A little history:

AdaBoost came out of the PAC Learning community

- PAC Learning model developed by Valiant ("A Theory of the Learnable" 1989)
- Kearns & Valiant (1988, 1994) posed the question of whether a "weak" PAC learning algorithm, e.g., one that performs slightly better than a random guess, could be used to construct a strong PAC learning algorithm, i.e., one that performs arbitrarily well (yet still poly-time & obeys other technical conditions).

- Freund & Schapire's example: betting strategies for horse racing:
  - An expert gambler may not be able to explain his/her betting strategy, but when presented with data for a specific set of races, s/he can give us a "rule of thumb"
    - Bet on the horse that has won the most recently
    - "..." w/ the most favored odds

  Rules of thumb are not very accurate, but a little better than a random guess.

  Boosting algorithms combine these rules into a single, highly accurate prediction rule.

- Schapire (1989) showed that the answer to Kearns & Valiant's question was "yes". Proof by construction of the 1st boosting algorithm. (Wasn't so practical though!)

- Between '89-'93 a few other boosting alg's were designed but none were very practical.

Let's derive AdaBoost. (But not in the way F1S did it. Turns out AdaBoost is stagewise optimization, discovered by at least 5 groups. We'll do it that way.)

**Standard Classification Task:**

training set: \( \{ (x_i, y_i) \}_{i=1}^{m} \), \( x \in X, \ y \in \{-1, 1\} \) chosen randomly from unknown prob. dist'n.

Want to find \( f : X \rightarrow \mathbb{R} \) such that \( \text{sign}(f(x)) \) agrees with \( y \) as much as possible, \( \forall \ f \in F \).

(We hope that if our chosen \( f \) performs well on the training set, it will perform well on the whole prob. dist'n.)

misclassification error of \( f \) w.r.t. training set = \# of times \( y_i \not= \text{sign}(f(x_i)) \)

= \# of times \( y_i f(x_i) < 0 \)

= \( \sum_{i=1}^{m} \mathbb{1}[y_i f(x_i) < 0] \) \( \mathbb{1} \)-indicator function is 1 if condition holds, 0 otherwise

We want misclassification error to be small, i.e., would like to choose \( f \) to minimize misclassification error. The problem is that in terms of practical optimization, minimizing this quantity is very difficult. It would be much easier if the misclassification error were convex.
Aside: A function is convex if and only if the set of points lying on or above the graph is a convex set. Pick any 2 points above the graph of \( f \). If \( f \) is convex, the whole line of points connecting the 2 points is also above \( f \).

For convex functions all local minima are global minima.

Sums of convex functions are convex, and other nice properties.

Let us perform a trick, namely to use:

\[
H_{x < 0} \leq e^{-x}
\]

Thus,

\[
\text{misclassification error of } \ f \ \text{w.r.t. training set} = \sum_{i=1}^{n} H_{y \cdot f(x_i) < 0}
\]

\[
\leq \sum_{i=1}^{n} e^{-y_i \cdot f(x_i)} =: L(f)
\]

We hope that choosing an \( f \) to yield small values of \( L(f) \) will yield small values of the misclassification error.
Need to choose the form of \( f \). For AdaBoost, \( f \) is a linear combination of "weak classifiers" or "rules of thumb".

\[
f(x) = \sum_{j=1}^{n} \lambda_j h_j(x) \quad \text{where} \quad h_j : X \to \{-1, 1\}
\]

AdaBoost's Objective Function: \( L(\lambda) = \sum_{i=1}^{n} e^{-\sum_{j=1}^{n} \lambda_j y_i h_j(x_i)} \)

Want to minimize this \( \text{w.r.t.} \lambda \).

About the weak classifiers \( h_j \): \( j = 1, \ldots, n \)

- AdaBoost can be used in 2 ways, either to do most of the work, or as a "wrapper" to increase the accuracy of an already accurate base learning algorithm (e.g., neural networks).

In the second case, the \( h_j \)'s are classifiers that come from the base learning algorithm.

\( n \) can be huge... or even infinite.

We assume the \( h_j \)'s are given to us, so we just need to find \( \lambda \).

Aside: Derivation of Logistic Regression's Objective:

If instead of \( \odot \) we had used

\[
\| z \|_2^2 = \log(1 + e^z)
\]

then we would derive the objective for the classification algorithm called \underline{Logistic Regression}.

\[
L_{\text{LogReg}}(\lambda) = \sum_{i=1}^{n} \log(1 + e^{-\sum_{j=1}^{n} \lambda_j y_i h_j(x_i)})
\]

Aside: Part of SVM's Objective:

If instead of \( \odot \) we had used

\[
\| z \|_2^2 = \begin{cases} 1 - z & z \leq 1 \\ 0 & z \geq 1 \end{cases}
\]

then we would have derived part of the objective for SVM's.

(We'll talk about this another day.)
Since $L(\lambda)$ is convex in $\lambda$, we can use simple techniques to minimize $L(\lambda)$ w.r.t. $\lambda$ in $\mathbb{R}^n$. We'll use "coordinate descent":

$$\text{for } t = 1, \ldots, T$$

**Step 1:** Find the steepest "direction" $j_t$ (i.e. choose a weak classifier)

**Step 2:** move along that direction until $L(\lambda)$ is minimized (i.e. choose $\alpha_t$ to minimize $L(\lambda + \alpha_t e_{j_t})$ where $e_{j_t} = \begin{pmatrix} 0 \cr \vdots \cr 1 \cr \vdots \cr 0 \end{pmatrix}$)

Formally,

- **Objective:** $L(\lambda) = \sum_{i=1}^{m} e^{-\lambda_i y_i h_s(x_i)}$

- **Directional Derivative:**

  $$\frac{dL(\lambda + \alpha e_{j})}{d\alpha} \bigg|_{\alpha = 0} = \sum_{i=1}^{m} y_i h_{s_i}(x_i) e^{-\lambda_i y_i h_{s_i}(x_i) + \alpha y_i h_{s_i}(x_i)}$$

- **Step 1:**

  $$j_t = \arg \max_{j} \left\{ \frac{dL(\lambda + \alpha e_{j})}{d\alpha} \bigg|_{\alpha = 0} \right\} = \arg \max_{j} \sum_{i=1}^{m} y_i h_{s_i}(x_i) e^{-\lambda_i y_i h_{s_i}(x_i) + \alpha y_i h_{s_i}(x_i)}$$

  $$= \arg \max_{j} \sum_{i=1}^{m} y_i h_{s_i}(x_i) d_i \quad \text{where } d_i = \frac{e^{-\lambda_i y_i h_{s_i}(x_i)}}{Z}$$

- **Step 2:**

  $$0 = \frac{dL(\lambda + \alpha e_{j})}{d\alpha} \bigg|_{\alpha = 0} = \sum_{i=1}^{m} y_i h_{s_i}(x_i) e^{-\lambda_i y_i h_{s_i}(x_i) + \alpha y_i h_{s_i}(x_i)}$$

  $$= \sum_{i: y_i h_{s_i}(x_i) = 1} d_i e^{-\alpha} + \sum_{i: y_i h_{s_i}(x_i) = -1} -1 d_i e^{\alpha}$$

  $$= (1 - d) e^{-\alpha} - d e^{\alpha} \quad \text{where } d = \sum_{i: y_i h_{s_i}(x_i) = -1} d_i$$

  $$(1-d) e^{-\alpha} = d e^{\alpha} \quad \Rightarrow \alpha = \frac{1}{2} \ln \frac{1-d}{d}$$
Pseudocode for AdaBoost:

Given \( \mathbf{x}_i \) \( \mathbf{y}_i \) \( i=1...m \), \( \mathbf{h}_j \) \( j=1...n \), \( T \), \( \lambda_{ij} = 0 \) \( \forall j \)

For \( t=1...T \)

\( d_{t,i} = \frac{1}{m} \mathbf{y}_i \)

\( j_t = \arg\max \sum_{i} \mathbf{y}_i \mathbf{h}_j (\mathbf{x}_i) \mathbf{d}_i \) "train weak learner" using dist \( \mathbf{d} \)

\( \mathbf{d}_{e_t} = \sum_{\mathbf{y}_i \mathbf{h}_{j_t} (\mathbf{x}_i) = -1} \mathbf{d}_{t,i} \) error of weak classifier

\( \alpha_t = \frac{1}{2} \ln \left( \frac{1-d_{e_t}}{d_{e_t}} \right) \)

\( \lambda_{t+1} = \lambda_t + \alpha_t \mathbf{e}_t \) add weak classifier's contribution

\( \mathbf{d}_{t+1,i} = \mathbf{d}_{t,i} \cdot \left( e^{-\alpha_t} \text{ if } \mathbf{h}_{j_t} (\mathbf{x}_i) = \mathbf{y}_i \right) / Z_t \) write weights for next round in terms of weights for this round

end

To evaluate \( f \):

\( f(\mathbf{x}) = \sum_{j} \lambda_{T,j} \mathbf{h}_j (\mathbf{x}) \).

Notes: This is not the usual way AdaBoost is introduced or derived. Usually one thinks of the \( d_{t,i} \)'s as fundamental, viewing them as "weights" on each training example. (Here, \( d_{t,i} \) 's fall naturally out of the derivation.)

At each \( t \), give more credit to classifiers that did well w.r.t. the weighted training examples.