Xi'an Jiaotong-Liverpool University
西交利物浦大学

INT305 Machine Learning

Lecture 8

Bagging & Boosting

Jimin Xiao

Department Intelligence Science

Jimin.xiao@xjtlu.edu.cn

Today

- Today we will introduce ensembling methods that combine multiple models and can perform better than the individual members.
  - We've seen many individual models (KNN, linear models, neural networks, decision trees)
- We will see bagging:
  - Train models independently on random "resamples" of the training data.
- And boosting:
  - Train models sequentially, each time focusing on training examples that the previous ones got wrong.
- Bagging and boosting serve slightly different purposes. Let's briefly review bias/variance decomposition.

Bias/Variance Decomposition

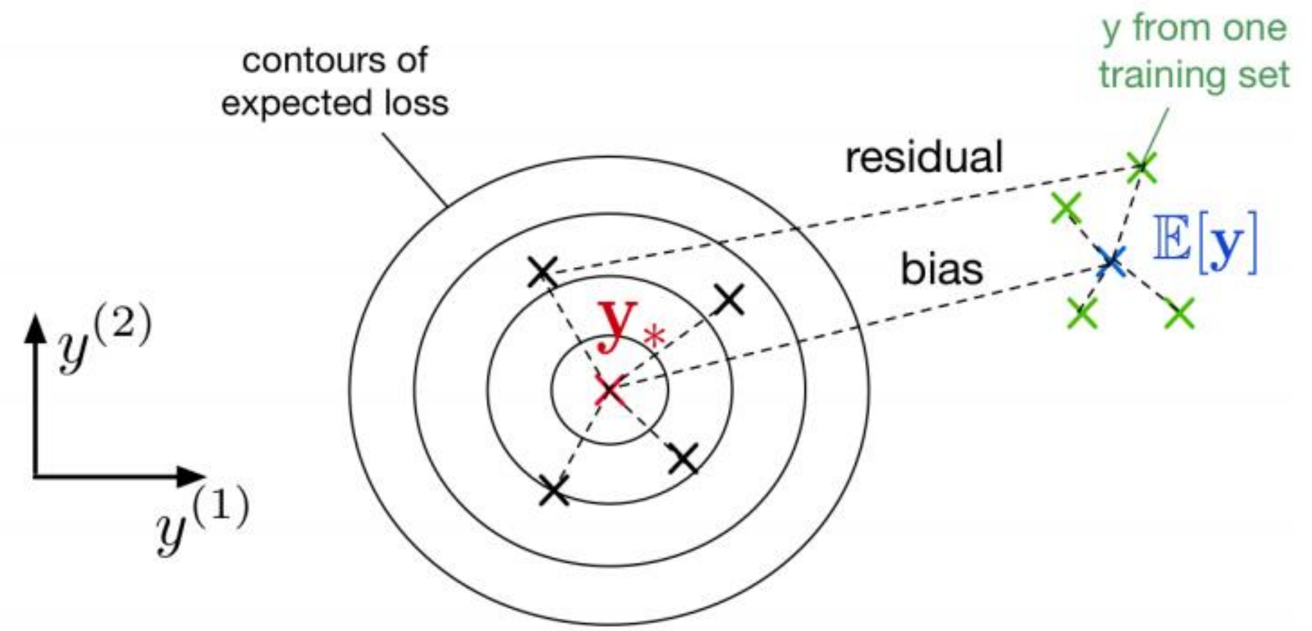
- Recall, we treat predictions y at a query $\mathbf{x}$ as a random variable (where the randomness comes from the choice of dataset), $y_\star$ is the optimal deterministic prediction, t is a random target sampled from the true conditional $p(t|\mathbf{x})$.

$$\mathbb{E}[(y-t)^2] = \underbrace{(y_\star - \mathbb{E}[y])^2}_{\text{bias}} + \underbrace{\mathrm{Var}(y)}_{\text{variance}} + \underbrace{\mathrm{Var}(t)}_{\text{Bayes error}}$$

- Bias/variance decomposes the expected loss into three terms:
  - bias: how wrong the expected prediction is (corresponds to underfitting)
  - variance: the amount of variability in the predictions (corresponds to overfitting)
  - Bayes error: the inherent unpredictability of the targets
- Even though this analysis only applies to squared error, we often loosely use "bias" and "variance" as synonyms for "underfitting" and "overfitting".

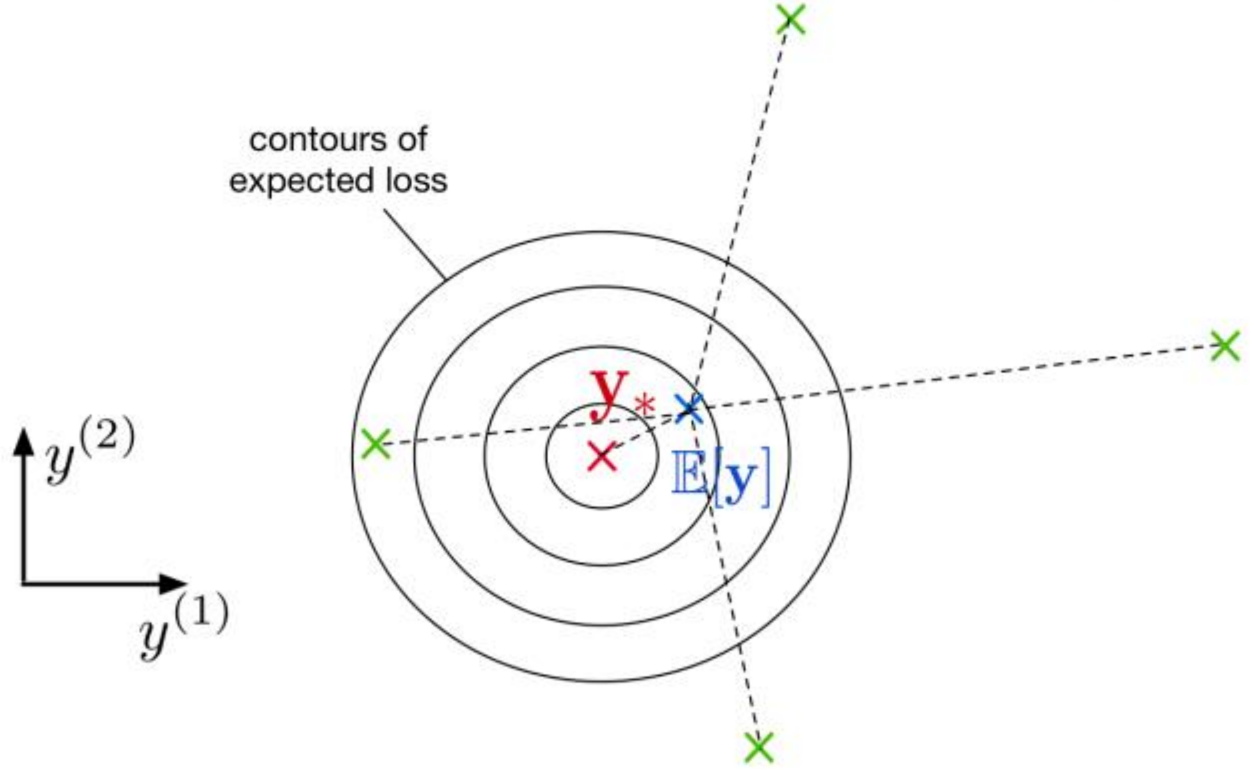
Bias/Variance Decomposition: Another Visualization

- We can visualize this decomposition in output space, where the axes correspond to predictions on the test examples.
- If we have an overly simple model (e.g. KNN with large k), it might have
  - high bias (because it cannot capture the structure in the data)
  - low variance (because there's enough data to get stable estimates)



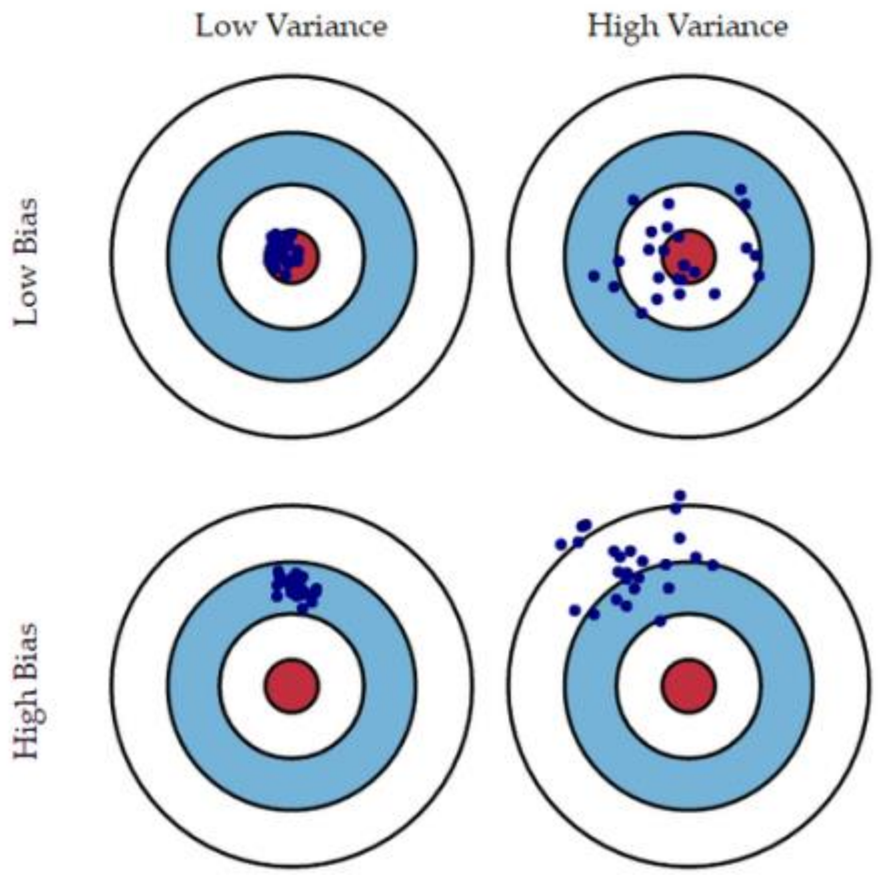
Bias/Variance Decomposition: Another Visualization

- If you have an overly complex model (e.g. KNN with $k = 1$), it might have
  - low bias (since it learns all the relevant structure)
  - high variance (it fits the quirks of the data you happened to sample)



Bias/Variance Decomposition: Another Visualization

- The following graphic summarizes the previous two slides:



Bagging: Motivation

- Suppose we could somehow sample m independent training sets from p_{sample}.
- We could then compute the prediction y_i based on each one, and take the average $y = \frac{1}{m}\sum_{i=1}^{m} y_i$.
- How does this affect the three terms of the expected loss?
  - **Bayes error: unchanged**, since we have no control over it
  - **Bias: unchanged**, since the averaged prediction has the same expectation

$$\mathbb{E}[y] = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} y_i\right] = \mathbb{E}[y_i]$$

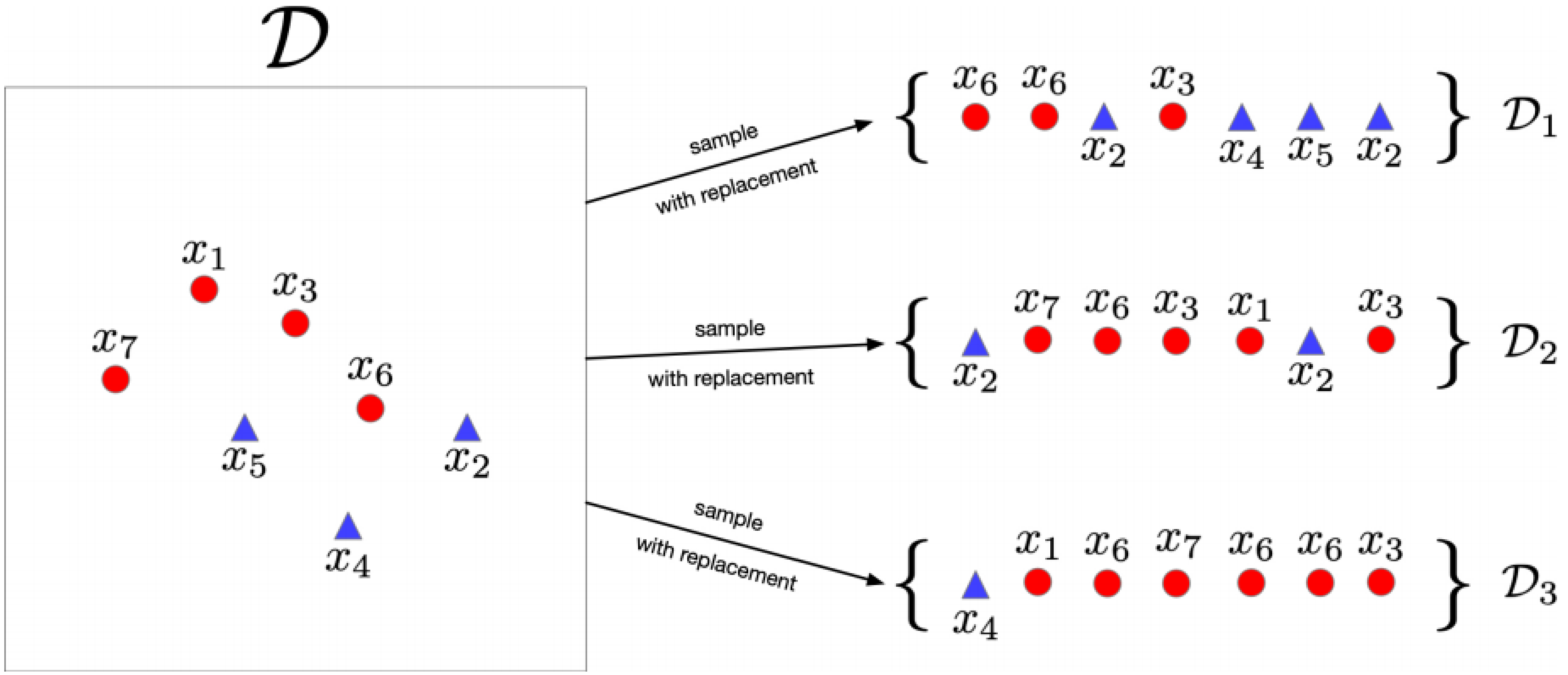
  - **Variance: reduced**, since we're averaging over independent samples

$$\mathrm{Var}[y] = \mathrm{Var}\left[\frac{1}{m}\sum_{i=1}^{m} y_i\right] = \frac{1}{m^2}\sum_{i=1}^{m} \mathrm{Var}[y_i] = \frac{1}{m}\mathrm{Var}[y_i].$$

Bagging: The Idea

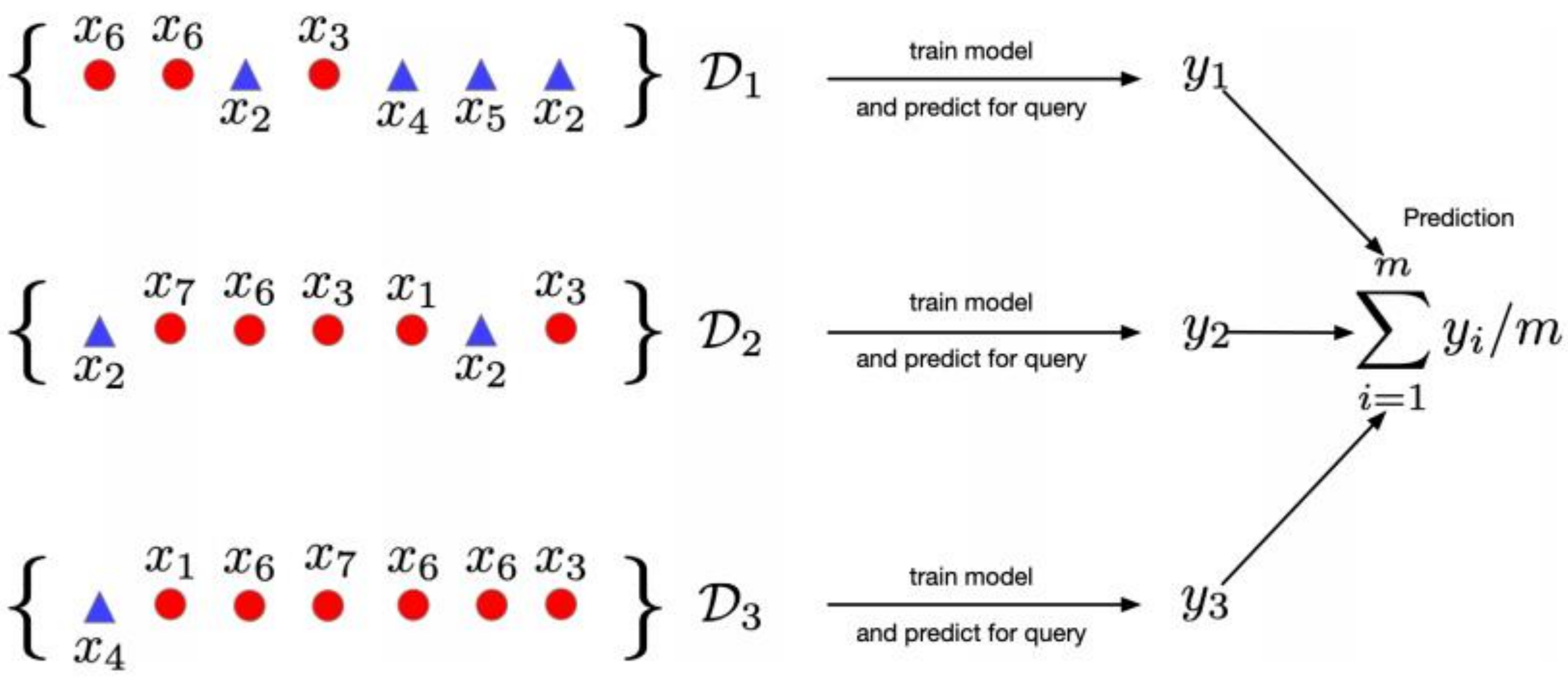
- In practice, the sampling distribution p_{sample} is often finite or expensive to sample from.
- So training separate models on independently sampled datasets is very wasteful of data!
  - Why not train a single model on the union of all sampled datasets?
- Solution: given training set $\mathcal{D}$, use the empirical distribution $p_{\mathcal{D}}$ as a proxy for p_{sample}. This is called bootstrap aggregation, or bagging .
  - Take a single dataset $\mathcal{D}$ with n examples.
  - Generate m new datasets ("resamples" or "bootstrap samples"), each by sampling n training examples from $\mathcal{D}$, with replacement.
  - Average the predictions of models trained on each of these datasets.
- The bootstrap is one of the most important ideas in all of statistics!
  - Intuition: As $|\mathcal{D}| \to \infty$, we have $p_{\mathcal{D}} \to p_{\text{sample}}$.

Bagging



in this example $n = 7$, $m = 3$

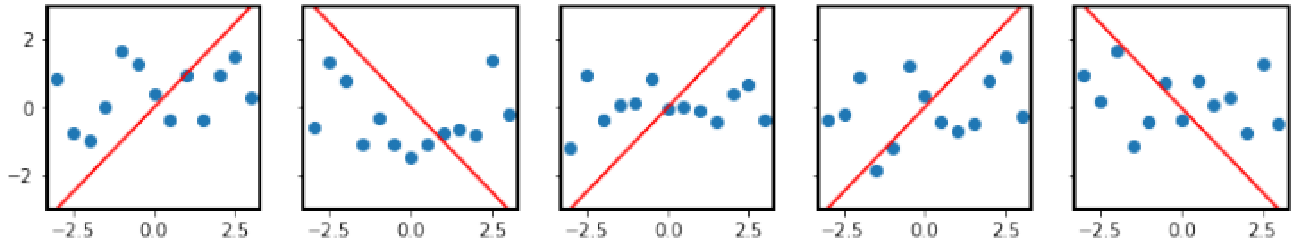
Bagging



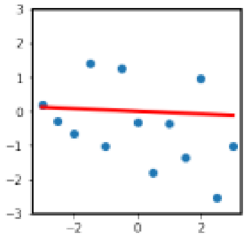
predicting on a query point x

Bagging: Effect on Hypothesis Space

- We saw that in case of squared error, bagging does not affect bias.
- But it can change the hypothesis space / inductive bias.
- Illustrative example:
  - $x \sim \mathcal{U}(-3, 3)$, $t \sim \mathcal{N}(0, 1)$
  - $\mathcal{H} = \{wx \mid w \in \{-1, 1\}\}$
  - Sampled datasets & fitted hypotheses:



  - Ensembled hypotheses (mean over 1000 samples):



  - The ensembled hypothesis is not in the original hypothesis space!
- This effect is most pronounced when combining classifiers ...

Bagging for Binary Classification

- If our classifiers output real-valued probabilities, $z_i \in [0, 1]$, then we can average the predictions before thresholding:

$$y_{\text{bagged}} = \mathbb{I}(z_{\text{bagged}} > 0.5) = \mathbb{I}\left(\sum_{i=1}^{m} \frac{z_i}{m} > 0.5\right)$$

- If our classifiers output binary decisions, $y_i \in \{0, 1\}$, we can still average the predictions before thresholding:

$$y_{\text{bagged}} = \mathbb{I}\left(\sum_{i=1}^{m} \frac{y_i}{m} > 0.5\right)$$

  This is the same as taking a majority vote.

- A bagged classifier can be stronger than the average underlying model.
  - E.g., individual accuracy on “Who Wants to be a Millionaire” is only so-so, but “Ask the Audience” is quite effective.

Bagging: Effect of Correlation

- Problem: the datasets are not independent, so we don't get the $1/m$ variance reduction.
  - Possible to show that if the sampled predictions have variance σ^2 and correlation ρ, then

$$\mathrm{Var}\left(\frac{1}{m}\sum_{i=1}^{m} y_i\right) = \frac{1}{m}(1-\rho)\sigma^2 + \rho\sigma^2.$$

Random Forests

- Random forests = bagged decision trees, with one extra trick to decorrelate the predictions
  - When choosing each node of the decision tree, choose a random set of d input features, and only consider splits on those features
- Random forests are probably the best black-box machine learning algorithm — they often work well with no tuning whatsoever.
  - one of the most widely used algorithms in Kaggle competitions

Bagging Summary

- Bagging reduces overfitting by averaging predictions.
- Used in most competition winners
  - Even if a single model is great, a small ensemble usually helps.
- Limitations:
  - Does not reduce bias in case of squared error.
  - There is still correlation between classifiers.
    - Random forest solution: Add more randomness.
  - Naive mixture (all members weighted equally).
    - If members are very different (e.g., different algorithms, different data sources, etc.), we can often obtain better results by using a principled approach to weighted ensembling.
- Boosting, up next, can be viewed as an approach to weighted ensembling that strongly decorrelates ensemble members.

Boosting

- Boosting
  - Train classifiers sequentially, each time focusing on training examples that the previous ones got wrong.
  - The shifting focus strongly decorrelates their predictions.
- To focus on specific examples, boosting uses a weighted training set.

Weighted Training set

- The misclassification rate $\frac{1}{N}\sum_{n=1}^{N}\mathbb{I}[h(x^{(n)}) \neq t^{(n)}]$ weights each training example equally.
- Key idea: we can learn a classifier using different costs (aka weights) for examples.
  - Classifier "tries harder" on examples with higher cost
- Change cost function:

$$\sum_{n=1}^{N}\frac{1}{N}\mathbb{I}[h(x^{(n)}) \neq t^{(n)}] \quad \text{becomes} \quad \sum_{n=1}^{N} w^{(n)}\mathbb{I}[h(x^{(n)}) \neq t^{(n)}]$$

- Usually require each $w^{(n)} > 0$ and $\sum_{n=1}^{N} w^{(n)} = 1$

AdaBoost (Adaptive Boosting)

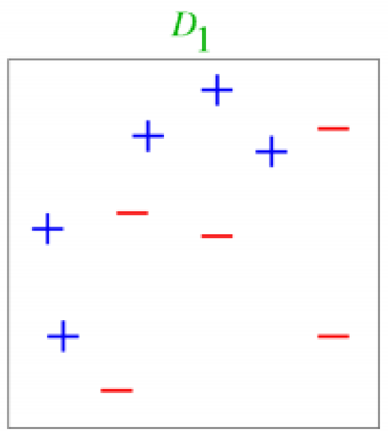
- We can now describe the AdaBoost algorithm.
- Given a base classifier, the key steps of AdaBoost are:
  1. At each iteration, re-weight the training samples by assigning larger weights to samples (i.e., data points) that were classified incorrectly.
  2. Train a new base classifier based on the re-weighted samples.
  3. Add it to the ensemble of classifiers with an appropriate weight.
  4. Repeat the process many times.
- Requirements for base classifier:
  - Needs to minimize weighted error.
  - Ensemble may get very large, so base classifier must be fast. It turns out that any so-called weak learner/classifier suffices.
- Individually, weak learners may have high bias (underfit). By making each classifier focus on previous mistakes, AdaBoost reduces bias.

Weak Learner/Classifier

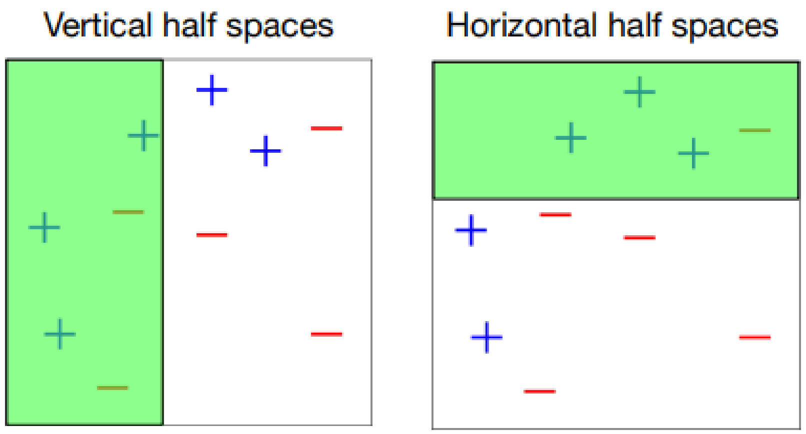
- (Informal) Weak learner is a learning algorithm that outputs a hypothesis (e.g., a classifier) that performs slightly better than chance, e.g., it predicts the correct label with probability 0.51 in binary label case.
- We are interested in weak learners that are *computationally* efficient.
  - Decision trees
  - Even simpler: Decision Stump: A decision tree with a single split

[Formal definition of weak learnability has quantifies such as "for any distribution over data" and the requirement that its guarantee holds only probabilistically.]

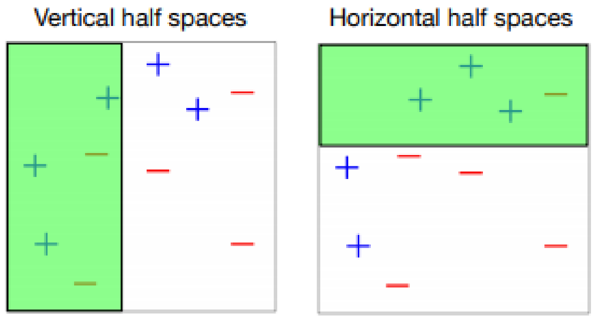
Weak Classifiers



These weak classifiers, which are decision stumps, consist of the set of horizontal and vertical half spaces.



Weak Classifiers



- A *single* weak classifier is not capable of making the training error small
- But if can guarantee that it performs slightly better than chance, i.e., the weighted error of classifier h according to the given weights $\mathbf{w} = (w_1, \ldots, w_N)$ is at most $\frac{1}{2} - \gamma$ for some $\gamma > 0$, using it with AdaBoost gives us a universal function approximator!
- Last lecture we used information gain as the splitting criterion. When using decision stumps with AdaBoost we often use a "GINI Impurity", which (roughly speaking) picks the split that directly minimizes error.
- Now let's see how AdaBoost combines a set of weak classifiers in order to make a better ensemble of classifiers...

Notation in this lecture

- Input: Data $\mathcal{D}_N = \{\mathbf{x}^{(n)}, t^{(n)}\}_{n=1}^{N}$ where $t^{(n)} \in \{-1, +1\}$
  - This is different from previous lectures where we had $t^{(n)} \in \{0, +1\}$
  - It is for notational convenience, otw equivalent.
- A classifier or hypothesis $h : \mathbf{x} \to \{-1, +1\}$
- 0-1 loss: $\mathbb{I}[h(x^{(n)}) \neq t^{(n)}] = \frac{1}{2}(1 - h(x^{(n)}) \cdot t^{(n)})$

Ada Boost Algorithm

- Input: Data $\mathcal{D}_N$, weak classifier WeakLearn (a classification procedure that returns a classifier h, e.g. best decision stump, from a set of classifiers $\mathcal{H}$, e.g. all possible decision stumps), number of iterations T
- Output: Classifier $H(x)$
- Initialize sample weights: $w^{(n)} = \frac{1}{N}$ for $n = 1, \dots, N$
- For $t = 1, \dots, T$
  - Fit a classifier to weighted data ($h_t \leftarrow \text{WeakLearn}(\mathcal{D}_N, \mathbf{w})$), e.g.,

$$h_t \leftarrow \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{n=1}^{N} w^{(n)} \mathbb{I}\{h(\mathbf{x}^{(n)}) \neq t^{(n)}\}$$

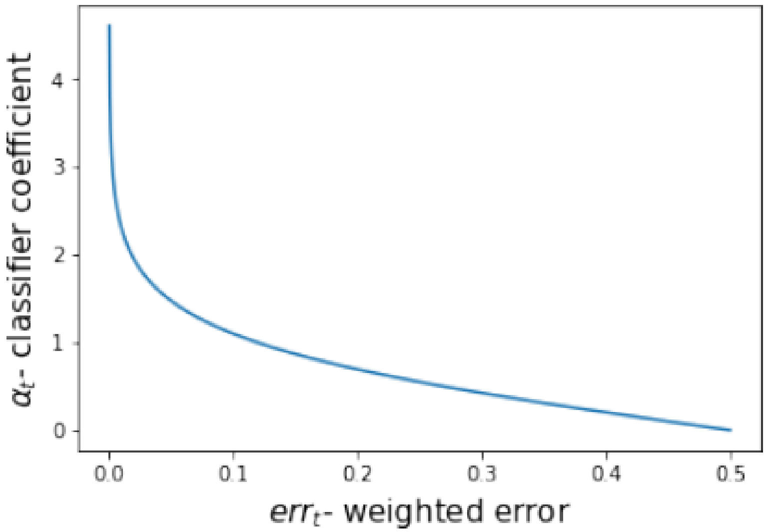
  - Compute weighted error $\text{err}_t = \frac{\sum_{n=1}^{N} w^{(n)} \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}}{\sum_{n=1}^{N} w^{(n)}}$
  - Compute classifier coefficient $\alpha_t = \frac{1}{2} \log \frac{1 - \text{err}_t}{\text{err}_t} \quad (\in (0, \infty))$
  - Update data weights

$$w^{(n)} \leftarrow w^{(n)} \exp\left(-\alpha_t t^{(n)} h_t(\mathbf{x}^{(n)})\right) \left[\equiv w^{(n)} \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}\right)\right]$$

- Return $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$

Weighting Intuition

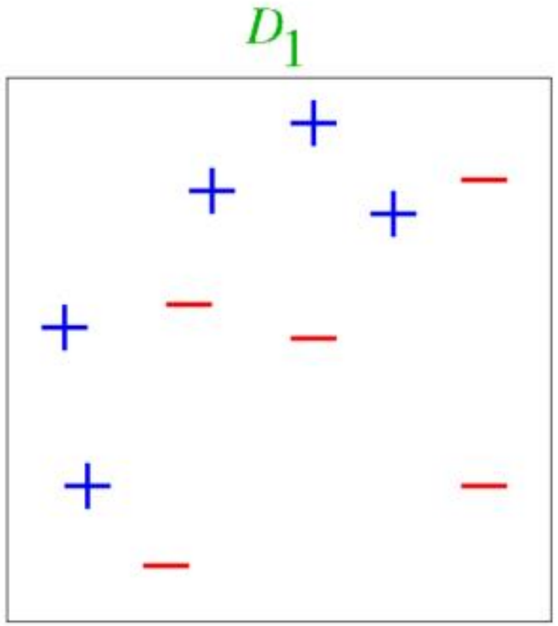
- Recall: $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$ where $\alpha_t = \frac{1}{2}\log\frac{1-\text{err}_t}{\text{err}_t}$



- Weak classifiers which get lower weighted error get more weight in the final classifier
- Also: $w^{(n)} \leftarrow w^{(n)} \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}\right)$
  - If $\text{err}_t \approx 0$, α_t high so misclassified examples get more attention
  - If $\text{err}_t \approx 0.5$, α_t low so misclassified examples are not emphasized

AdaBoost Example

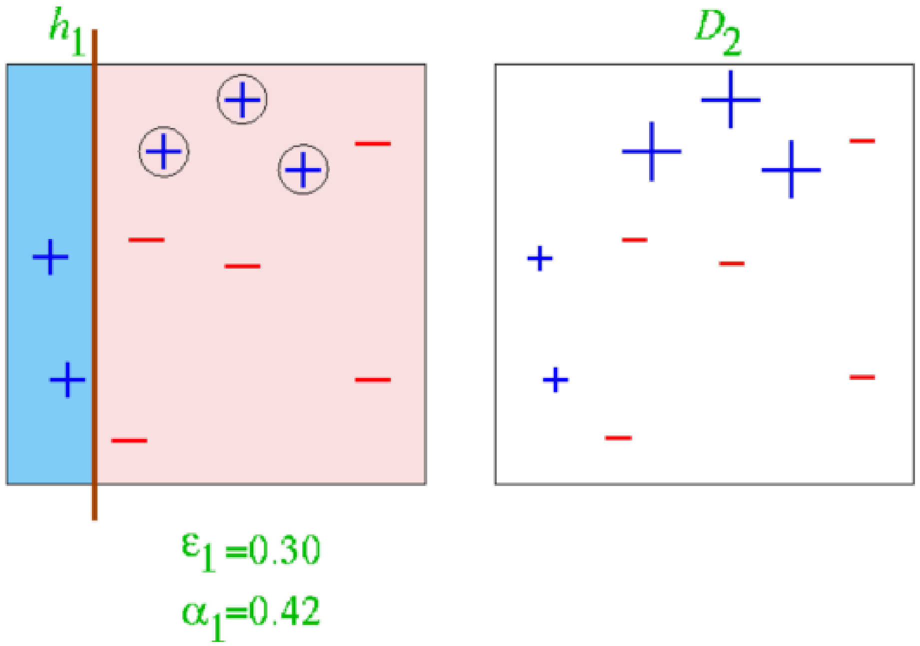
- Training data



[Slide credit: Verma & Thrun]

AdaBoost Example

- Round 1



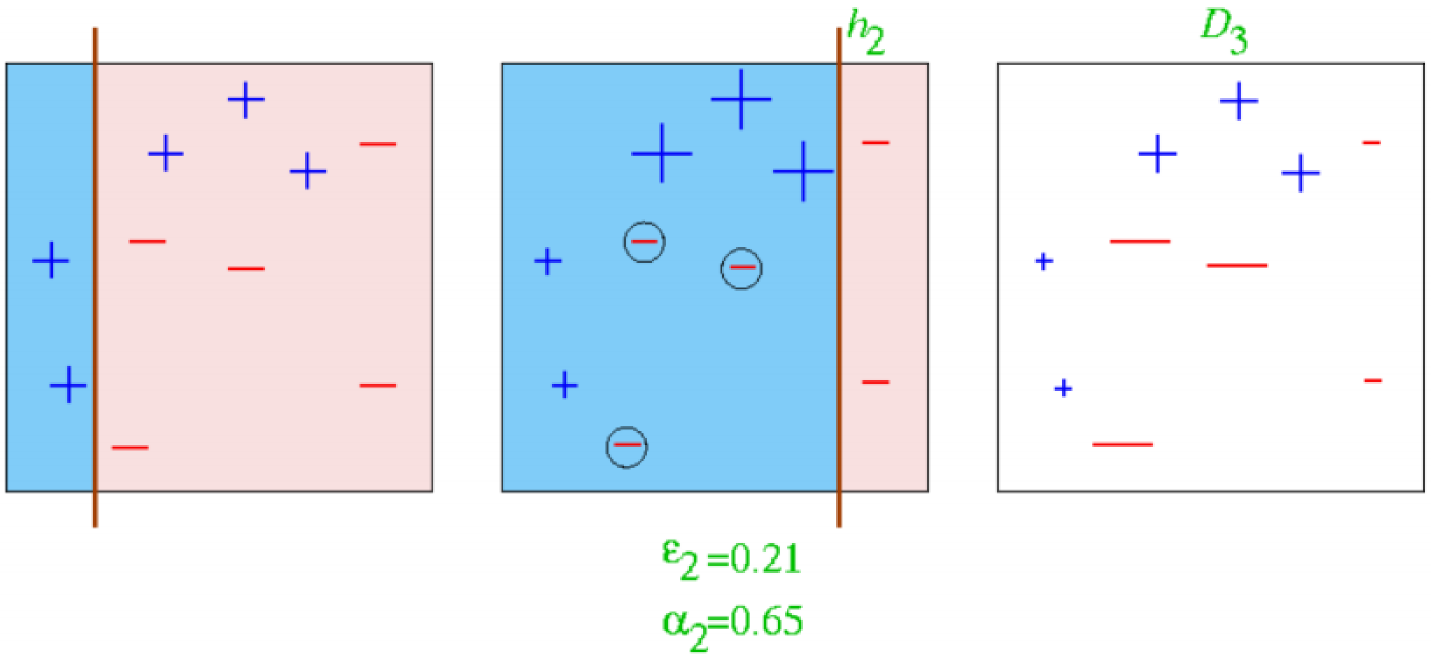
$$\mathbf{w} = \left(\frac{1}{10}, \ldots, \frac{1}{10}\right) \Rightarrow \text{Train a classifier (using } \mathbf{w}) \Rightarrow \text{err}_1 = \frac{\sum_{i=1}^{10} w_i \mathbb{I}\{h_1(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i} = \frac{3}{10}$$

$$\Rightarrow \alpha_1 = \frac{1}{2} \log \frac{1 - \text{err}_1}{\text{err}_1} = \frac{1}{2} \log\left(\frac{1}{0.3} - 1\right) \approx 0.42 \Rightarrow H(\mathbf{x}) = \text{sign}\left(\alpha_1 h_1(\mathbf{x})\right)$$

[Slide credit: Verma & Thrun]

AdaBoost Example

- Round 2



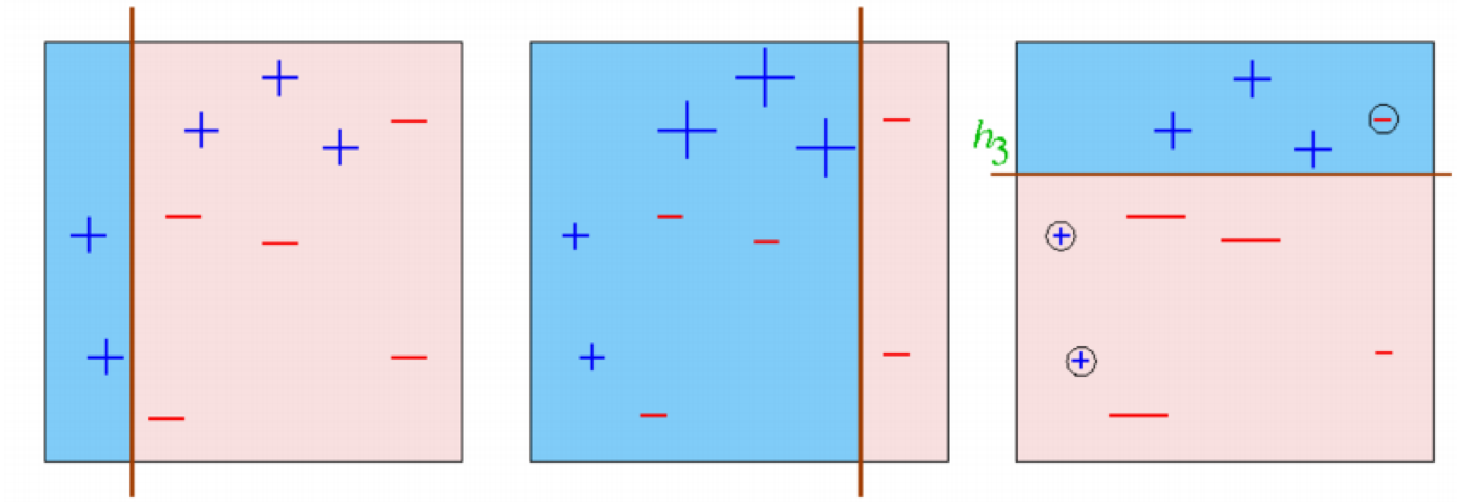
$$\mathbf{w} = \text{updated weights} \Rightarrow \text{Train a classifier (using } \mathbf{w}) \Rightarrow \text{err}_2 = \frac{\sum_{i=1}^{10} w_i \mathbb{I}\{h_2(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i} = 0.21$$

$$\Rightarrow \alpha_2 = \frac{1}{2} \log \frac{1 - \text{err}_3}{\text{err}_3} = \frac{1}{2} \log(\frac{1}{0.21} - 1) \approx 0.66 \Rightarrow H(\mathbf{x}) = \text{sign}\left(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x})\right)$$

[Slide credit: Verma & Thrun]

AdaBoost Example

- Round 3



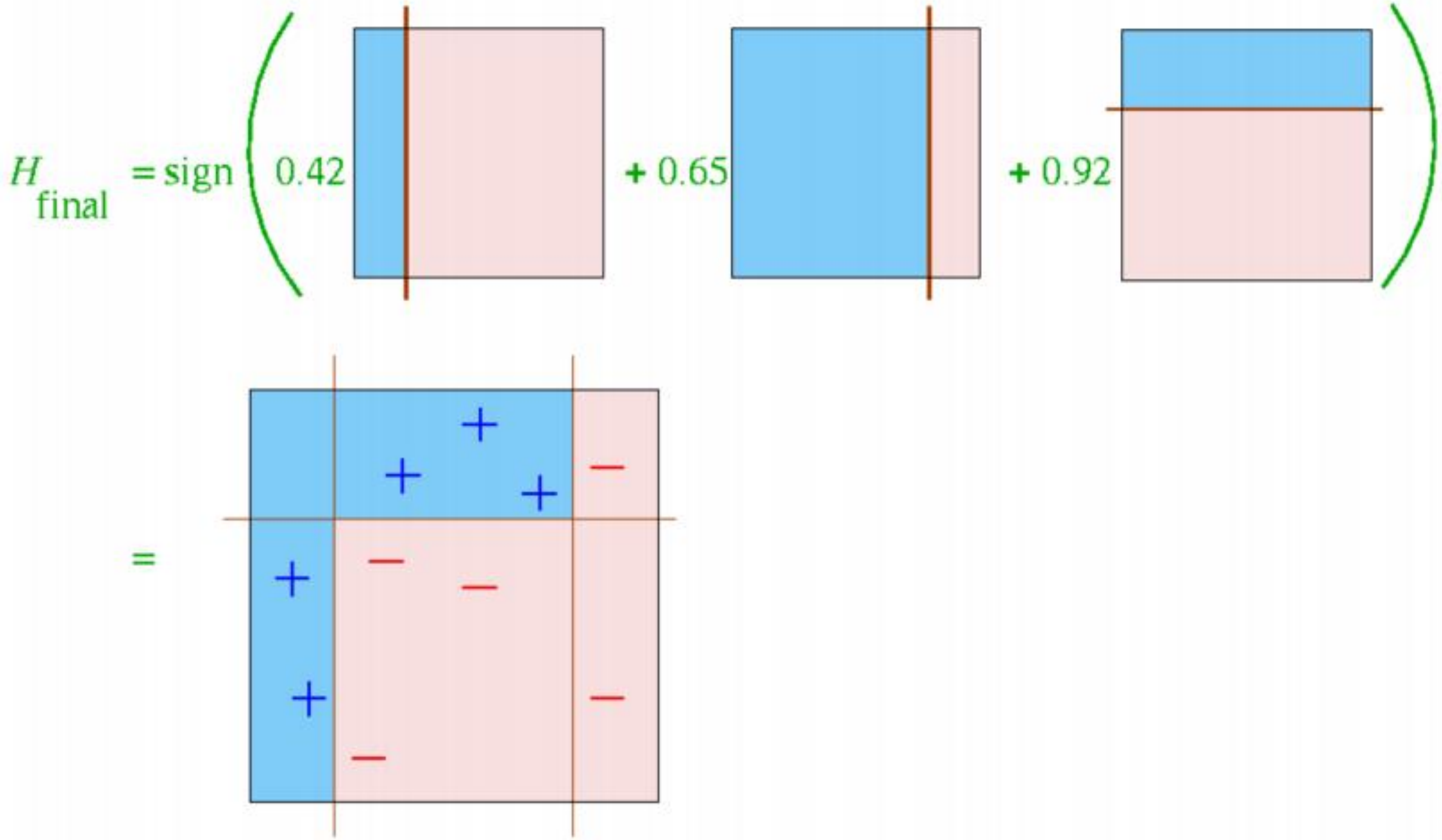
$$\mathbf{w} = \text{updated weights} \Rightarrow \text{Train a classifier (using } \mathbf{w}) \Rightarrow \text{err}_3 = \frac{\sum_{i=1}^{10} w_i \mathbb{I}\{h_3(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i} = 0.14$$

$$\Rightarrow \alpha_3 = \frac{1}{2} \log \frac{1 - \text{err}_3}{\text{err}_3} = \frac{1}{2} \log(\frac{1}{0.14} - 1) \approx 0.91 \Rightarrow H(\mathbf{x}) = \text{sign}\left(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \alpha_3 h_3(\mathbf{x})\right)$$

[Slide credit: Verma & Thrun]

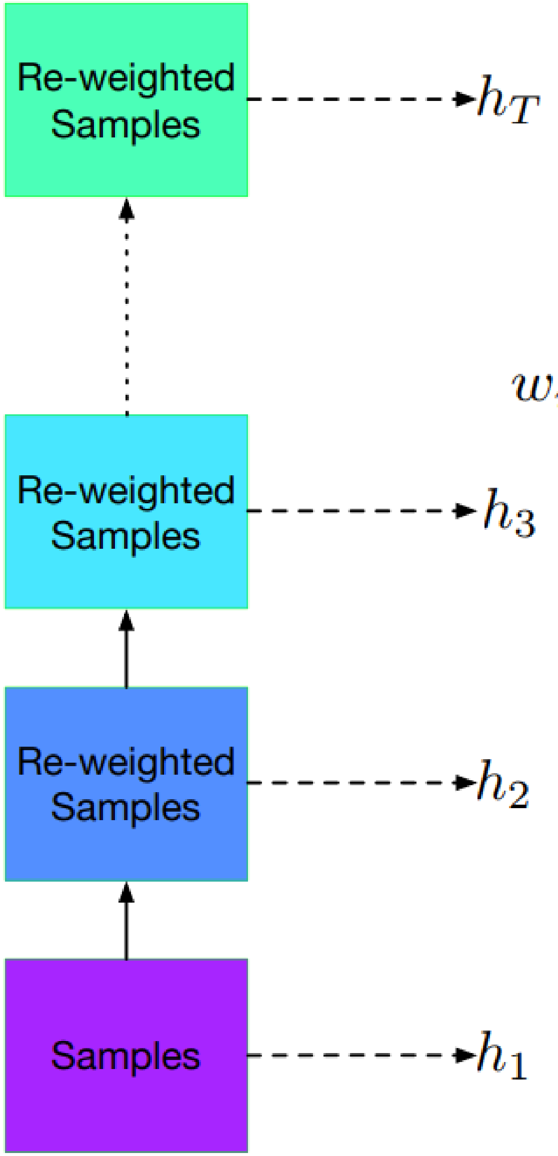
AdaBoost Example

- Final classifier



[Slide credit: Verma & Thrun]

AdaBoost Algorithm



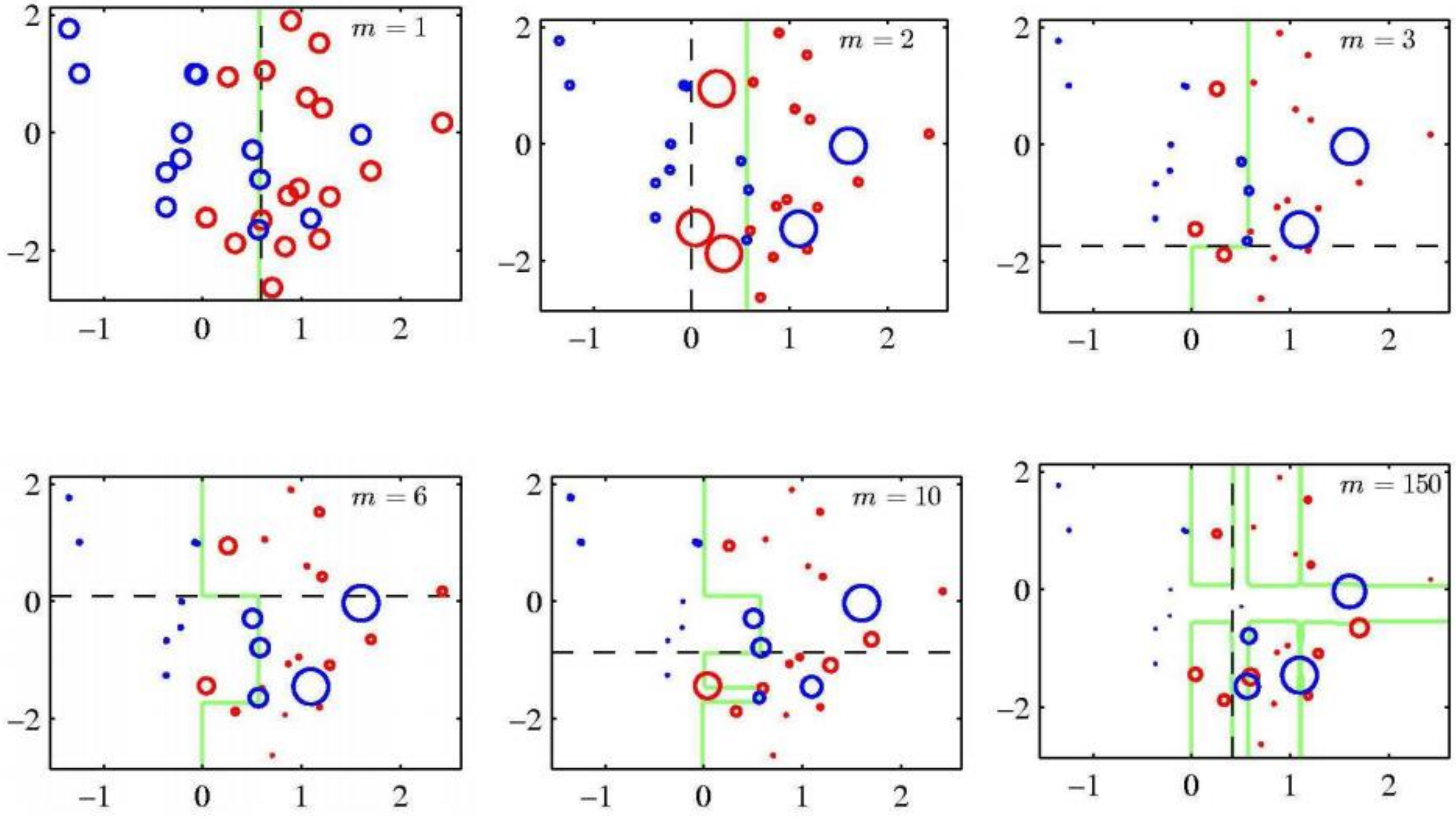
$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

$$w_i \leftarrow w_i \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(\mathbf{i})}) \neq \mathbf{t}^{(\mathbf{i})}\}\right)$$

$$\alpha_t = \frac{1}{2}\log\left(\frac{1 - \mathrm{err}_t}{\mathrm{err}_t}\right)$$

$$\mathrm{err}_t = \frac{\sum_{i=1}^{N} w_i \mathbb{I}\{h_t(\mathbf{x}^{(\mathbf{i})} \neq \mathbf{t}^{(\mathbf{i})}\}}{\sum_{i=1}^{N} w_i}$$

AdaBoost Example



- Each figure shows the number m of base learners trained so far, the decision of the most recent learner (dashed black), and the boundary of the ensemble (green)

AdaBoost Minimizes the Training Error

Theorem

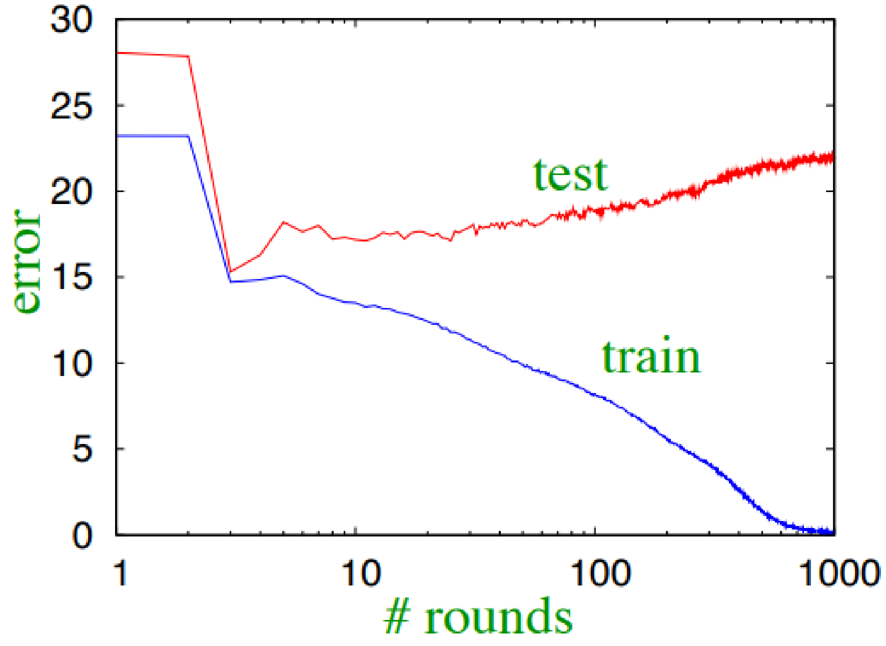
Assume that at each iteration of AdaBoost the WeakLearn returns a hypothesis with error $\mathrm{err}_t \leq \frac{1}{2} - \gamma$ for all $t = 1, \ldots, T$ with $\gamma > 0$. The training error of the output hypothesis $H(\mathbf{x}) = \mathrm{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$ is at most

$$L_N(H) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\{H(\mathbf{x}^{(i)}) \neq t^{(i)})\} \leq \exp\left(-2\gamma^2 T\right).$$

- This is under the simplifying assumption that each weak learner is γ-better than a random predictor.
- This is called geometric convergence. It is fast!

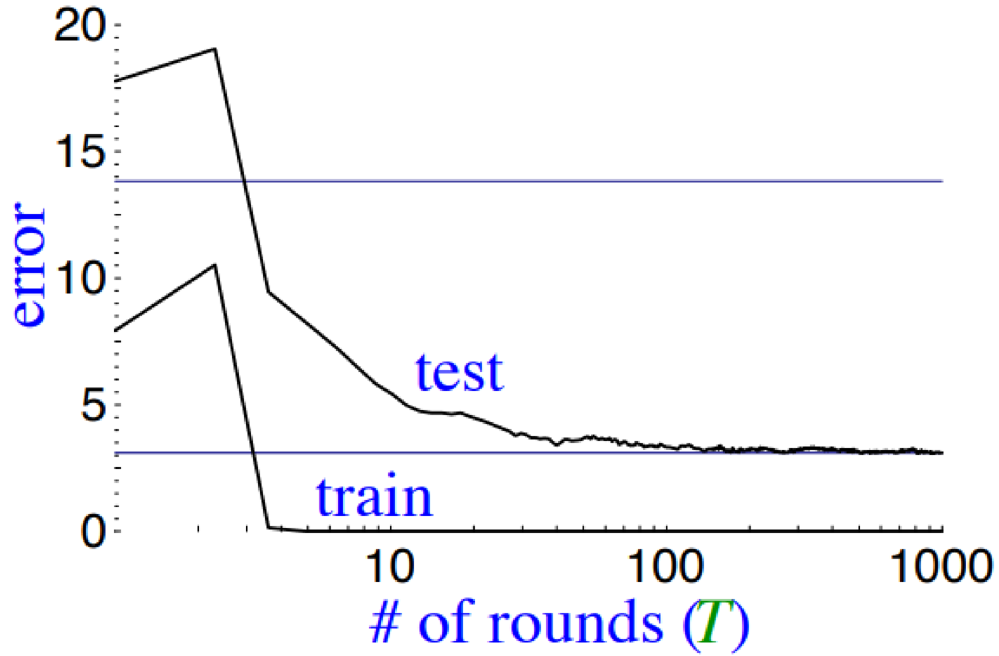
Generalization Error of AdaBoost

- AdaBoost's training error (loss) converges to zero. What about the test error of H?
- As we add more weak classifiers, the overall classifier H becomes more "complex".
- We expect more complex classifiers overfit.
- If one runs AdaBoost long enough, it can in fact overfit.



Generalization Error of AdaBoost

- But often it does not!
- Sometimes the test error decreases even after the training error is zero!



- How does that happen?
- Next, we provide an alternative viewpoint on AdaBoost.

Additive Models

Next, we'll now interpret AdaBoost as a way of fitting an additive model.

- Consider a hypothesis class $\mathcal{H}$ with each $h_i : \mathbf{x} \mapsto \{-1, +1\}$ within $\mathcal{H}$, i.e., $h_i \in \mathcal{H}$. These are the "weak learners", and in this context they're also called **bases**.
- An **additive model** with m terms is given by

$$H_m(x) = \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x}),$$

  where $(\alpha_1, \cdots, \alpha_m) \in \mathbb{R}^m$.
- Observe that we're taking a linear combination of base classifiers $h_i(\mathbf{x})$, just like in boosting.
- Note also the connection to feature maps (or basis expansions) that we saw in linear regression and neural networks!

Stagewise Training of Additive Models

A greedy approach to fitting additive models, known as **stagewise training**:

1. Initialize $H_0(x) = 0$
2. For $m = 1$ to T:
   - Compute the m-th hypothesis $H_m = H_{m-1} + \alpha_m h_m$, i.e. h_m and α_m, assuming previous additive model H_{m-1} is fixed:

$$(h_m, \alpha_m) \leftarrow \underset{h \in \mathcal{H}, \alpha}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}\left(H_{m-1}(\mathbf{x}^{(i)}) + \alpha h(\mathbf{x}^{(i)}),\ t^{(i)}\right)$$

   - Add it to the additive model

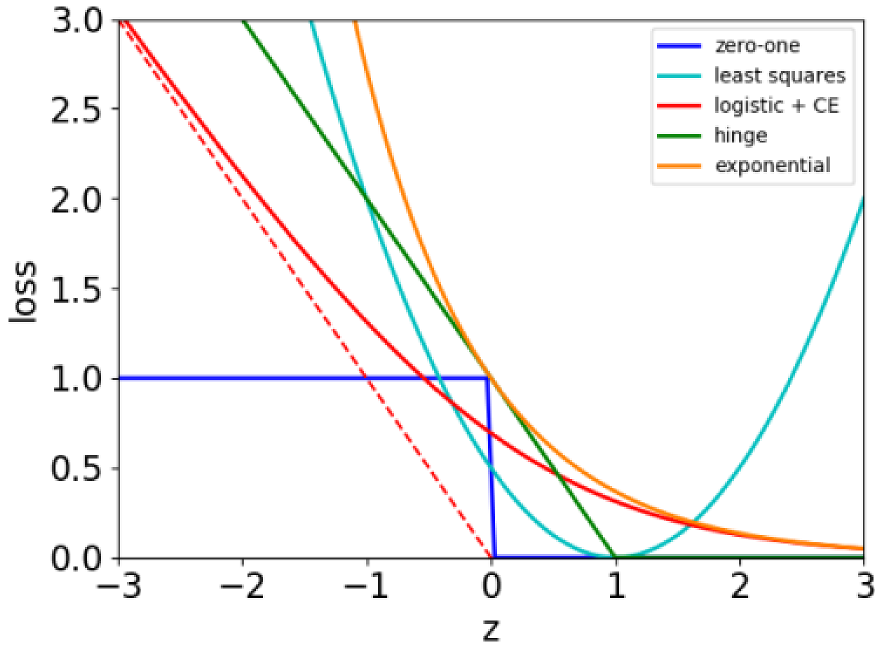
$$H_m = H_{m-1} + \alpha_m h_m$$

Additive Models with Exponential Loss

Consider the exponential loss

$$\mathcal{L}_{\mathrm{E}}(z,t) = \exp(-tz).$$

We want to see how the stagewise training of additive models can be done.



Additive Models with Exponential Loss

Consider the exponential loss

$$\mathcal{L}_{\mathrm{E}}(z, t) = \exp(-tz).$$

We want to see how the stagewise training of additive models can be done.

$$\begin{aligned}
(h_m, \alpha_m) &\leftarrow \underset{h \in \mathcal{H}, \alpha}{\operatorname{argmin}} \sum_{i=1}^{N} \exp\left(-\left[H_{m-1}(\mathbf{x}^{(i)}) + \alpha h(\mathbf{x}^{(i)})\right] t^{(i)}\right) \\
&= \sum_{i=1}^{N} \exp\left(-H_{m-1}(\mathbf{x}^{(i)}) t^{(i)}\right) \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) \\
&= \sum_{i=1}^{N} w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right).
\end{aligned}$$

Here we defined $w_i^{(m)} \triangleq \exp\left(-H_{m-1}(\mathbf{x}^{(i)}) t^{(i)}\right)$ (doesn't depend on h, α).

Additive Models with Exponential Loss

We want to solve the following minimization problem:

$$(h_m, \alpha_m) \leftarrow \underset{h \in \mathcal{H}, \alpha}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right). \qquad (1)$$

- Recall

$$w^{(n)} \exp\left(-\alpha_t h_t(\mathbf{x}^{(n)}) t^{(n)}\right) \propto w^{(n)} \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}\right)$$

- Thus, for h_m, the above minimization is equivalent to:

$$\begin{aligned}
h_m &\leftarrow \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}\right) \\
&= \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \left(\exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}\right) - 1\right) && \triangleright \text{ subtract } \textstyle\sum w_i^{(m)} \\
&= \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\} && \triangleright \text{ divide by } (\exp(2\alpha_t) - 1)
\end{aligned}$$

- This means that h_m is the minimizer of the weighted 0/1-loss.

Additive Models with Exponential Loss

- Now that we obtained h_m, we can plug it into our exponential loss objective (1) and solve for α_m.
- The derivation is a bit laborious and doesn't provide additional insight, so we skip it.
- We arrive at:

$$\alpha_m = \frac{1}{2} \log \left(\frac{1 - \mathrm{err}_m}{\mathrm{err}_m} \right),$$

where err_m is the weighted classification error:

$$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i^{(m)}}.$$

Additive Models with Exponential Loss

We can now find the updated weights for the next iteration:

$$\begin{aligned}
w_i^{(m+1)} &= \exp\left(-H_m(\mathbf{x}^{(i)})t^{(i)}\right) \\
&= \exp\left(-\left[H_{m-1}(\mathbf{x}^{(i)}) + \alpha_m h_m(\mathbf{x}^{(i)})\right] t^{(i)}\right) \\
&= \exp\left(-H_{m-1}(\mathbf{x}^{(i)})t^{(i)}\right) \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)})t^{(i)}\right) \\
&= w_i^{(m)} \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)})t^{(i)}\right)
\end{aligned}$$

Additive Models with Exponential Loss

To summarize, we obtain the additive model $H_m(x) = \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x})$ with

$$h_m \leftarrow \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\},$$

$$\alpha = \frac{1}{2} \log\left(\frac{1 - \mathrm{err}_m}{\mathrm{err}_m}\right), \qquad \text{where } \mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i^{(m)}},$$

$$w_i^{(m+1)} = w_i^{(m)} \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)}) t^{(i)}\right).$$

We derived the AdaBoost algorithm!

Boosting Summary

- Boosting reduces bias by generating an ensemble of weak classifiers.
- Each classifier is trained to reduce errors of previous ensemble.
- It is quite resilient to overfitting, though it can overfit.

Ensembles Recap

- Ensembles combine classifiers to improve performance
- Boosting
  - Reduces bias
  - Increases variance (large ensemble can cause overfitting)
  - Sequential
  - High dependency between ensemble elements
- Bagging
  - Reduces variance (large ensemble can't cause overfitting)
  - Bias is not changed (much)
  - Parallel
  - Want to minimize correlation between ensemble elements.