CS6375: Machine Learning

Gautam Kunapuli

Ensemble Methods: Boosting

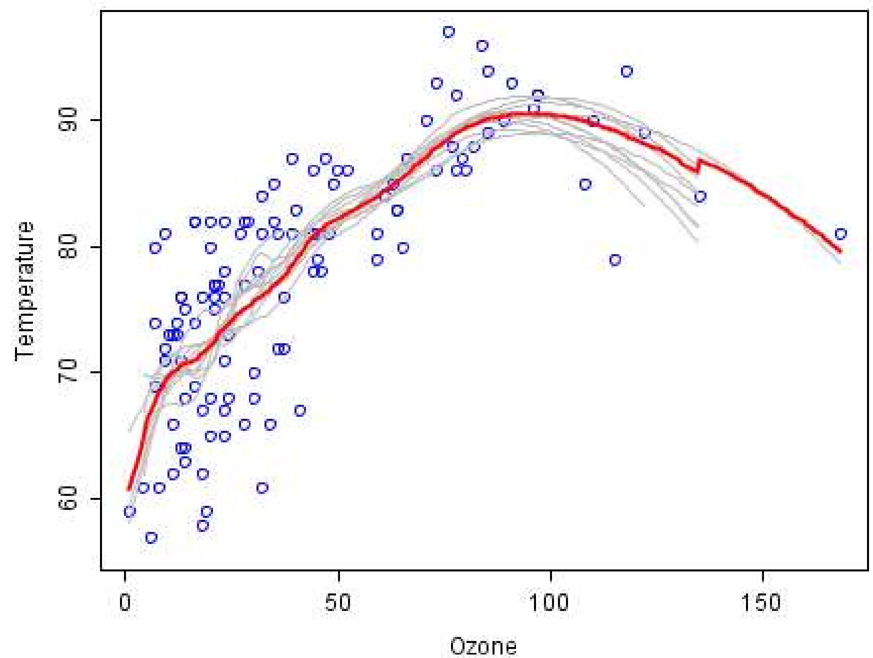
THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

Ensemble Methods

Idea: Train models on different data set samples to reduce the model variance and/or bias
Problem: Only one training set; where do multiple models come from?
Solution: Take a **single** learning algorithm and generate multiple variations called **ensembles**

Bagging

- Uses **bootstrap sampling** to learn many (strong) models, whose predictions can be **aggregated**
- bagging reduces **variance** (and in practice, maybe increases bias slightly)
- bagging learns with **bootstrap samples** of the **same size as the original data set**
    - with decision trees, typically learns full trees
    - computational complexity is higher
- each model is learned independently of other models
    - insight from one model does not influence the learning of the next model



Boosting

- uses weak learners that have high bias
    - e.g., decision stumps (decision trees with depth 1)
- boosting reduces both **bias** and **variance**
- **iterative algorithm** that increases weights on hard examples
    - insight from previous iterations guides learning

AdaBoost: Adaptive Boosting

Basic idea behind Boosting: examples are given **weights:** at each iteration, a new hypothesis is learned and examples are **reweighted** to enable focus on examples that **most recently learned classifier** got wrong

Basic Algorithm for Boosting:
Initialize: set all examples to have equal weights
for each $t = 1, \dots, T$,
  Learn a hypothesis h_t from weighted examples
  Decrease weights of examples h_t classifies **correctly**
  Calculate α_t , the weight of the current weak learner, h_t
return $h(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$

Weighted examples: Base (weak) learner must focus on correctly classifying the most highly weighted examples while strongly avoiding over-fitting.

Weighted Hypotheses: During testing, each of the *T* hypotheses get a weighted vote proportional to their accuracy on the training data.

- Originally developed by computational learning theorists to guarantee performance improvements on fitting training data for a **weak learner**
  - a **weak learner** only needs to generate a hypothesis with a training accuracy **greater than 0.5** (Schapire, 1990);
  - **often, weak learners are only slightly better than random**
- practical algorithm, **AdaBoost**, for building ensembles that **empirically improves generalization** (Freund & Schapire, 1996).

AdaBoost: Weak Learners

Basic Algorithm for Boosting:
Initialize: set all examples to have equal weights
for each $t = 1, \ldots, T$,
Learn a hypothesis h_t from weighted examples
Decrease weights of examples h_t classifies **correctly**
Calculate α_t , the weight of the current weak learner, h_t
return $h(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$

a **weak learner** is typically easy to train and is simple, that is, of low complexity

- **high bias**, low variance
- **boosting** takes a weak learner and converts it to a strong learner
- just has to achieve an **accuracy slightly better than random guessing**, that is, error $\epsilon < 0.5$
- a weak learner achieves **accuracy-to-error ratio:**

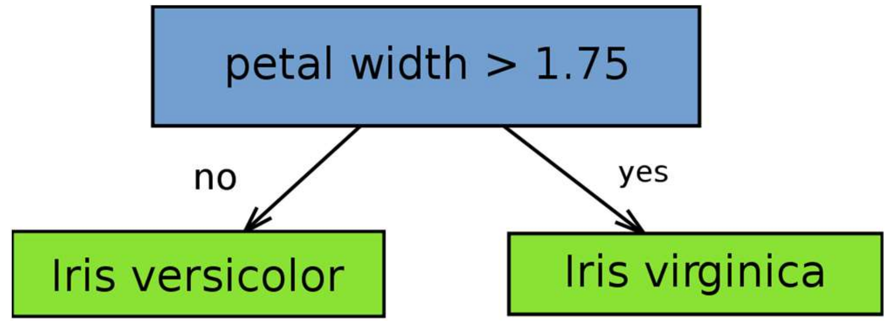
$$\frac{1-\epsilon}{\epsilon} > 1$$

- we can make the weight of a weak learner during boosting depend on its accuracy

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon}{\epsilon}\right) > 0$$

- stronger learners have higher weights

Decision stumps are classical and often-used weak learners; Naïve Bayes, Logistic Regression also return probability of classification



Weak learners commonly used in practice:

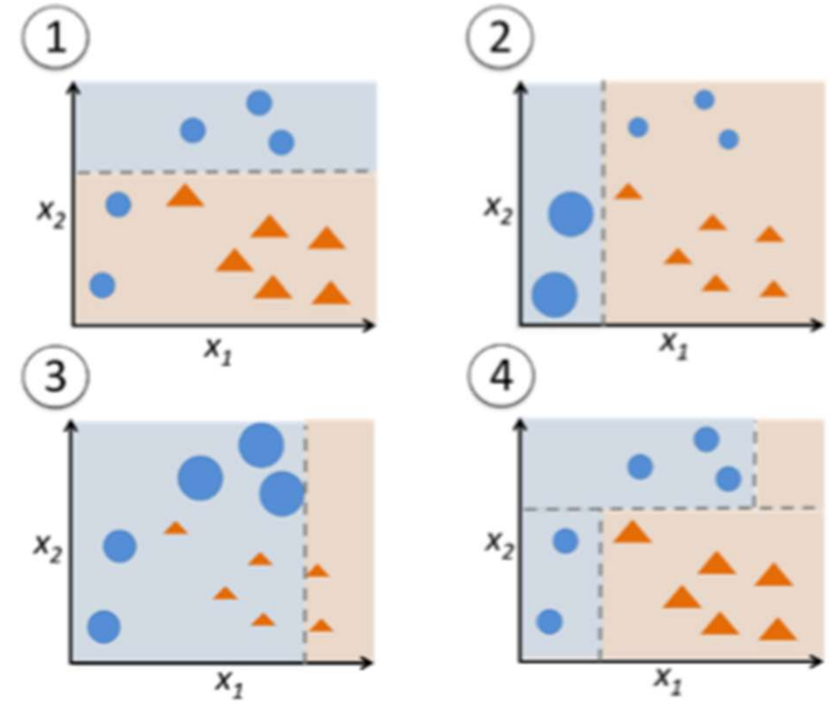
- Decision stumps (axis parallel splits)
- Shallow decision trees
- Multi-layer neural networks
- Radial basis function networks

Note: There is nothing inherently weak about weak learners – we just think of them this way. In fact, **any learning algorithm** can be used as a weak learner.

AdaBoost: Training Set Distributions

Basic idea behind AdaBoost: maintain a **distribution** over **examples** that reflects their ``hardness" of classification; a new hypothesis is learned and the distribution is **updated** to enable focus on examples that **most recently learned classifier** got wrong

Basic Algorithm for Boosting:
Initialize: set all examples to have equal weights
for each $t = 1, \ldots, T$,
Learn a hypothesis h_t from weighted examples
Decrease weights of examples h_t classifies **correctly**
Calculate α_t , the weight of the current weak learner, h_t
return $h(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$



weights on examples can be converted to a **distribution** that reflects their **"hardness of classification"**

thus, each training example $(\boldsymbol{x}_i, y_i)$ has weights $D(i)$, with

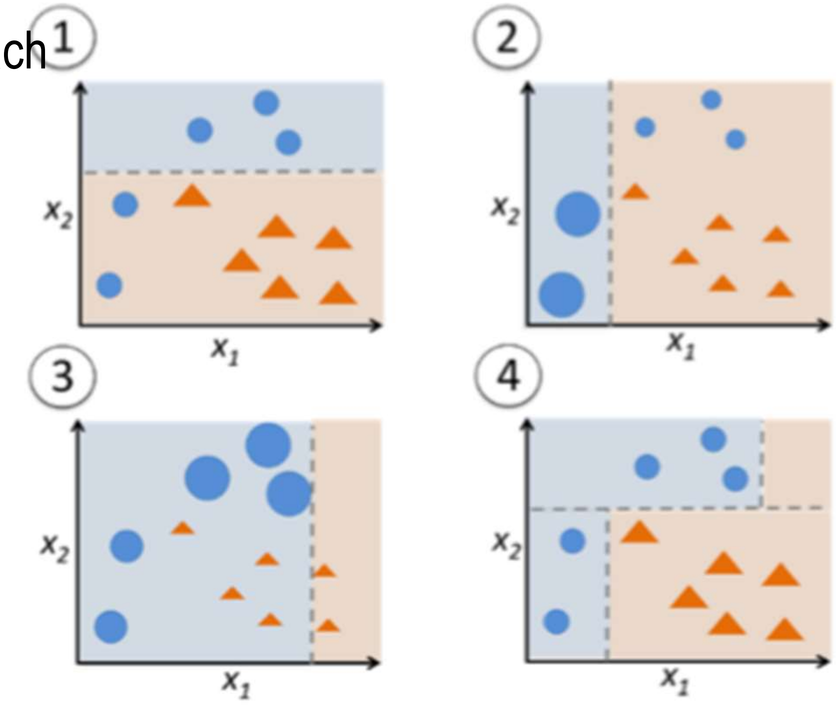
$$\sum_{i=1}^{n} D(i) = 1$$

- most misclassified points get highest weights
- this ensures that the algorithm can focus on training examples with higher weights

For boosting, **we need a weak learner** that can
- handle weighted **examples/distributions**
- alternately, **sample** training examples **according to the distribution** (more on next slide)
  - contrast this with bagging!

Learning with Weighted Training Examples

In a **weighted dataset** we have a weight associated with each training example:
- $D(i)$ is the weight of i-th training example $(\boldsymbol{x}_i, y_i)$
- i-th training example counts as $D(i)$ training examples; if we "resampled" data, we would get more samples of "heavier" data points
- Now, in all calculations, the i-th **training example counts as $D(i)$ "examples"**



Example 1: in Maximum Likelihood Estimation
Unweighted data: $\#(y = c) \ = \sum_i \mathbf{1}(y = c)$
Weighted data: $\#(y = c) \ = \sum_i D(i) \, \mathbf{1}(y = c)$

Example 2: in Decision Stumps:
- first, when computing $H(Y) = -\sum_y P(Y = c) \mathbf{log_2} \, \mathrm{P}(Y = c)$ for a class c, use the weights; for instance, in the binary classification case:

$$P(y = 0) = \frac{\#(y=0)}{\#(y=0)+\#(y=1)} \text{ (unweighted)} \qquad P(y = 0) = \frac{\sum w(y=0)}{\sum w(y=0) + \sum w(y=1)} \text{ (weighted)}$$

- second, when computing $H(Y|X) = -\sum_x P(X = x) \sum_y P(Y = c \mid X = x) \ \log_2 (Y = c \mid X = x)$
- alternately, since decision stumps are easy to compute, simply compute **all possible decision stumps and select the one with the smallest weighted error** as the best weak learners

AdaBoost: Full Algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$

best decision stump is the one that **minimizes** the **weighted training error**

$$\epsilon_t = \frac{1}{\sum_{i=1}^{n} D_t(i)} \sum_{i=1}^{n} D_t(i) \cdot [\![h_t(\boldsymbol{x}_i) \neq y_i]\!]$$

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

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

re-normalize the weights into a distribution

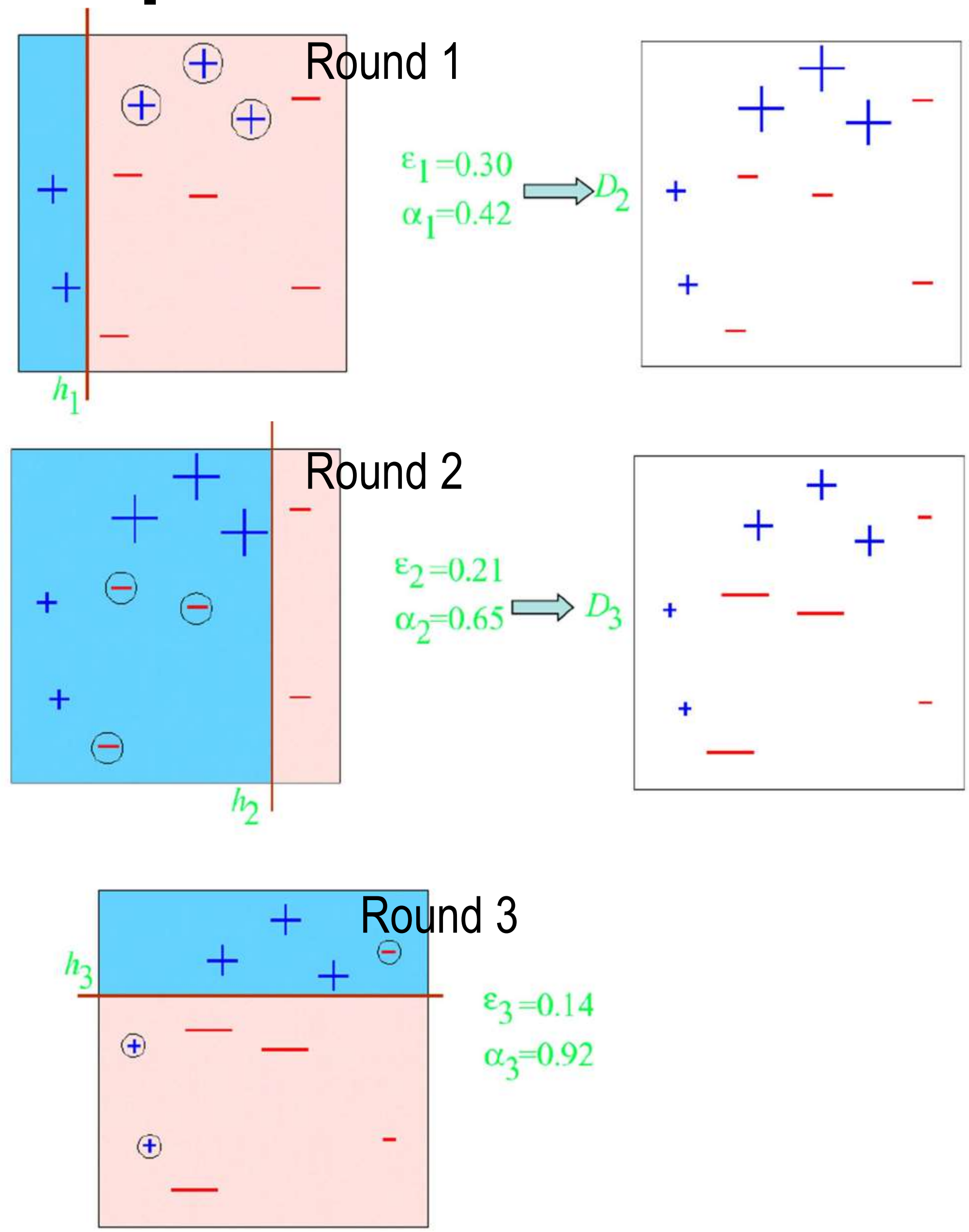
$$Z_t = \sum_{i=1}^{n} D_t(i) \cdot \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

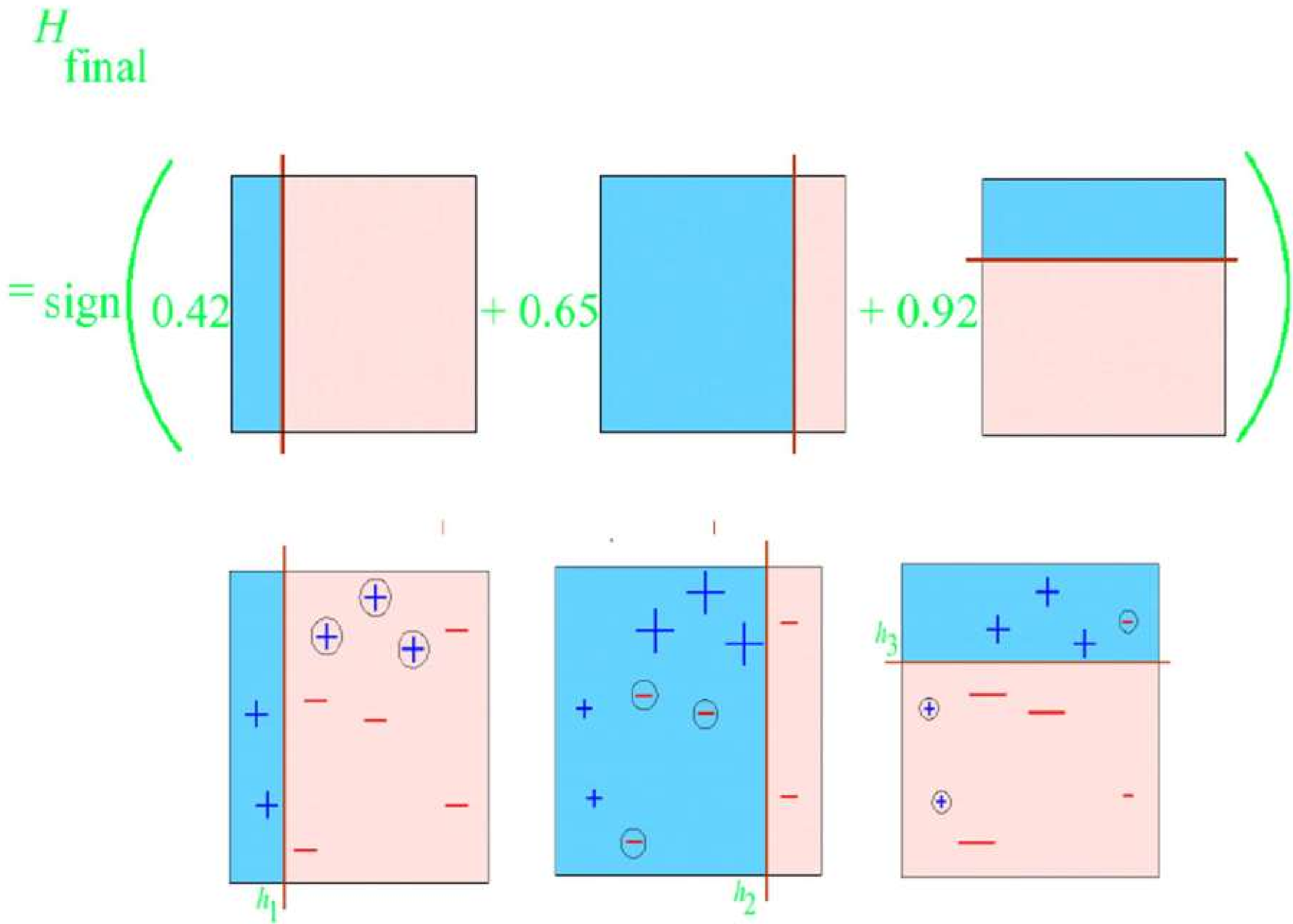
Output the final hypothesis:

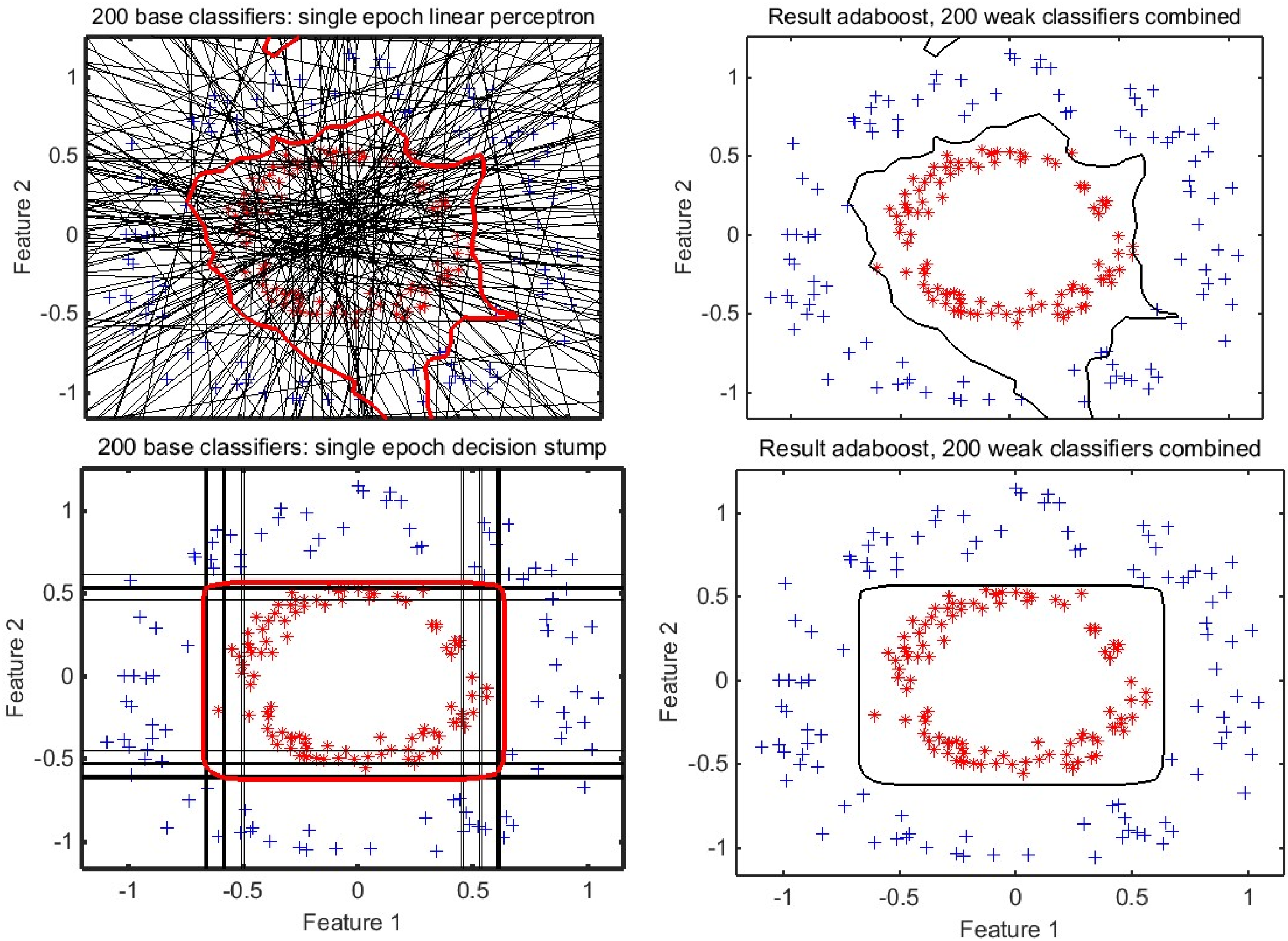
$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

AdaBoost: Example



AdaBoost: Example




200 base classifiers: single epoch linear perceptron
Result adaboost, 200 weak classifiers combined
200 base classifiers: single epoch decision stump
Result adaboost, 200 weak classifiers combined
Feature 2
Feature 1
1
0.5
0
-0.5
-1

Boosting Optimizes Exponential Loss

In 2000, Friedman et al. interpreted AdaBoost as stage-wise forward additive model that actually minimizes the exponential loss function, $L(y, f(\boldsymbol{x})) = E[\![e^{-yf(x)}]\!]$.

First, note that the overall model at the t-th iteration is:

$$f_t(x) = f_{t-1}(x) + \alpha_t h_t(x)$$

Split the exponential loss into positive and negative components

$$E[\![e^{-yf(x)}]\!] = \sum_{i=1}^{n} \exp(-y_i[f_{t-1}(\boldsymbol{x}_i) + \alpha_t h_t(\boldsymbol{x}_i)])$$

$$= \sum_{i=1}^{n} \underbrace{\exp(-y_i f_{t-1}(\boldsymbol{x}_i))}_{D_t(i)} \exp(-y_i \alpha_t h_t(\boldsymbol{x}_i))$$

$$= \sum_{y_i h_t(\boldsymbol{x}_i)=1} D_t(i) \exp(-\alpha_t) + \sum_{y_i h_t(\boldsymbol{x}_i)=-1} D_t(i) \exp(\alpha_t)$$

correctly classified *misclassified*

Take the gradient and set to zero

$$\frac{d}{d\alpha_t} E[\![e^{-yf(x)}]\!] = -\exp(-\alpha_t) \sum_{y_i h_t(\boldsymbol{x}_i)=1} D_t(i) + \exp(\alpha_t) \sum_{y_i h_t(\boldsymbol{x}_i)=-1} D_t(i) = 0$$

Take the gradient and set to zero

$$\exp(2\alpha_t) = \frac{\sum_{y_i h_t(\boldsymbol{x}_i)=1} D_t(i)}{\sum_{y_i h_t(\boldsymbol{x}_i)=-1} D_t(i)}$$

Take the log on both sides

$$\alpha_t = \frac{1}{2} \log \frac{\sum_{y_i h_t(\boldsymbol{x}_i)=1} D_t(i)}{\sum_{y_i h_t(\boldsymbol{x}_i)=-1} D_t(i)}$$

Boosting Optimizes Exponential Loss

Boosting optimizes **exponential loss**

$$\alpha_t = \frac{1}{2}\log\frac{\sum_{y_i h_t(x_i)=1} D_t(i)}{\sum_{y_i h_t(x_i)=-1} D_t(i)}$$

Logistic regression optimizes **logistic loss**

$$f(x) = \frac{1}{2}\log\frac{P(y=1)}{P(y=-1)}$$

Logistic regression:

- Minimize log loss

$$\sum_{i=1}^{m} \ln(1+\exp(-y_i f(x_i)))$$

- Define

$$f(x) = \sum_j w_j x_j$$

where x_j predefined features

(linear classifier)

- Jointly optimize over all weights $w_0, w_1, w_2 \ldots$

Boosting:

- Minimize exp loss

$$\sum_{i=1}^{m} \exp(-y_i f(x_i))$$

- Define

$$f(x) = \sum_t \alpha_t h_t(x)$$

where $h_t(x)$ defined dynamically to fit data

(not a linear classifier)

- Weights α_t learned per iterati t incrementally

Boosting Increases The Margin

*the **margin of a single data point** is defined to be the **distance** from the data point **to the decision boundary***

We can write the **combined classifier** in a more useful form by dividing the predictions by the "total number of votes":

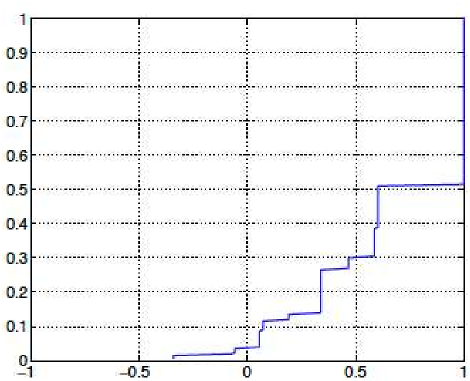
$$h_{t+1}(\boldsymbol{x}_i) = \frac{\alpha_1 h_1(\boldsymbol{x}_i) + \cdots + \alpha_t h_t(\boldsymbol{x}_i)}{\alpha_1 + \cdots + \alpha_t}$$

• This allows us to define a clear notion of "**voting margin**" that the combined classifier achieves for each training example:

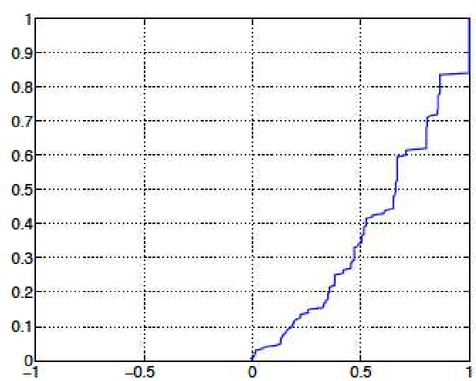
$$\text{margin}(\boldsymbol{x}_i) = y_i h_{t+1}(\boldsymbol{x}_i)$$

• The margin lies in [−1, 1] and is negative for all misclassified examples.

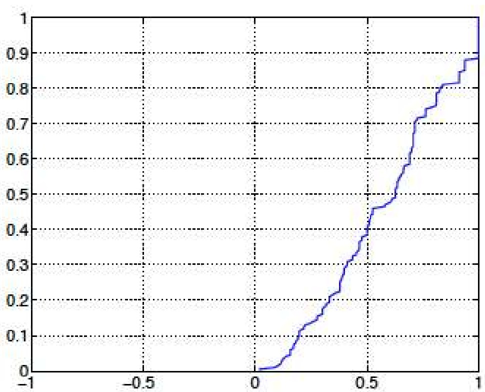
• Successive boosting iterations **improve the majority vote** or **margin** for the training examples



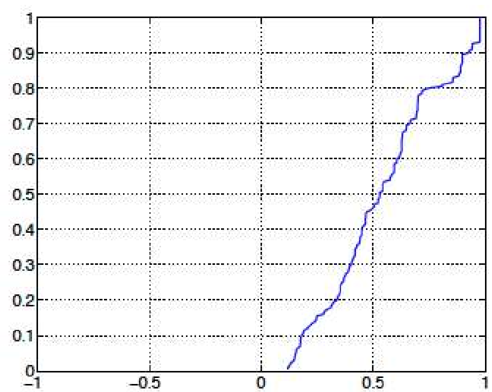
4 iterations



10 iterations



20 iterations



50 iterations

Cumulative distributions of margin values

Boosting: Pros and Cons

Pros

- fast
- simple and easy to program
- no parameters to tune (except T)
- flexible — can combine with any learning algorithm
- no prior knowledge needed about weak learner
- provably effective, provided can consistently find rough rules of thumb
  - shift in mind set — goal now is merely to find classifiers barely better than random guessing
- versatile
- can use with data that is textual, numeric, discrete, etc.
- has been extended to learning problems well beyond binary classification

Cons

- performance of AdaBoost depends on data and weak learner
- consistent with theory, AdaBoost can fail if
- weak classifiers too complex (! overfitting)
- weak classifiers too weak (! underfitting)
- empirically, AdaBoost seems especially susceptible to uniform noise

Good ☺ : Can identify outliers since focuses on examples that are hard to categorize

Bad ☹ : Too many outliers can degrade classification performance dramatically increase time to convergence