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!

- **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 $t$:
    - weight each training example by how incorrectly it was classified
    - Learn a hypothesis — $h_t$
    - A strength for this hypothesis — $\alpha_t$

- **Final classifier:**
  - **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
  - $D(j)$ – weight of $j$th training example $(x_i, y_i)$
  - Interpretations:
    - $j$th training example counts as $D(j)$ examples
    - If I were to “resample” data, I would get more samples of “heavier” data points

- Now, in all calculations, whenever used, $j$th training example counts as $D(j)$ “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 $\alpha_t$
  - Update weights:
    $$D_{t+1}(j) = \frac{D_t(j) \exp(-\alpha_t y_j h_t(x_j))}{Z_t}$$
    - Where $Z_t$ is normalizer:
      $$Z_t = \sum_{j=1}^{N} D_t(j) \exp(-\alpha_t y_j h_t(x_j))$$

- Output final classifier:
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:

$$\epsilon_t = \sum_{j=1}^{N} D_t(j) \mathbb{1}[h_t(x^j) \neq y^j]$$

Why choose $\alpha_t$ for hypothesis $h_t$ this way? [Schapire, 1989]

Training error of final classifier is bounded by:

$$\frac{1}{N} \sum_{j=1}^{N} \mathbb{1}[H(x^j) \neq y^j] \leq \frac{1}{N} \sum_{j=1}^{N} \exp(-y^j f(x^j))$$

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? [Schapire, 1989]

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]:

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

You'll prove this in your homework! 😊

Strong, weak classifiers

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

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

$$\frac{1}{N} \sum_{j=1}^{N} I[H(x^j) \neq y^j] \leq \prod_{t=1}^{T} Z_t \leq \exp \left( -2 \sum_{t=1}^{T} (1/2 - \epsilon_t)^2 \right)$$

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

[Schapire, 1989]

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

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:

\[ P(Y = 1 | X) = \frac{1}{1 + \exp(f(x))} \]

And tries to maximize data likelihood:

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

Equivalent to minimizing log loss

\[
\sum_{j=1}^{N} \ln(1 + \exp(-y_j f(x_j)))
\]
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 fn
  $$\sum_{j=1}^{N} \ln(1 + \exp(-y_j f(x^j)))$$
- 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 fn
  $$\sum_{j=1}^{N} \exp(-y_j f(x^j))$$
- Define
  $$f(x) = \sum_{t} \alpha_t h_t(x)$$
  where $h_t(x)$ defined dynamically to fit data
  (not a linear classifier)
- Weights $\alpha_t$ learned incrementally
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