Fighting the bias-variance tradeoff

- Simple (a.k.a. weak) learners are good
  - e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)
  - Low variance, don’t usually overfit too badly

- Simple (a.k.a. weak) learners are bad
  - High bias, can’t solve hard learning problems

- Can we make weak learners always good???
  - No!!
  - But often yes…
Voting (Ensemble Methods)

- Instead of learning a single (weak) classifier, learn **many weak classifiers** that are good at different parts of the input space.

- **Output class**: (Weighted) vote of each classifier
  - Classifiers that are most “sure” will vote with more conviction.
  - Classifiers will be most “sure” about a particular part of the space.
  - On average, do better than single classifier!

\[ H(x) = \text{Sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right) \]

- But how do you ???
  - force classifiers to learn about different parts of the input space?
  - weigh the votes of different classifiers?

Boosting [Schapire, 1989]

- Idea: given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote.

- On each iteration:
  - weight each training example by how incorrectly it was classified.
  - Learn a hypothesis – \( h_t \): focus on “difficult” parts of the space.
  - A strength for this hypothesis – \( \alpha_t \).

\[ H(x) = \text{Sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right) \]

- Practically useful
- Theoretically interesting
Learning from weighted data

- Sometimes not all data points are equal
  - Some data points are more equal than others
- Consider a weighted dataset
  - $D(i)$ – weight of $i$th training example $(x_i, y_i)$
  - Interpretations:
    - $i$th training example counts as $D(i)$ examples
    - If I were to “resample” data, I would get more samples of “heavier” data points

- Now, in all calculations, whenever used, $i$th training example counts as $D(i)$ “examples”

Adaboost

- Initialize weights to uniform dist: $D_1(j) = 1/N$
- For $t = 1 \ldots T$
  - Train weak learner $h_t$ on distribution $D_t$ over the data
  - Choose weight $\alpha_t > 0$ (usually)
  - Update weights:
    $$D_{t+1}(j) = \frac{D_t(j) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$
    - Where $Z_t$ is normalizer:
      $$Z_t = \sum_{j=1}^{N} D_t(j) \exp(-\alpha_t y_i h_t(x_i))$$
    - So final weights add up to 1
- Output final classifier:
  $$H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$$
Picking Weight of Weak Learner

- Weigh \( h_t \) higher if it did well on training data (weighted by \( D_t \)):

  \[
  \alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
  \]

  Where \( \epsilon_t \) is the weighted training error:

  \[
  \epsilon_t = \sum_{j=1}^{N} D_t(j) I[h_t(x_j) \neq y_j]
  \]

  Magic:

  \( \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \)

  (e.g. \( \epsilon_t = 1 \) : perfectly wrong)

  \( \epsilon_t = 0 \) : perfectly right

  \( \epsilon_t > 0 \) : imperfect

  \( \epsilon_t < 0 \) : impossible

Why choose \( \alpha_t \) for hypothesis \( h_t \) this way? 

[Schapire, 1989]

Training error of final classifier is bounded by:

\[
\frac{1}{N} \sum_{j=1}^{N} \text{I}[H(x_j) \neq y_j] \leq \frac{1}{N} \sum_{j=1}^{N} \exp(-y_j f(x_j))
\]

Where \( f(x) = \sum_t \alpha_t h_t(x) \); \( H(x) = \text{sign}(f(x)) \)
Why choose $\alpha_t$ for hypothesis $h_t$ this way? [Schapire, 1989]

Training error of final classifier is bounded by:

$$\frac{1}{N} \sum_{j=1}^{N} \mathbb{I}[H(x_j) \neq y_j] \leq \frac{1}{N} \sum_{j=1}^{N} \exp(-y_j f(x_j)) = \prod_{t=1}^{T} Z_t$$

Where

$$f(x) = \sum_{t} \alpha_t h_t(x); \ H(x) = \text{sign}(f(x))$$

If we minimize $\prod_{t} Z_t$, we minimize our training error.

We can tighten this bound greedily, by choosing $\alpha_t$ and $h_t$ on each iteration to minimize $Z_t$.

$$Z_t = \sum_{j=1}^{N} D_t(j) \exp(-\alpha_t y_j h_t(x_j))$$
Why choose $\alpha_t$ for hypothesis $h_t$ this way?  

We can minimize this bound by choosing $\alpha_t$ on each iteration to minimize $Z_t$.

$$Z_t = \sum_{j=1}^{N} D_t(j) \exp(-\alpha_t y_j h_t(x_j))$$

For boolean target function, this is accomplished by [Freund & Schapire '97]:

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

You'll prove this in your homework! 😊

---

Strong, weak classifiers

- If each classifier is (at least slightly) better than random:
  - $\varepsilon_t < 0.5$

- AdaBoost will achieve zero training error (exponentially fast):

$$\frac{1}{N} \sum_{j=1}^{N} \mathbb{1}[H(x_j) \neq y_j] \leq \prod_{t=1}^{T} Z_t \leq \exp \left( -2 \sum_{t=1}^{T} \frac{1}{2} (1 - \varepsilon_t)^2 \right)$$

- $\varepsilon_t < \frac{1}{2}$ weak classifier must be strictly better than random

- Is it hard to achieve better than random training error?
Boosting results – Digit recognition

Boosting often
- Robust to overfitting
- Test set error decreases even after training error is zero

[Schapire, 1989]

Boosting: Experimental Results

Comparison of C4.5, Boosting C4.5, Boosting decision stumps (depth 1 trees), 27 benchmark datasets

[Freund & Schapire, 1996]
Boosting and Logistic Regression

Logistic regression assumes:
\[ f(x) = \mathbf{w}_0 + \sum_{i=1}^{n} \mathbf{w}_i \mathbf{h}_i(x) \]

And tries to maximize data likelihood:

\[
P(D|H) = \prod_{j=1}^{N} \frac{1}{1 + \exp(-y_j f(x_j))} \approx \sum_{i=1}^{n} \ln(1 + \exp(-y_j f(x_j)))
\]

Equivalent to minimizing log loss
Boosting and Logistic Regression

Logistic regression equivalent to minimizing log loss

\[ \sum_{j=1}^{N} \ln(1 + \exp(-y_j f(x_j))) \]

Boosting minimizes similar loss function!!

\[ \frac{1}{N} \sum_{j=1}^{N} \exp(-y_j f(x_j)) = \prod_{t=1}^{T} Z_t \]

Both smooth approximations of 0/1 loss!

Logistic regression and Boosting

Logistic regression:
- Minimize loss function
  \[ \sum_{j=1}^{N} \ln(1 + \exp(-y_j f(x_j))) \]
- Define
  \[ f(x) = w_0 + \sum_{i} w_i x_i \]
  where features \( x_i \) are predefined
- Weights \( w_i \) are learned in joint optimization

Boosting:
- Minimize loss function
  \[ \sum_{j=1}^{N} \exp(-y_j f(x_j)) \]
- Define
  \[ f(x) = \alpha_i h_t(x) \]
  where \( h_t(x) \) defined dynamically to fit data
  (not a linear classifier)
  via weak classifier
What you need to know about Boosting

- Combine weak classifiers to obtain very strong classifier
  - Weak classifier – slightly better than random on training data
  - Resulting very strong classifier – can eventually provide zero training error
- AdaBoost algorithm
- Boosting v. Logistic Regression
  - Similar loss functions
  - Single optimization (LR) v. Incrementally improving classification (B)
- Most popular application of Boosting:
  - Boosted decision stumps!
  - Very simple to implement, very effective classifier