# 13. Multiple Learners

*Reference: Intro to ML Ch. 17, Pattern etc. Ch. 14.*

Train many different learners/models and then combine results

**Committes:** set of models trained on a dataset.

## 13.1 Voting (parallel training)

1. use $D$ to train a set of models $y_m(\mathbf{x})$, for $m = 1, \ldots \boxed{M}$ Models
2. make predictions with weighted Sum of all the inputs

$$y_{voting}(\mathbf{x}) = \sum_{m=1}^{M} w_m y_m(\mathbf{x}) \quad \text{lin. comb of the outputs.} \quad \text{(regression)}$$

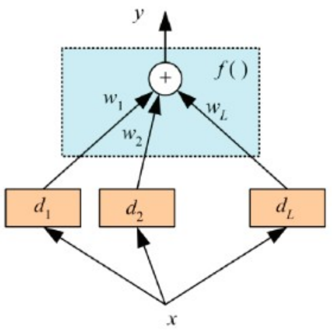


Figure 1: Voting

$$y_{voting}(\mathbf{x}) = \operatorname*{argmax}_{c} \sum_{m=1}^{M} w_m I(y_m(\mathbf{x}) = c) \qquad \text{weighted majority (classification)}$$

with $w_m \geq 0$, $\sum_m w_m = 1$ (prior probability of each model), $I(e) = 1$ if $e$ is true, 0 otherwise.

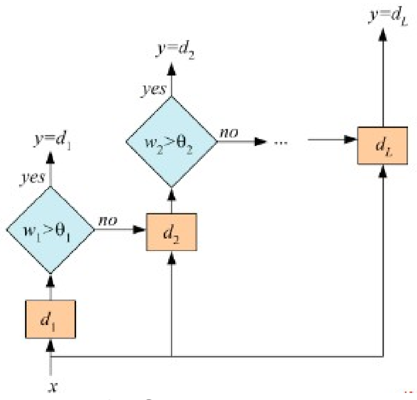


Figure 2: Cascading

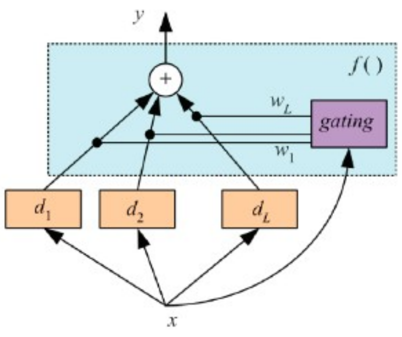


Figure 3: Mixture of experts

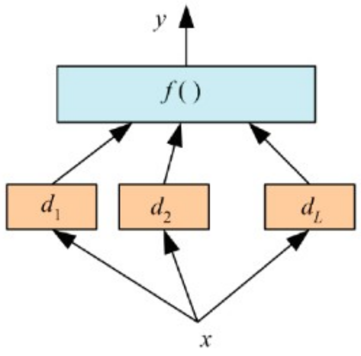


Figure 4: Stacking

## 13.2 Bagging (Parallel learning)

1. Generate M bootstraps datasets
2. Use each to train a model $y_m(x)$
3. make prediction with a voting scheme.

$$y_{bagging}(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} y_m(\mathbf{x})$$

Solve the problem of data availability

Only one model or slightly different models

Better than training any individual model.

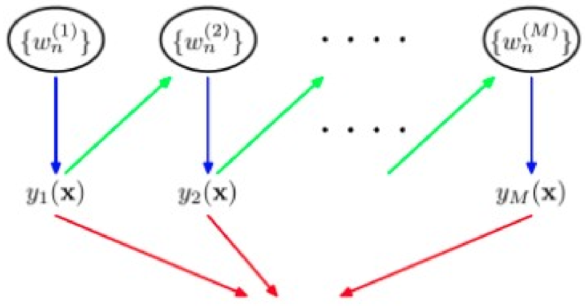
Bootstrap datasets chosen randomly

## 13.3 Boosting (Sequential learning)

Base classifiers trained sequentially on weighted data.

Weights depend on performance of previous classifiers (greater weights to points misclasified previously).

Predictions based on weighted majority of votes.



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right)$$

### 13.3.1 AdaBoost

Given $D = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$, where $\mathbf{x}_n \in \mathbf{X}$, $t_n \in \underline{\{-1, +1\}}$ outputs

1. Initialize $\underline{w_n^{(1)} = 1/N}$, $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$: assign uniform weights. [illegible] 15:46
   - Train a weak learner $y_m(\mathbf{x})$ by minimizing the weighted error function:

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n), \text{ with } I(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

By Some way you update weights by giving more importance to the instances that have been misclassification so far, by showing that these approach actually correspond to sequentially minimizing an esoosnential error.

   - Evaluate: $\epsilon_m = \dfrac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$ and $\alpha_m = \ln\left[\dfrac{1-\epsilon_m}{\epsilon_m}\right]$
   - Update the data weighting coefficients:

$$w_n^{(m+1)} = \underbrace{w_n^{(m)}}_{\uparrow} \exp[\alpha_m \boxed{I(y_m(\mathbf{x}_n) \neq t_n)}]$$

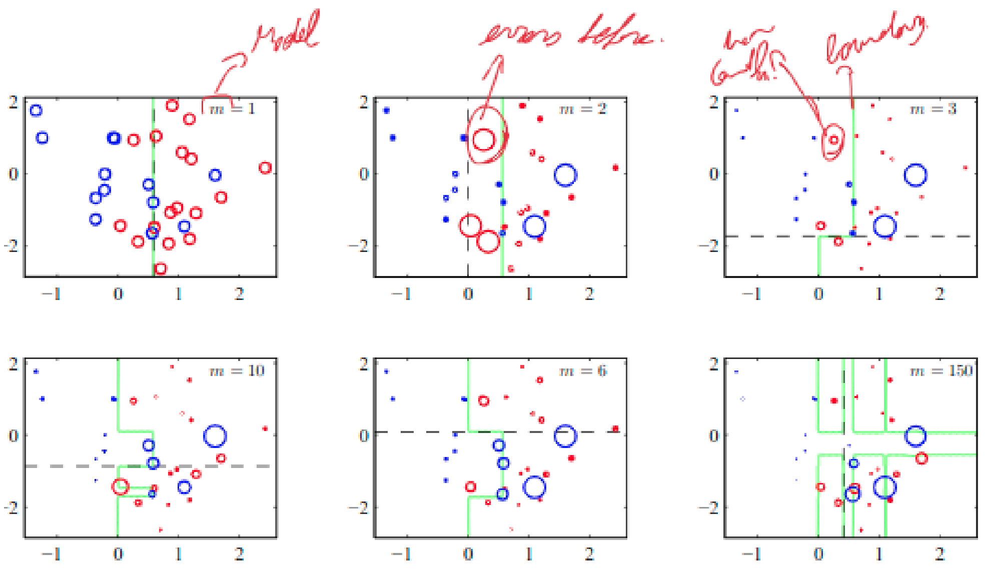
→ [illegible]

Which error

E is 1.

3. Output the final classifier

$$Y_M(\mathbf{x}) = \underline{\text{sign}}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right)$$



## 13.4 Exponential error minimization

AdaBoost can be explained as the sequential minimization of anexponential error function.

$$E = \sum_{n=1}^{N} \exp[-t_n f_M(\mathbf{x}_n)],$$

**Goal**: minimize E

$$f_M(\mathbf{x}) = \frac{1}{2}\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x}), \quad t_n \in \{-1, +1\}$$

**Sequential minimization.** Instead of minimizing $E$ globally

- assume $y_1(\mathbf{x}), \dots, y_{M-1}(\mathbf{x})$ and $\alpha_1, \dots, \alpha_{M-1}$ fixed;
- minimize w.r.t. $y_M(\mathbf{x})$ and $\alpha_M$.

Making $y_M(\mathbf{x})$ and $\alpha_M$ explicit we have:

$$\begin{aligned} E &= \sum_{n=1}^{N} \exp\left[-t_n f_{M-1}(\mathbf{x}_n) - \frac{1}{2} t_n \alpha_M y_M(\mathbf{x}_n)\right] \\ &= \sum_{n=1}^{N} w_n^{(M)} \exp\left[-\frac{1}{2} t_n \alpha_M y_M(\mathbf{x}_n)\right], \end{aligned}$$

with $w_n^{(M)} = \exp[-t_n f_{M-1}(\mathbf{x}_n)]$ constant as we are optimizing w.r.t. $\alpha_M$ and $y_M(\mathbf{x})$.

From sequential minimization of $E$, we obtain

$$w_n^{(m+1)} = w_n^{(m)} \exp[\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)] \text{ and } \alpha_m = \ln\left[\frac{1-\epsilon_m}{\epsilon_m}\right]$$

predictions are made with

$$\text{sign}(f_M(\mathbf{x})) = \text{sign}\left(\frac{1}{2}\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right)$$

which is equivalent to

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right)$$

thus proving that AdaBoost minimizes such error function.

**-Advantages:**

fast, simple and easy to program

no prior knowledge about base learner is required

no parameters to tune (except forM)

can be combined with any method for finding base learners

theoretical guarantees given sufficient data and base learners withmoderate accuracy

**-Issues:**

Performance depends on data and the base learners(can fail with insufficient data or when base learners are too weak)

Sensitive to noise