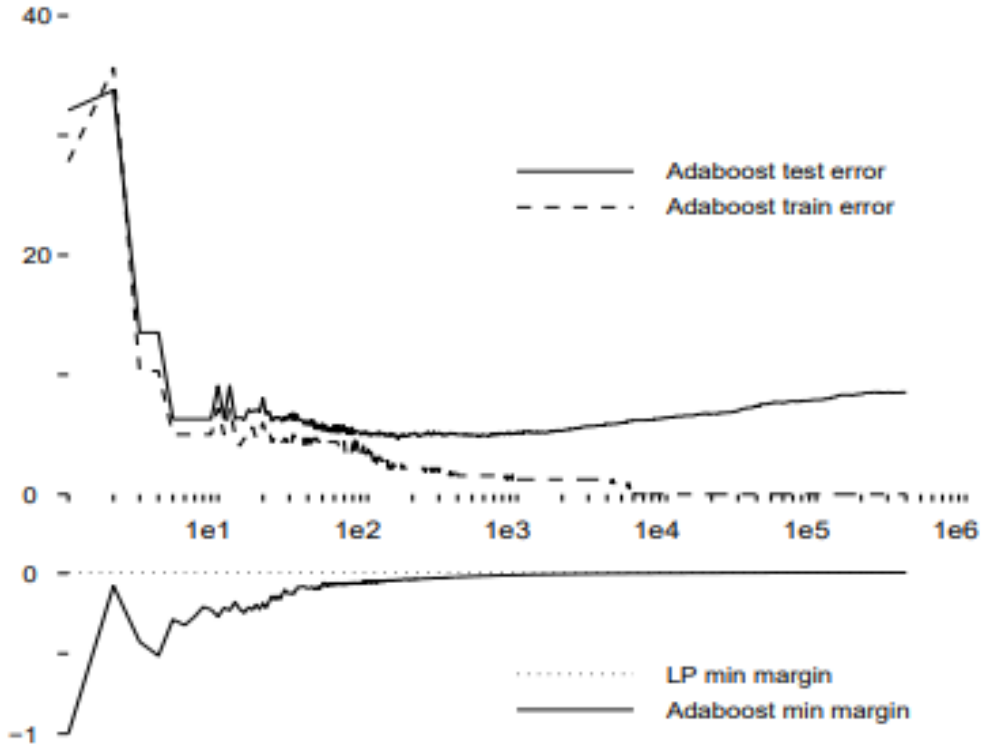
Training Error

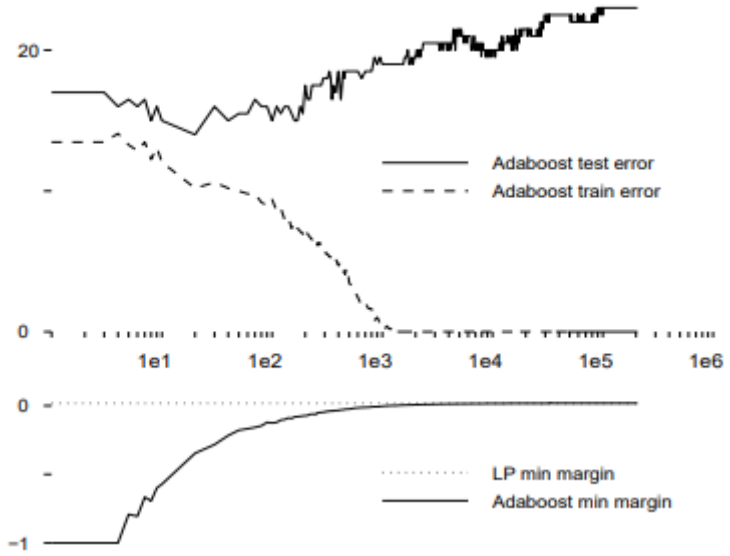
- Training error: $\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{h(x_i) \neq y_i\} \leq 2^T \prod_{t=1}^{T} \sqrt{\varepsilon_t(1-\varepsilon_t)}$
- Suppose we say $\varepsilon_t \leq \frac{1}{2} - \gamma$ (bound on error of t-th classifier)
- Then training error $\leq \exp(-2T\gamma^2)$ (decreases exponentially fast)
- But we really want good test error…
- An alternate perspective: gradient descent on loss $\sum e^{-y_i h(x_i)}$
- Generalize well when using a simple base classifier

Overfitting with AdaBoost



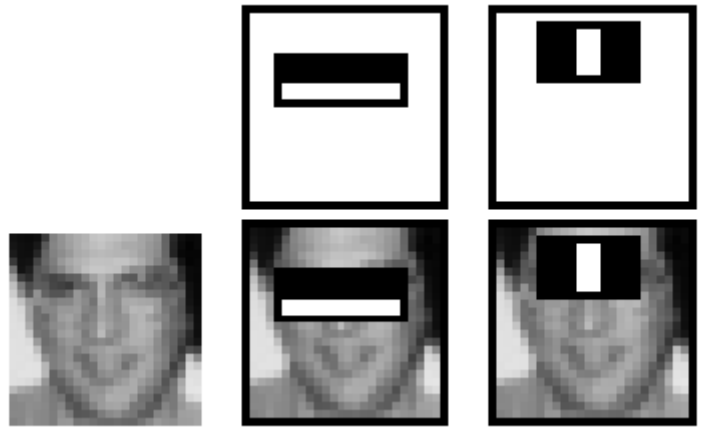
Overfitting with AdaBoost



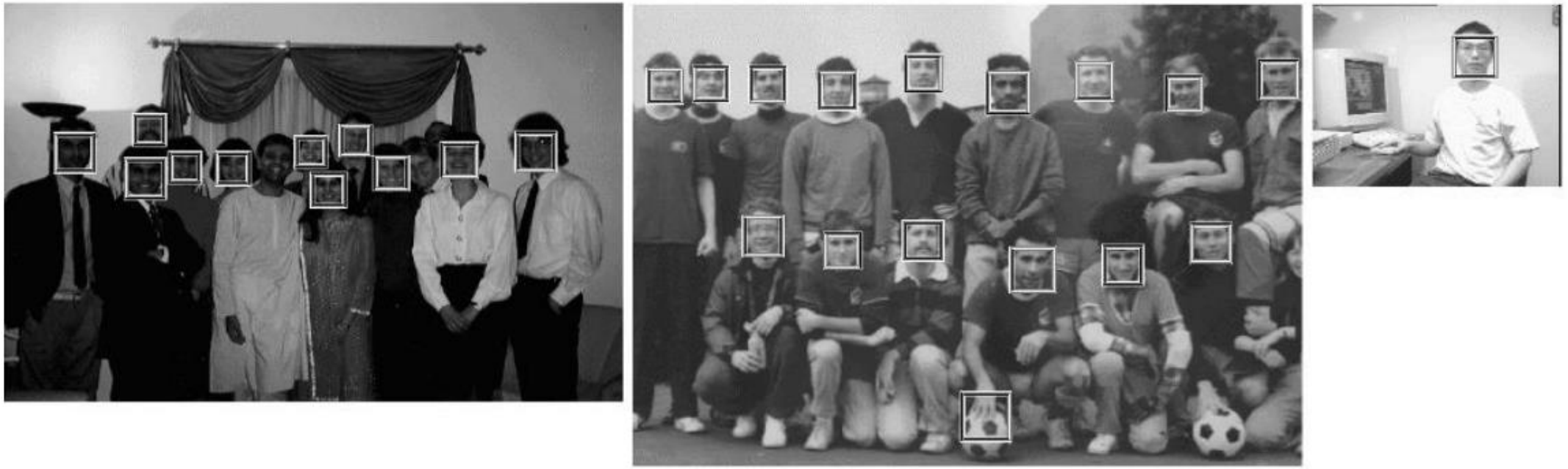


Face detection application (Viola, Jones ’01)

- Start with very very simple classifiers
- Use boosting to combine into something better



Face detection application (Viola, Jones ’01)



Bagging and Boosting

- “Simple” way to improve performance
- Generic, flexible with any base learner
- Loses some interpretability
- Bagging: can be done in parallel
- Boosting: inherently sequential (thus takes lost of time)