

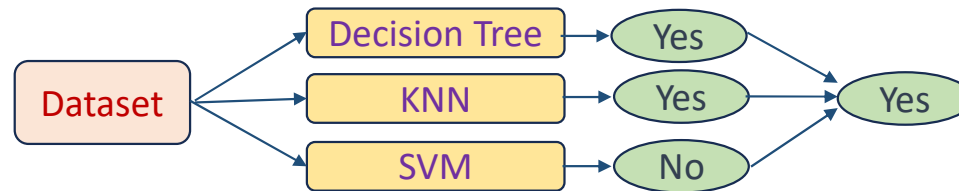
Ensemble Learning

Introduction

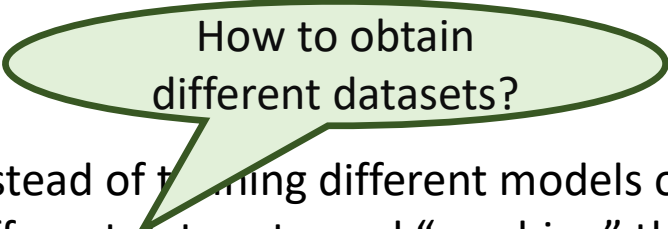
Goal of Supervised Learning: Minimize the probability of model prediction errors on future data

Methodology

1. **Traditional Approach:** Build one really good model
2. **Ensemble Learning:** Build many models and average the result
 - Voting or Averaging of predictions of multiple pre-trained models



Ensemble Learning



How to obtain
different datasets?

- Instead of training different models on same data, we can train same model multiple times on different data sets, and “combine” these “different” models
- Different models may be good at different 'parts' of data with individual mistakes 'averaged out'
- Ensembles are more robust to noisy data and outliers.
- Ensembles often reduce both **bias and variance**.

Bias and Variance

Ideal Model: A model which can capture the pattern in the training data while simultaneously generalizing well to the test data

Low Bias and a Low Variance

Bias:

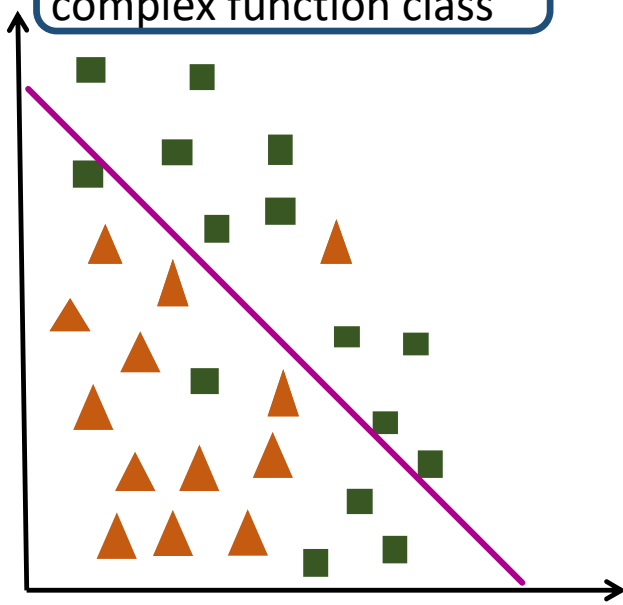
- ✓ Error caused because the model can not represent the concept
- ✓ A model with a high bias ignores the training data, oversimplifying the model

Variance:

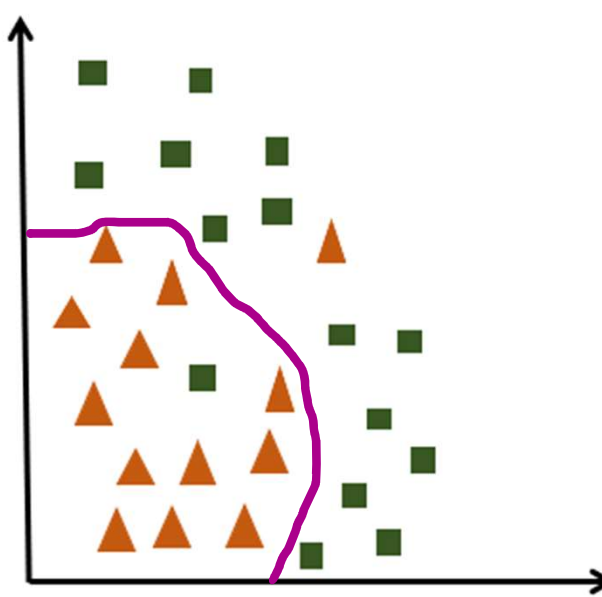
- ✓ Error caused because learning algorithm overreacts to small changes (noise) in the training data
- ✓ Model has a very complex fit to the training data and thus is unable to fit accurately on new data

Bias and Variance

We have an insufficiently complex function class

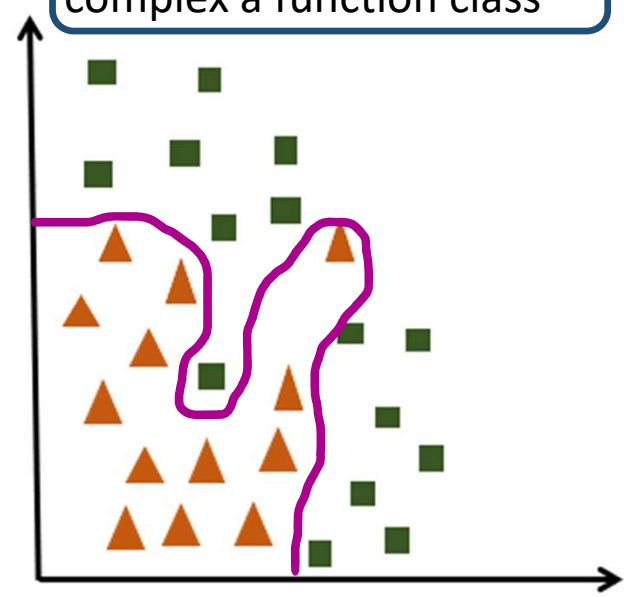


High Bias [Underfitting]

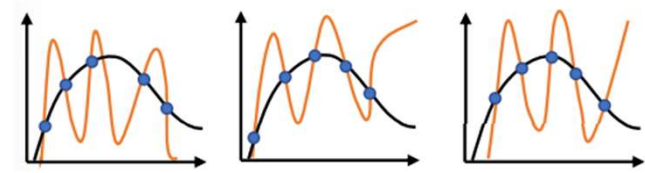
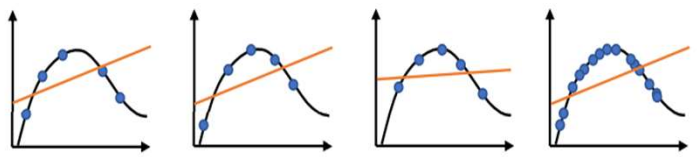


“Good Fit”, Just Right

We have too little data/too complex a function class



High Variance [Overfitting]



Classification



No Mask: Class 0



Mask: Class 1

Training Data Accuracy	79%	35%	38%	82%
Testing Data Accuracy	32%	36%	15%	83%

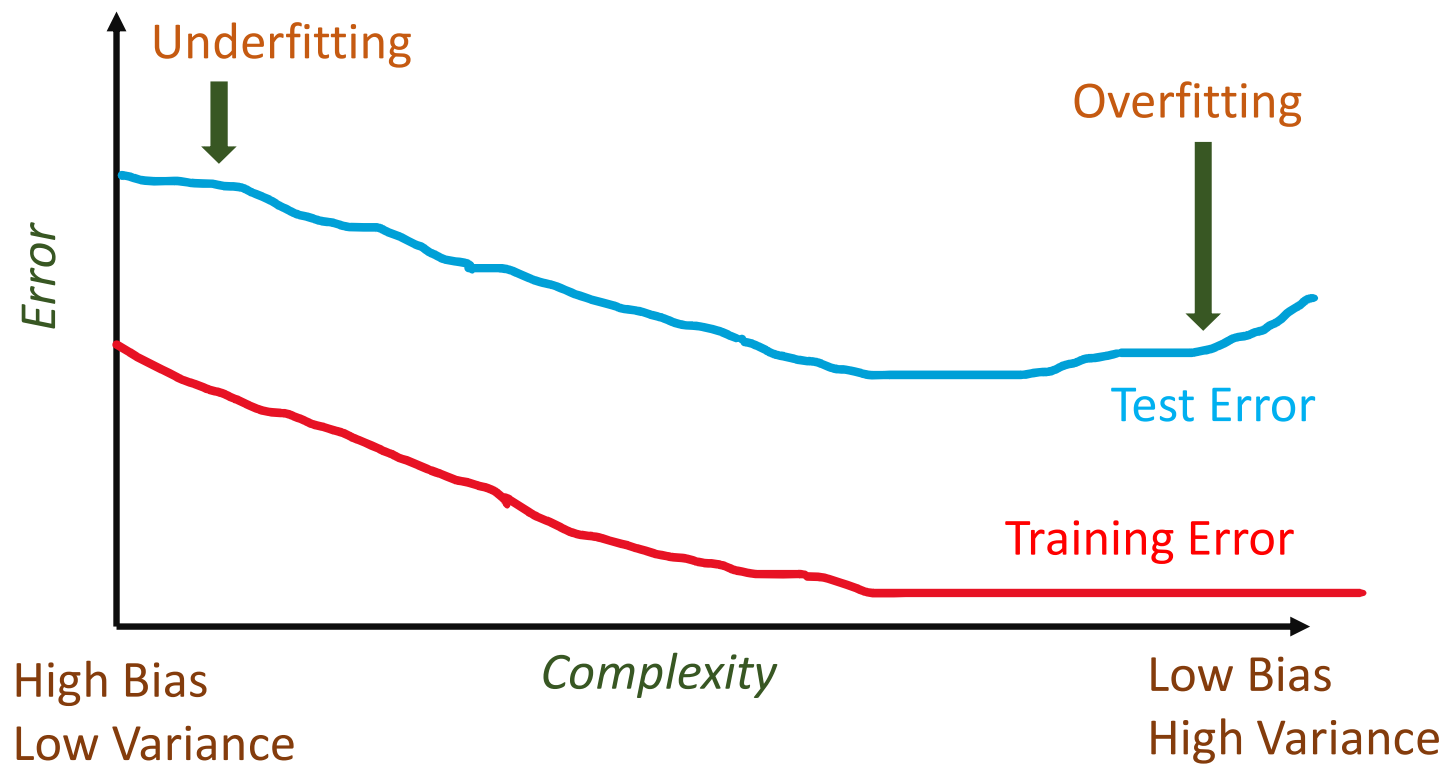
High Variance
[Overfitting]

High Bias
[Underfit]

High Bias and
High Variance

Low Bias and
Low Variance

Bias-Variance Trade-off



Decision Trees [Recap]

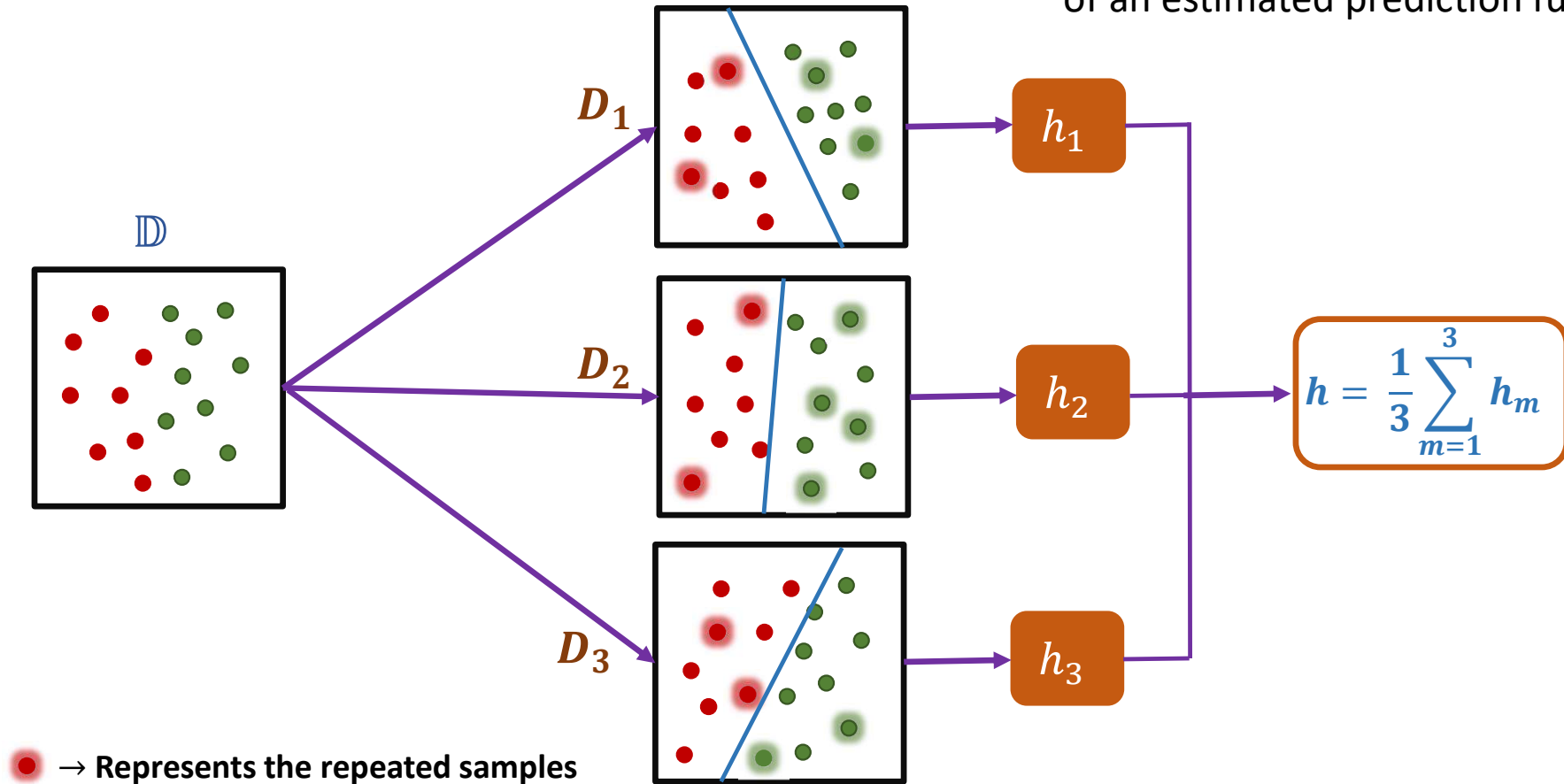
- Decision Trees are prone to overfitting. Can be prevented by controlling the (maximum) depth of the tree
- Shallow trees tend to have high bias but very low variance (underfitting)
- Deep trees have high variance but low bias (overfitting)
- Decision Trees are ideal for ensembling as their complexity can be easily controlled
 - Shallow trees: low variance, high bias can be reduced with **Boosting**
 - Deep trees: low bias, high variance can be reduced with **Bagging**

Bagging (Bootstrap Aggregating)

- It is an ensemble learning technique designed to improve the stability and accuracy of ML models by creating multiple subsets (bags) of the training dataset through a process called bootstrap sampling.
- Given an original dataset \mathbb{D} with N training examples
 - Create M samples $\{D_m\}_{m=1}^M$ of size N by drawing N examples from the original data, with replacement
- Each bootstrap samples (D_m) are reasonably different from each other and on average contain 63.2% of the unique training examples [rest are replicates]
- Train models $h_{1,...,M}$, using the respective datasets $D_{1,...,M}$, respectively, and average the models $h = \frac{1}{M} \sum_{m=1}^M h_m$ to obtain final decision

Bagging [Illustration]

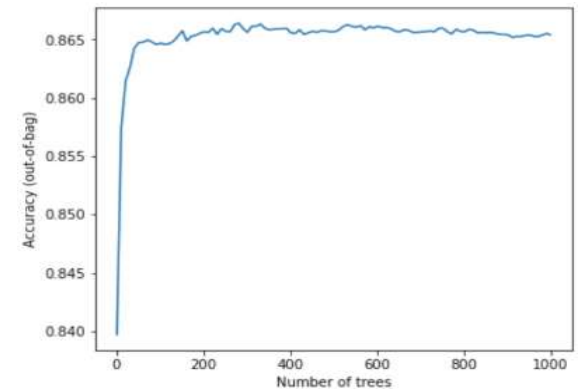
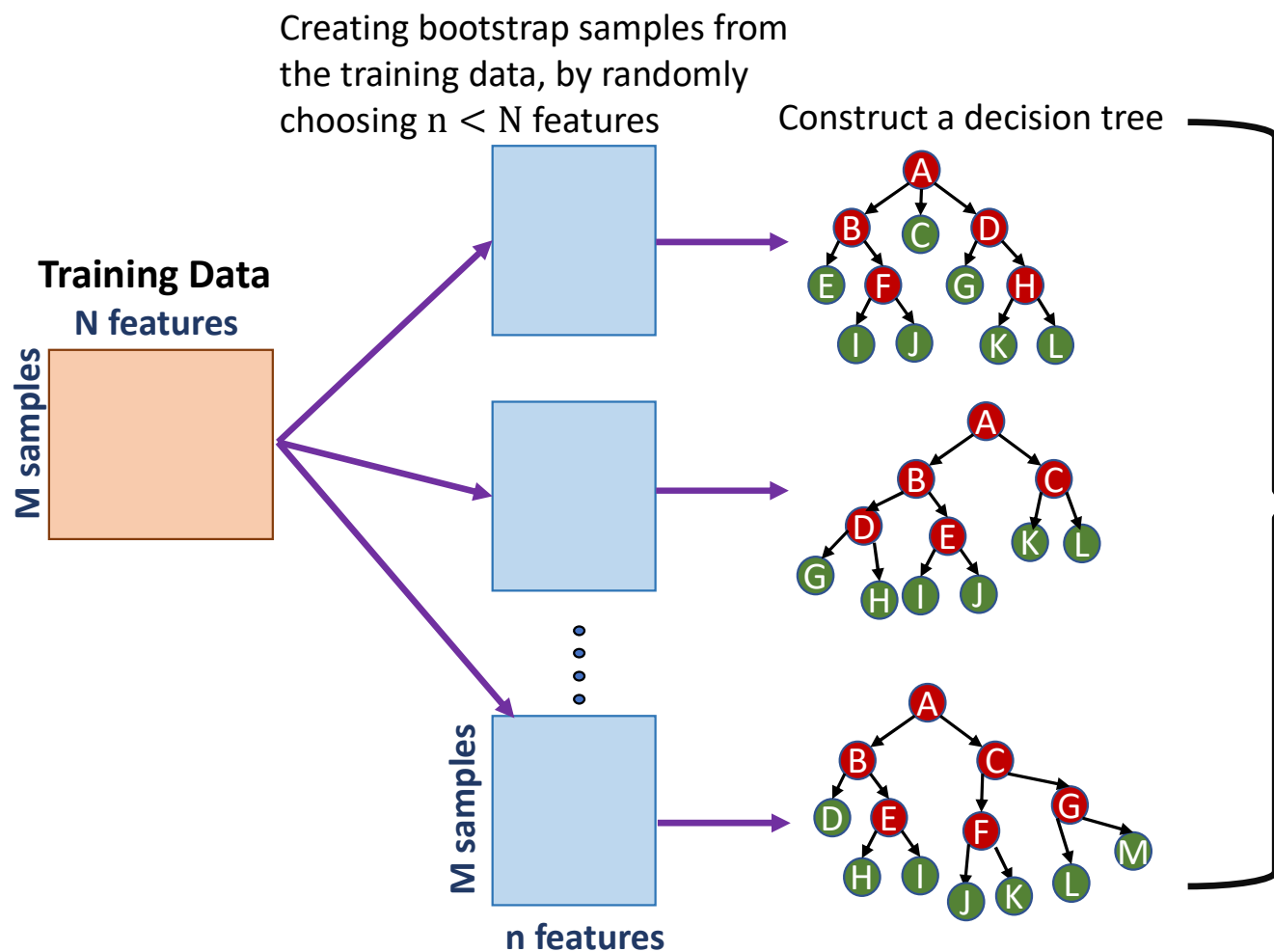
Bagging technique reduces the variance of an estimated prediction function.



Random Forests

- **Random forest** was first proposed by Tin Kam Ho of Bell Labs in 1995 and is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees.
- Uses bagging on features with each decision tree using a random set of features
 - Given a total of n features, each decision tree uses \sqrt{n} randomly chosen features
 - Randomly chosen features make the different trees de-correlated
- The decision trees usually have the same depth, with each of them splitting the training data differently at the leaves
- Prediction for a test example votes on/averages predictions from all the decision

Random Forest Classifier



Take the majority vote, outputs the class that is the mode of the class's output by individual decision trees

"Random Forests do not overfit, as more trees are added"

Leo Breiman

Notes on Random Forests

- Resists Overfitting
 - Partly maintains Explainability
 - Automatic Feature Selection
 - Efficient in Training and inference
 - Parallel training and inference
-
- As the decision trees are not pruned, size of random forest can be an issue
 - Cannot learn and reuse internal representation

Boosting

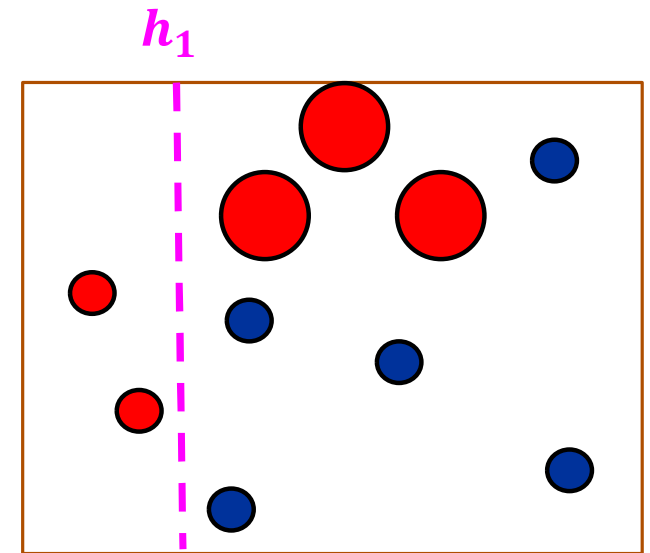
- Boosting is a sequential algorithm, which starts with a weak learning algorithm, and gradually improves by focusing on the misclassified/difficult cases.
- Training Dataset: $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, with \mathbf{x}_i as the features and $y_i \in \{-1, +1\}$, the corresponding binary class label
- Weak learner (base classifier) $h_t(\mathbf{x})$ takes an input \mathbf{x} and outputs a weak prediction $h_t(\mathbf{x}) \in \{-1, +1\}$
- Base (weak) learner must focus on correctly classifying the **most highly weighted examples** while strongly avoiding over-fitting.
- During testing, each of the hypotheses get a weighted vote proportional to their accuracy on the training data.

Adaptive Boosting - AdaBoost

1. Initialize Weights: Set initial weights uniformly: $w_i^{(1)} = \frac{1}{n}$ for $i = 1, 2, \dots, n$
2. For $t = 1$ to T , where T is the number of iterations:
 - Train a weak learner $h_t(x)$ using the training data D with weights $w_i^{(t)}$
3. Compute the weighted error of the weak learner: $\epsilon_t = \sum_{i=1}^n w_i^{(t)} \cdot \mathbb{I}[h_t(x_i) \neq y_i]$
4. Compute the weight of the weak learner in the final ensemble: $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
5. Update the weights of the examples: $w_i^{(t+1)} = w_i^{(t)} \cdot e^{(-\alpha_t \cdot y_i \cdot h_t(x_i))}$
6. Normalize weights to ensure they sum to 1: $w_i^{(t+1)} = \frac{w_i^{(t+1)}}{\sum_{i=1}^n w_i^{(t+1)}}$
7. Combine weak learners into a strong classifier: $H(x) = \text{sign}(\sum_{t=1}^T \alpha_t \cdot h_t(x))$

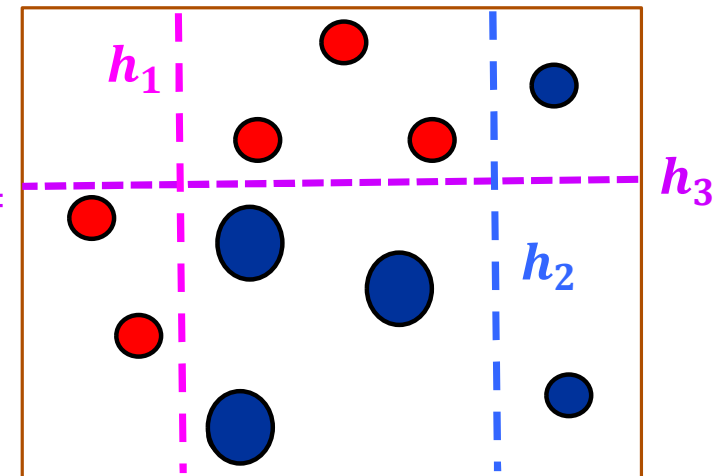
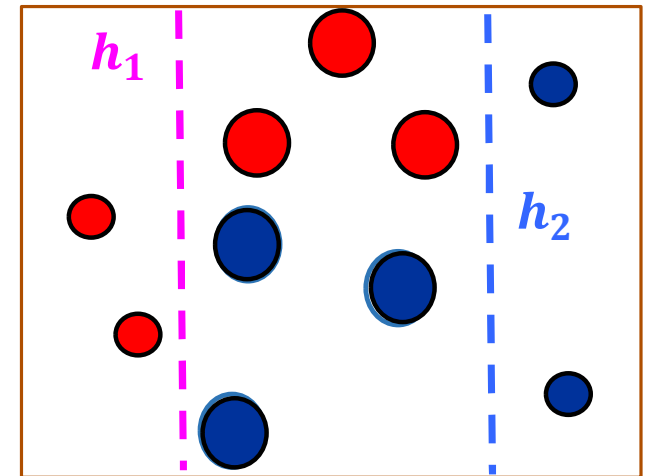
AdaBoost Illustration

- Consider binary classification with 10 training examples
- Initial weight distribution D_1 is uniform (each point has equal weight = 1/10)
- Error rate of h_1 : $\varepsilon_1 = 0.3$; weight of h_1 : $\alpha_1 = \frac{1}{2} \ln \left(\frac{1-0.3}{0.3} \right) = 0.42$
- Each misclassified point is up-weighted, weight multiplied by e^{α_1}
- Each correctly classified point is down-weighted, weight multiplied by $e^{-\alpha_1}$



Boosting Illustration

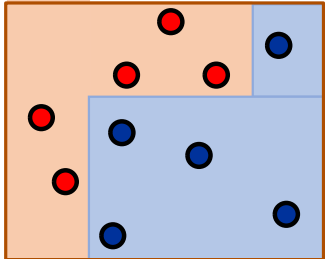
- Error rate of h_2 : $\varepsilon_2 = 0.21$; weight of h_2 : $\alpha_2 = \frac{1}{2} \ln \left(\frac{1-0.21}{0.21} \right) = 0.65$
- Each misclassified point is up-weighted, weight multiplied by e^{α_2}
- Each correctly classified point is down-weighted, weight multiplied by $e^{-\alpha_2}$
- Error rate of h_3 : $\varepsilon_3 = 0.14$; weight of h_3 : $\alpha_3 = \frac{1}{2} \ln \left(\frac{1-0.14}{0.14} \right) = 0.92$
- Suppose we decide to stop after round 3
- Our ensemble now consists of 3 classifiers, h_1, h_2, h_3 .



AdaBoost - Final Classifier

- Final classifier is a weighted linear combination of all the classifiers
- Classifier h_i gets a weight α_i

$$H_{final} = \text{sign} \left[0.42 \begin{array}{|c|} \hline \text{Diagram 1} \\ \hline \end{array} + 0.65 \begin{array}{|c|} \hline \text{Diagram 2} \\ \hline \end{array} + 0.92 \begin{array}{|c|} \hline \text{Diagram 3} \\ \hline \end{array} \right]$$

= 

The diagrams illustrate the combination of three weak classifiers into a strong classifier. Each diagram shows a 2D space with red and blue data points. The first three diagrams show the individual weak classifiers with their respective decision boundaries (dashed lines) and weights (0.42, 0.65, and 0.92). The final diagram shows the resulting strong classifier's decision boundary, which is a non-linear combination of the weak classifiers' boundaries.

- Multiple weak, linear classifiers combined to give a strong, nonlinear classifier

Gradient Boosting

- Ensemble of models, with the models built sequentially each fixing the remaining mistakes of the previous ones
- Achieved by building a new model on the errors or residuals of the previous model.
- Training Dataset: $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, with \mathbf{x}_i as the features and $y_i \in \{-1, +1\}$, the corresponding binary class label
- Number of weak learners (trees): T , Learning Rate: η
- We consider Logistic Loss, $L(y, F(x)) = \log(1 + e^{-y \cdot F(x)})$

Gradient Boosting

1. Set the initial prediction to the mean of the target value: $F_0(x) = \frac{1}{n} \sum_{i=1}^n y_i$
2. Compute the Pseudo-Residuals: $r_i^{(t)} = - \frac{\partial L(y_i, F_{t-1}(x_i))}{\partial F_{t-1}(x_i)} = \frac{y_i}{1 + e^{y_i \cdot F_{t-1}(x_i)}}$
3. Train a weak learner (e.g., a decision tree) to predict the pseudo-residuals: $h_t(x)$
4. Compute the contribution of the weak learner to the overall model:
 $F_t(x) = F_{t-1}(x) + \eta \cdot h_t(x)$
5. Update the model with the new contribution from the weak learner
6. Compute the Pseudo-Residuals: $r_i^{(t+1)} = - \frac{\partial L(y_i, F_t(x_i))}{\partial F_t(x_i)}$
7. The final model is the sum of all weak learners: $F(x) = F_0(x) + \eta \cdot \sum_{t=1}^T h_t(x)$
8. The final prediction for x is obtained by applying the logistic function to the final model's output, $\hat{y} = \text{sign} \left(\frac{1}{1 + e^{-F(x)}} \right)$

Gradient Boosting

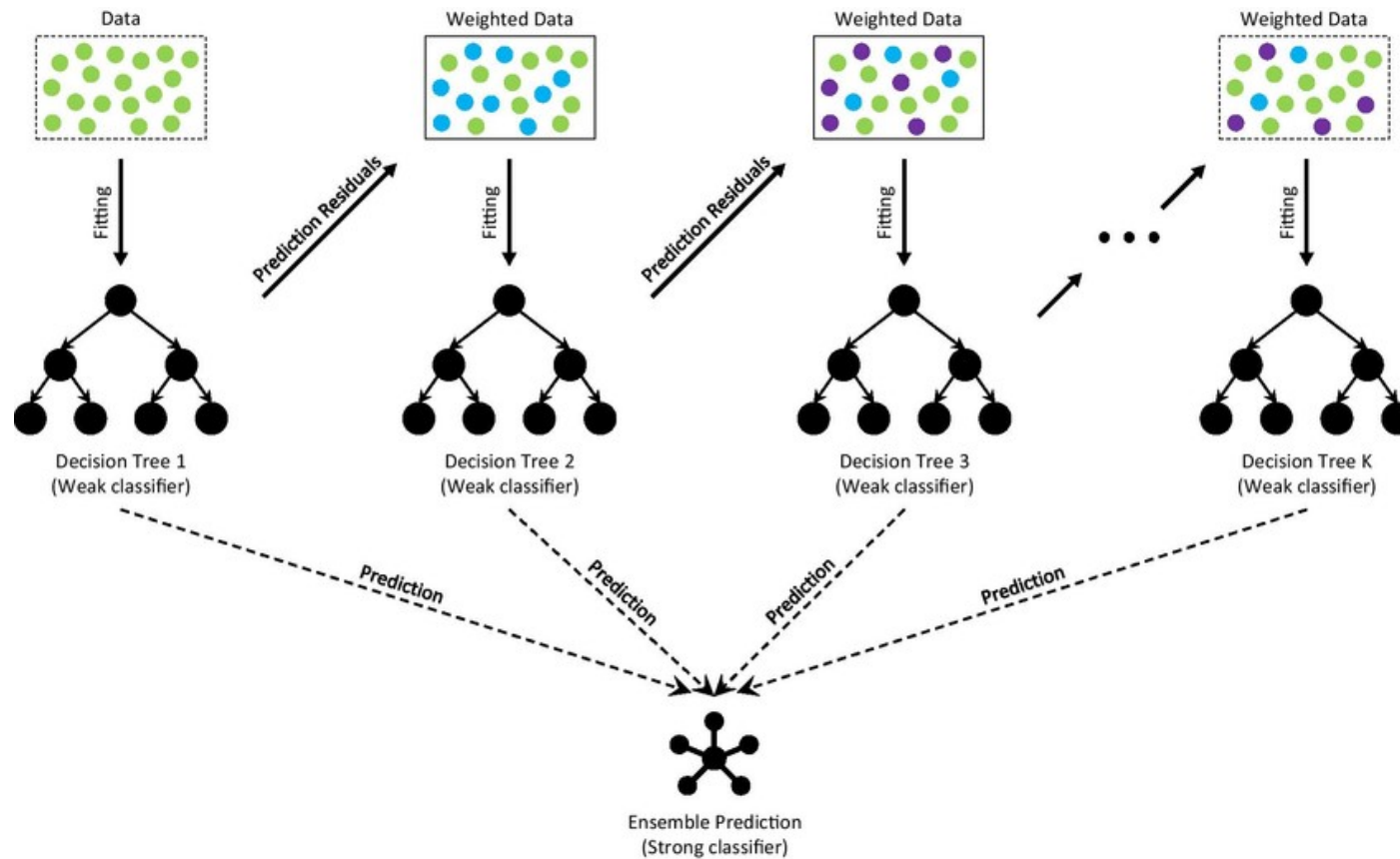


Image Source: https://www.researchgate.net/figure/The-architecture-of-Gradient-Boosting-Decision-Tree_fig2_356698772

Bagging vs Boosting

Method	Training Approach	Diversity	Algorithm Examples
Bagging	<ul style="list-style-type: none">Parallel training of base modelsEach model has equal weight in the final prediction	Creates diverse models by training on different subsets of the data and reduce variance	Random Forests
Boosting	<ul style="list-style-type: none">Sequential training, with each model focusing on correcting the errors of the previous ones.Dynamic Weights	Reduce bias and improves accuracy by giving more attention to examples that are frequently misclassified.	AdaBoost, Gradient Boosting

- No clear winner; usually depends on the specific problem and the dataset
- Bagging is computationally more efficient than boosting (note that bagging can train the M models in parallel, boosting can't)
- Bagging is generally robust and less prone to overfitting, while boosting can achieve higher accuracy but may be more sensitive to noisy data