

Decision Trees and Learning Ensembles for Classification/Ranking

293S T. Yang. UCSB, 2022

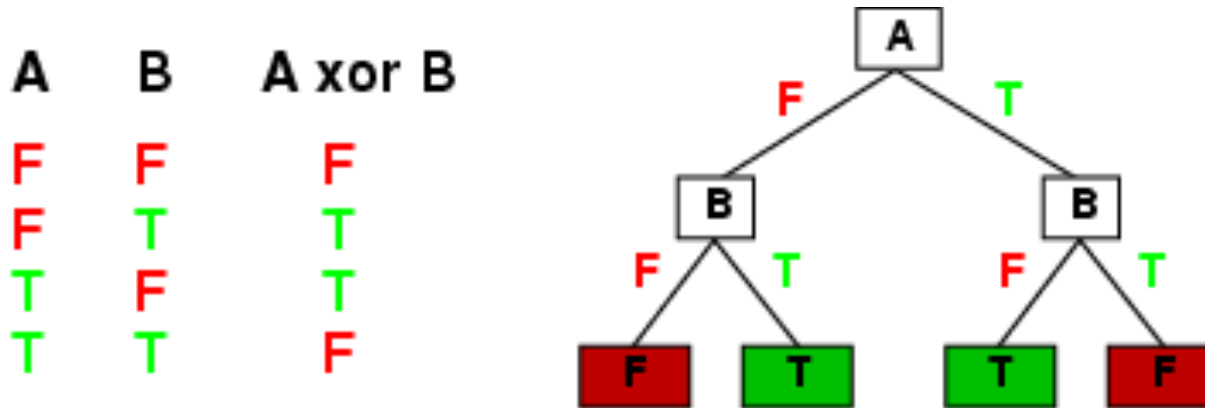
Outlines

- **Example of classification algorithms**
 - Decision trees
- **Learning Ensembles**
 - Random Forest
 - Adaboost
- **Boosting regression trees**

Classification with Decision Trees

Decision Trees

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row \rightarrow path to leaf:



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless f nondeterministic in x) but it probably won't generalize to new examples
- Prefer to find more **compact** decision trees: we don't want to memorize the data, we want to find **structure** in the data!

Decision Trees: Application Example

Problem: decide whether to wait for a table at a restaurant, based on the following **attributes, or called **features****

1. **Alternative**: is there an alternative restaurant nearby?
2. **Bar**: is there a comfortable bar area to wait in?
3. **Fri/Sat**: is today Friday or Saturday?
4. **Hungry**: are we hungry?
5. **Patrons**: number of people in the restaurant (None, Some, Full)
6. **Price**: price range (\$, \$\$, \$\$\$)
7. **Raining**: is it raining outside?
8. **Reservation**: have we made a reservation?
9. **Type**: kind of restaurant (French, Italian, Thai, Burger)
10. **WaitEstimate**: estimated waiting time (0-10, 10-30, 30-60, >60)

Training data: Restaurant waiting

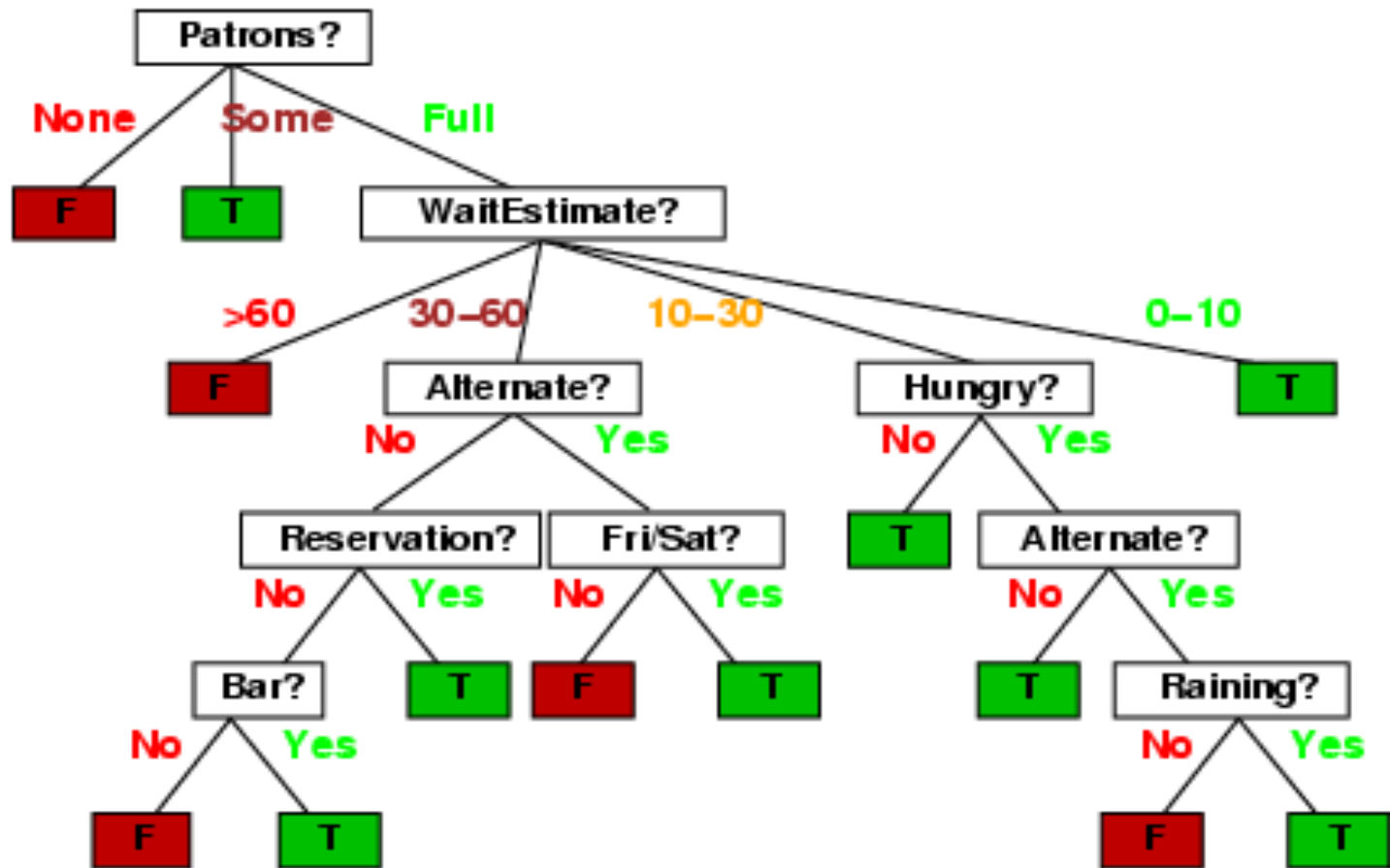
- Examples described by **attribute values or feature value** (Boolean, discrete, continuous)
- **Decision: I will/won't wait for a table:**

Example	Attributes										Target
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>Wait</i>
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0-10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30-60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0-10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10-30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0-10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0-10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0-10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10-30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0-10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30-60	T

- **Classification** of examples is **positive** (T, wait) or **negative** (F, not wait)
- Each training instance is modeled as a feature vector

A decision tree to decide whether to wait

- imagine someone talking a sequence of decisions.



Decision tree learning

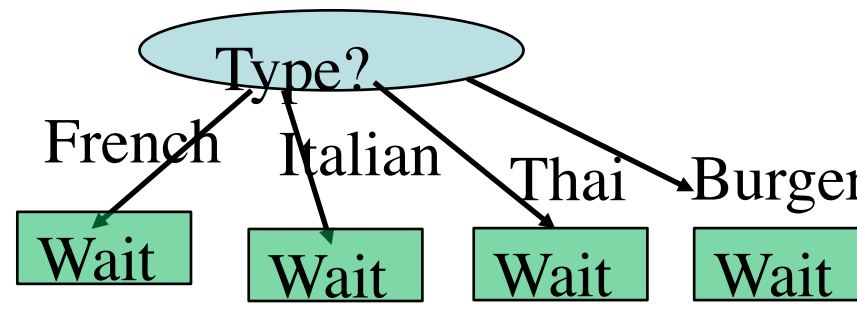
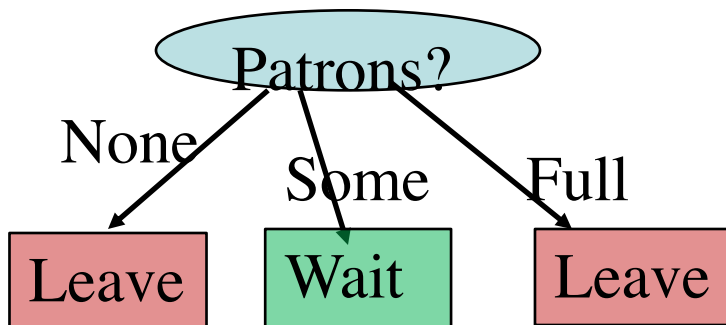
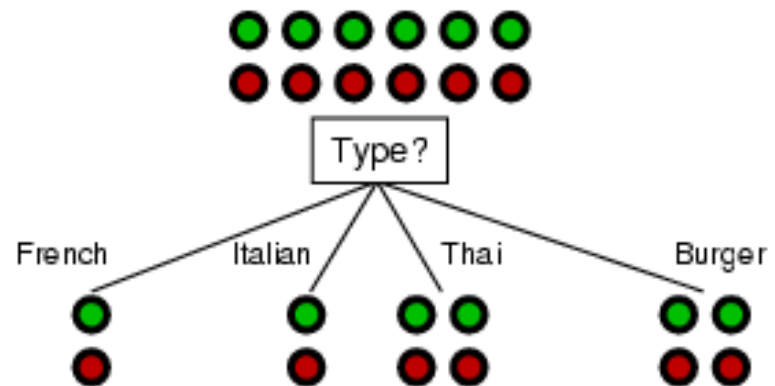
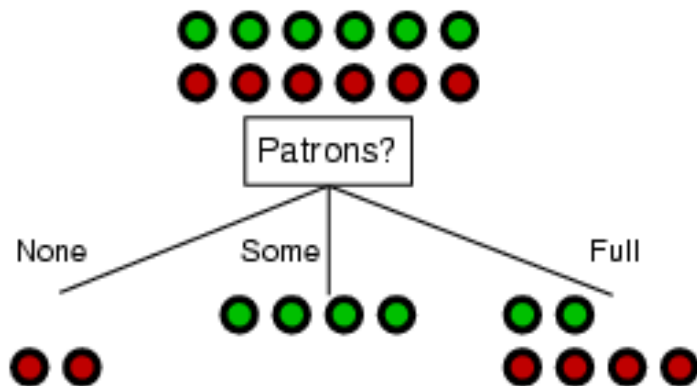
- The goal is to form a tree with the highest classification accuracy
- Test metric: Classification accuracy is the percentage of cases that the derived classifier predicts correctly.

If there are so many possible trees

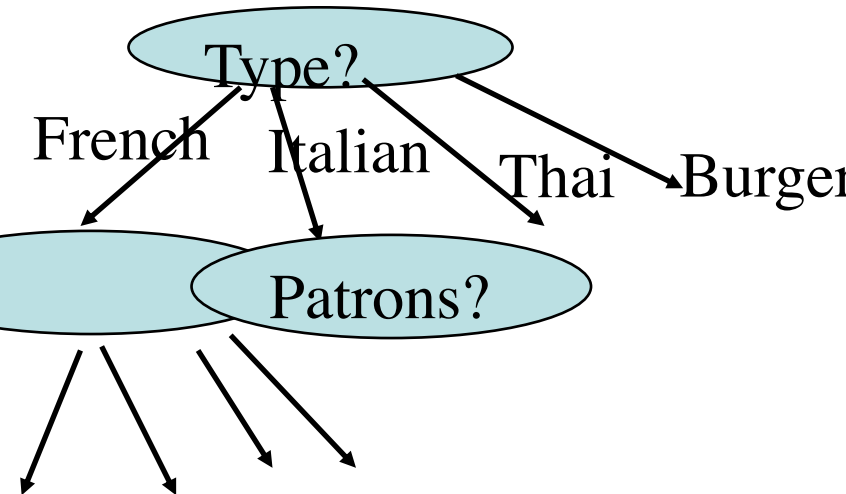
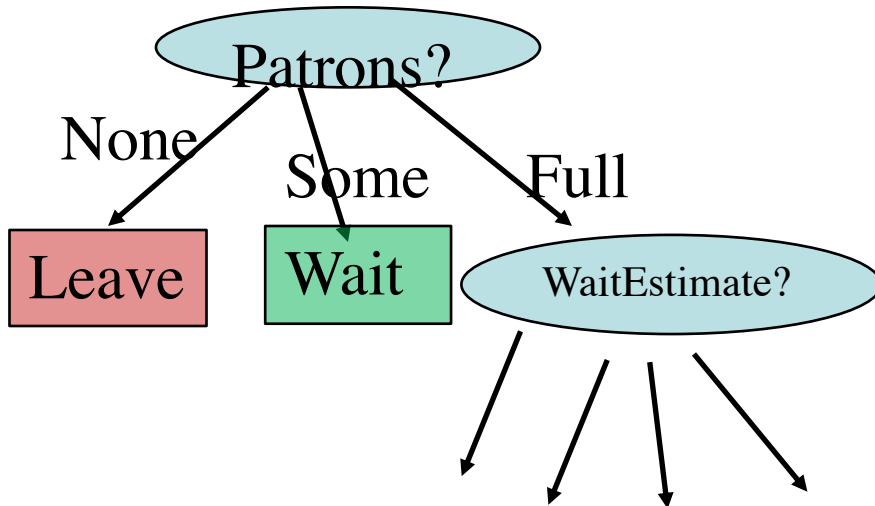
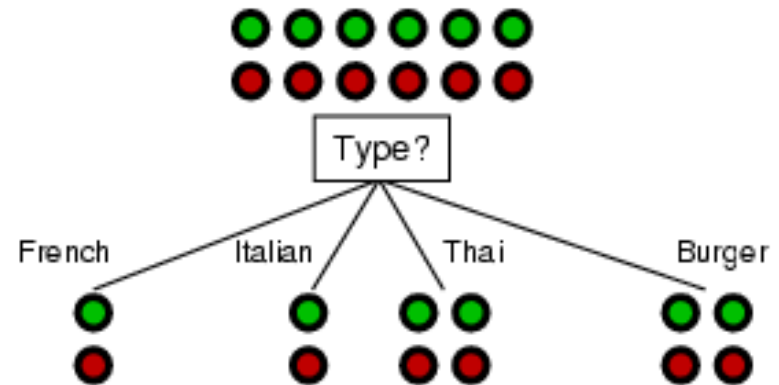
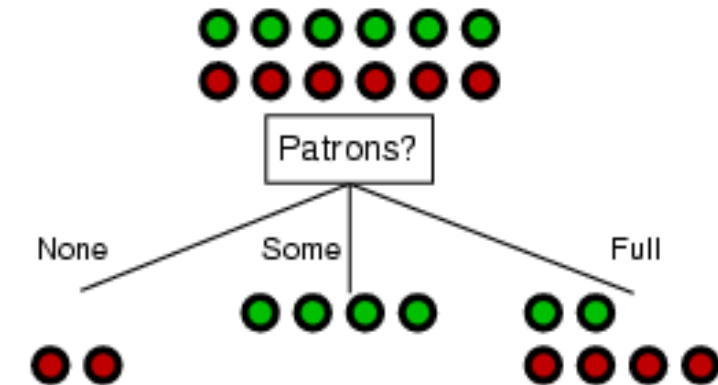
- **Aim**: find a small tree consistent with the training examples
- **Idea**: (recursively) choose "most significant" attribute as root of (sub)tree.

How to build a decision tree

- **Basic idea to form a decision node**
 - Pick up an attribute, and use the value range of this attribute as a branching condition
 - Expand each node by adding more children

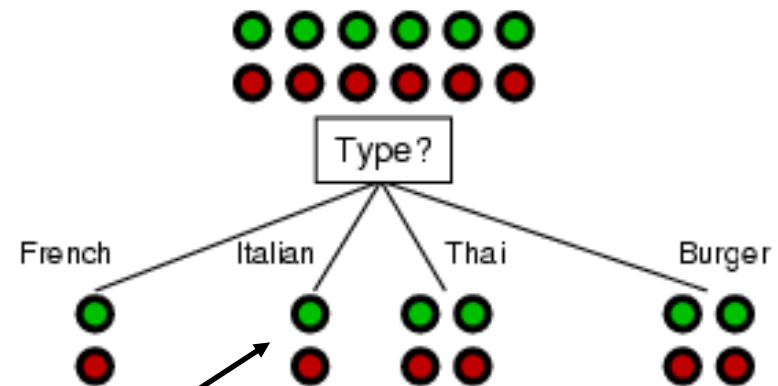
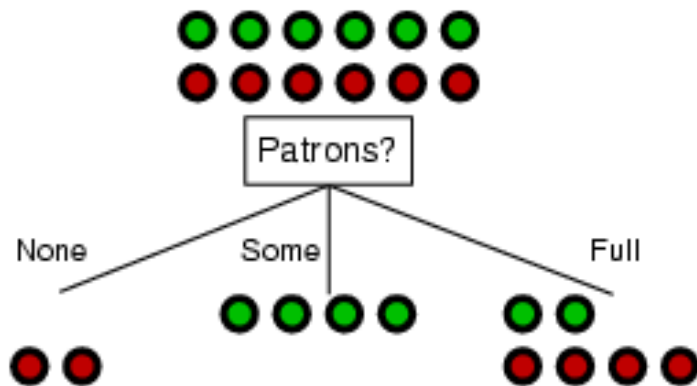


Grow a tree by adding children to some nodes



Choosing an attribute for a smaller tree

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



To wait or not to wait is still at 50%.

- Patrons or type?***
- Probability of being positive is p .
 $P=0.5$ is bad because it does not give any decidable information.
- Need to find a function f to measure uncertainty such that $f(p)$ = bad when $p=0.5$ and $f(p)$ = good when $p=1$ or 0 .

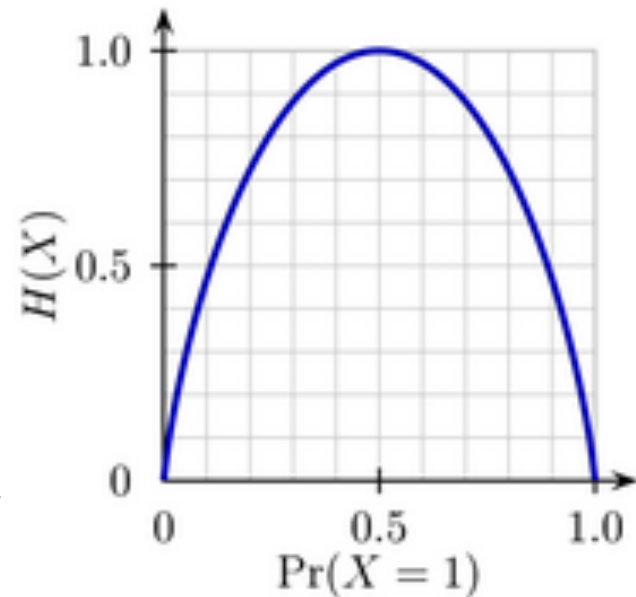
Information theory background: Entropy

- **Entropy** measures uncertainty

$$H(p, 1-p) = -p \log(p) - (1-p) \log(1-p)$$

Consider tossing a biased coin.
If you toss the coin VERY often,
the frequency of heads is, say, p ,
and hence the frequency of tails is
 $1-p$.

Uncertainty (entropy) is zero if $p=0$ or 1
and maximal if we have $p=0.5$.



Using information theory for binary decisions

- Imagine we have p examples which are true (positive) and n examples which are false (negative).
- Our best estimate of true or false is given by:
 - $\text{Prob}(\text{true}) = p/(p+n)$
 - $\text{Prov}(\text{false})=n/(p+n)$
- Hence the entropy is given by:

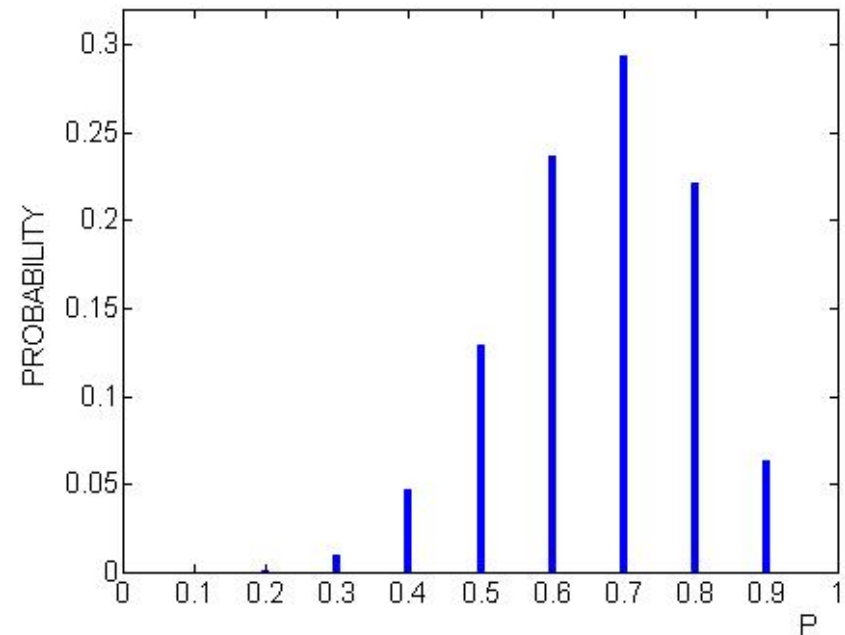
$$\text{Entropy}\left(\frac{p}{p+n}, \frac{n}{p+n}\right) \approx -\frac{p}{p+n} \log \frac{p}{p+n} - \frac{n}{p+n} \log \frac{n}{p+n}$$

Using information theory for more than 2 classes

- **n classes**

$$\sum_{s=1}^n p(s) = 1$$

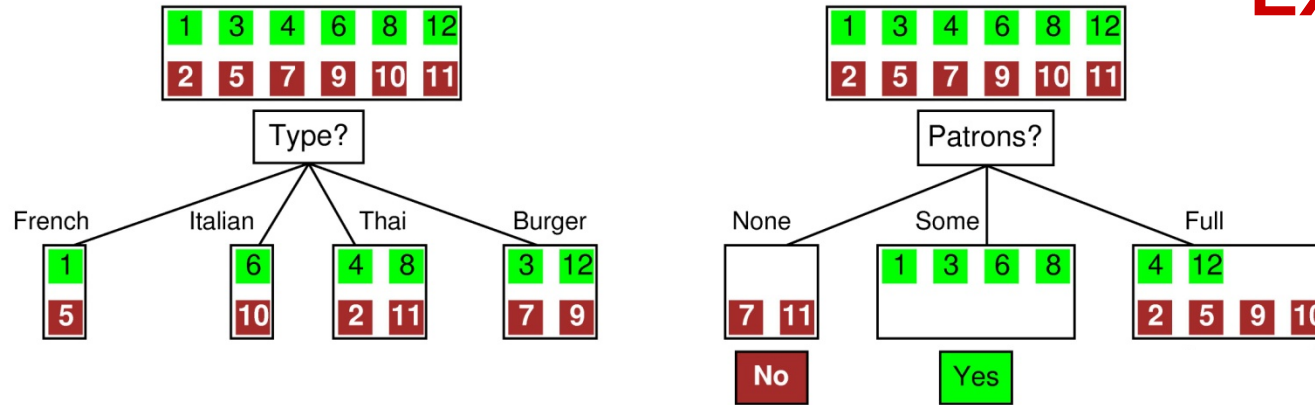
$$\begin{aligned} \text{Entropy}(p) = & -p(s=1)\log[p(s=1)] \\ & -p(s=2)\log[p(s=2)] \\ & \dots \\ & -p(s=n)\log[p(s=n)] \end{aligned}$$



ID3 Algorithm: Using Information Theory to Choose an Attribute

- How much information do we gain if we disclose the value of some attribute?
- ID3 algorithm by Ross Quinlan uses information gained measured by maximum entropy reduction:
 - $IG(A) = \text{uncertainty before} - \text{uncertainty after}$
 - Choose an attribute with the maximum IA

Example



Before: Entropy = $-\frac{1}{2} \log(1/2) - \frac{1}{2} \log(1/2) = \log(2) = 1$ bit:

There is “1 bit of information to be discovered”.

After: for “**Type:**” If we go into branch “French” we have 1 bit, similarly for the others.

} French: 1 bit
 } Italian: 1 bit
 } Thai: 1 bit **On average: 1 bit and gained nothing!**
 Burger: 1 bit

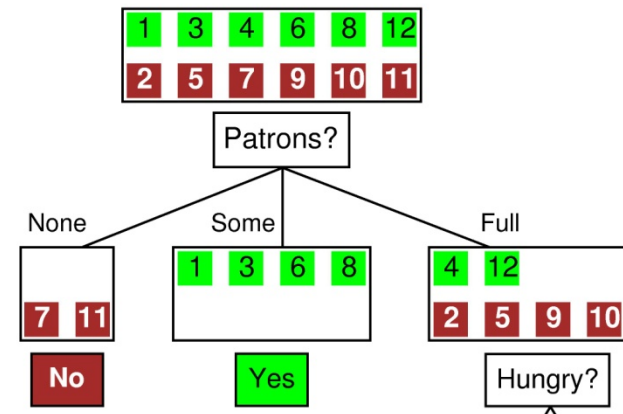
After: for “**Patrons:**” In branch “None” and “Some” entropy = 0!,

In “Full” entropy = $-\frac{1}{3} \log(1/3) - \frac{2}{3} \log(2/3) = 0.92$

So Patrons gains more information!

Information Gain: How to combine branches

- 1/6 of the time we enter “None”, so we weight “None” with 1/6.
Similarly: “Some” has weight: 1/3 and “Full” has weight 1/2.

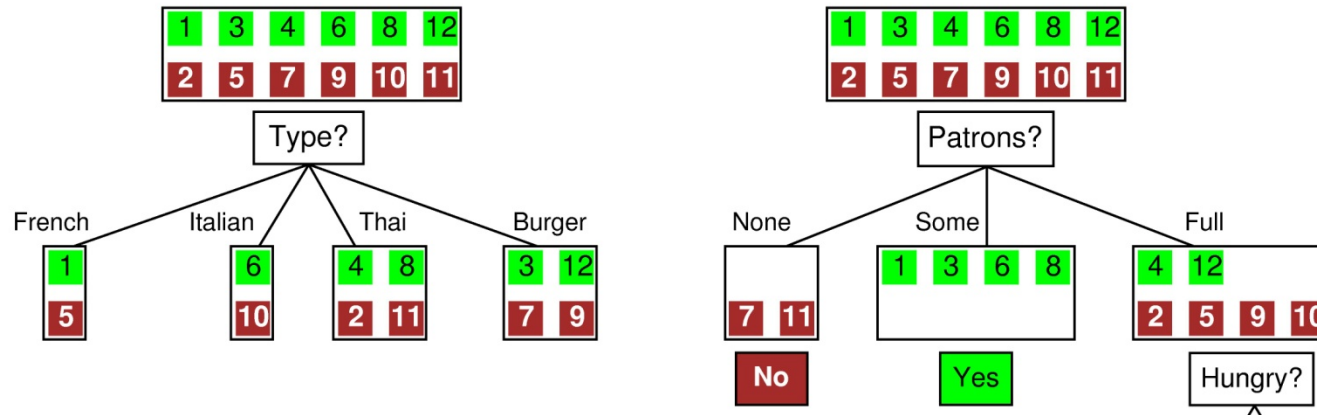


$$Entropy(A) = \sum_{i=1}^n \frac{p_i + n_i}{p + n} Entropy\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

↖
↖

weight for each branch
 entropy for each branch.

Choose an attribute: Restaurant Example



For the training set, $p = n = 6$, before split, $I(6/12, 6/12) = 1$ bit

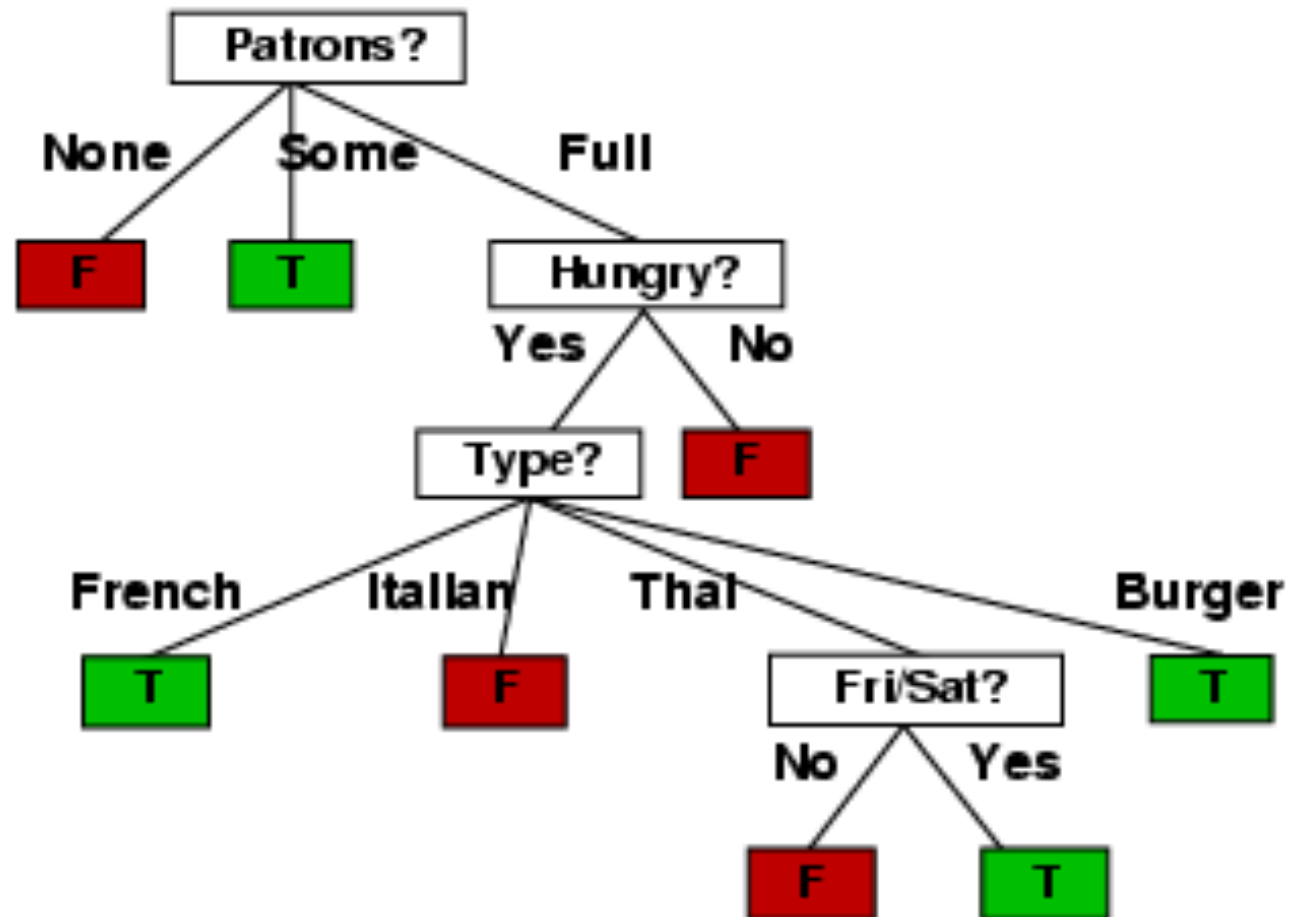
$$IG(Patrons) = 1 - \left[\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] = .0541 \text{ bits}$$

$$IG(Type) = 1 - \left[\frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes and so is chosen by the DTL algorithm as the root

Example: Decision tree learned

- Decision tree learned from the 12 examples:



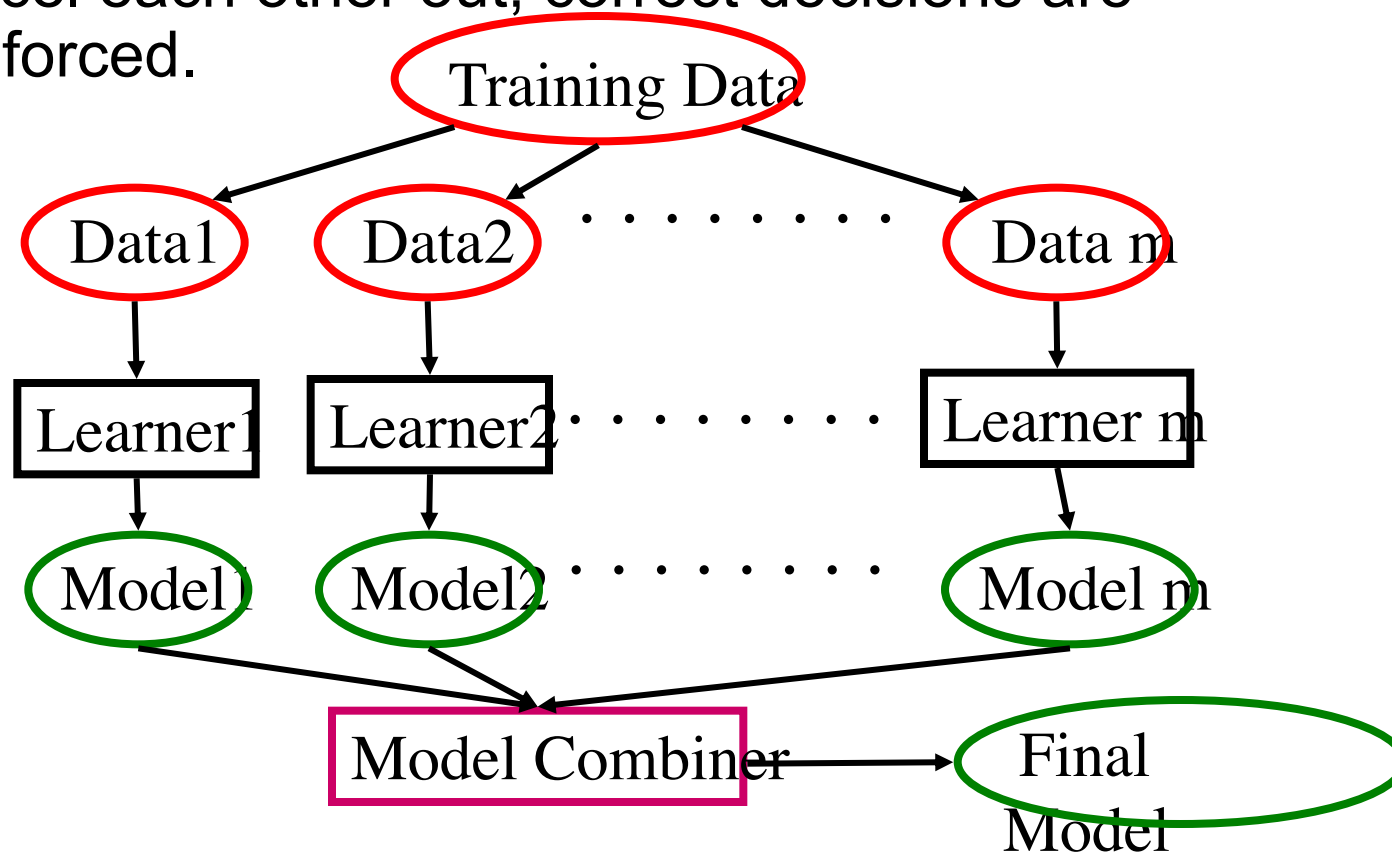
Issues and Discussion

- **When there are no attributes left:**
 - Stop growing and use majority vote.
- **Avoid over-fitting training data**
 - Control tree size with pruning
 - Stop growing a tree earlier
 - Grow first, and prune later.
- **Deal with continuous-valued attributes**
 - Dynamically select thresholds/intervals.
- **Handle missing attribute values**
 - Make up with common values
- **Other tree building methods: Regression with square error loss function**

Learning Ensembles

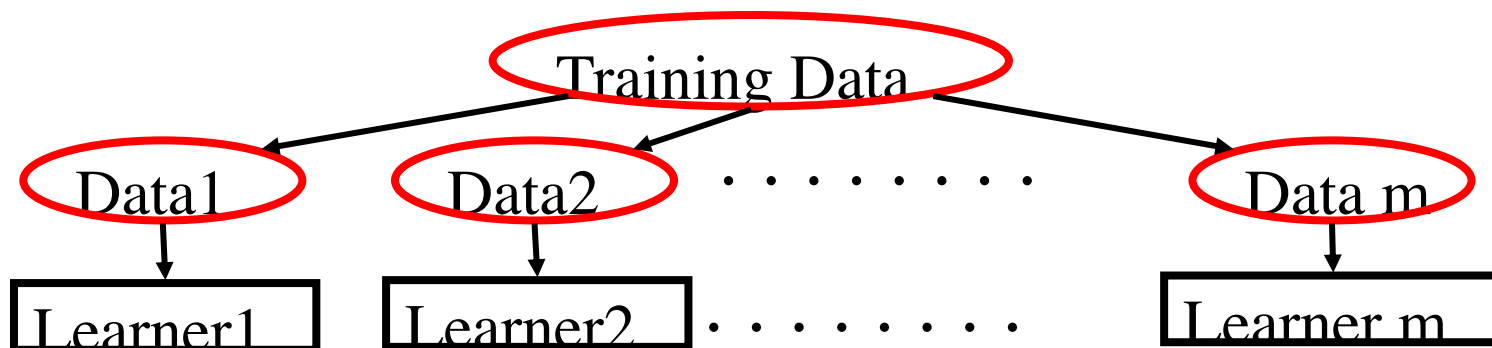
Learning Ensembles

- **Learn multiple classifiers separately**
- **Combine decisions (e.g. using weighted voting)**
- When combining multiple decisions, random errors cancel each other out, correct decisions are reinforced.



Homogenous Ensembles

- **Use a single, arbitrary learning algorithm but manipulate training data to make it learn multiple models.**
 - $\text{Data1} \neq \text{Data2} \neq \dots \neq \text{Data } m$
 - $\text{Learner1} = \text{Learner2} = \dots = \text{Learner } m$
- **Methods for changing training data:**
 - **Bagging:** Resample training data
 - **Boosting:** Reweight training data
 - **DECORATE:** Add additional artificial training data



Bagging

- Create ensembles by repeatedly randomly resampling the training data (Breiman, 1996).
- Given a training set of size n , create m sample sets
 - Each *bootstrap sample set* will on average contain 63.2% of the unique training examples, the rest are replicates.
 - Combine the m resulting models using majority vote
- Advantages:
 - Decreases error by decreasing the variance in the results due to *unstable learners*, algorithms (like decision trees) whose output can change dramatically when the training data is slightly changed.
 - Avoid overfitting training data

Random Forests

- **Introduce two sources of randomness: “Bagging” and “Random input vectors”**
 - Each tree is grown using a bootstrap sample of training data
 - At each node, best split is chosen from random sample of m variables instead of all variables M (features).
 - Final result is aggregated through average or majority voting
- **Advantages:**
 - Good accuracy without over-fitting
 - Fast algorithm (can be faster than growing/pruning a single tree); easily parallelized
 - Handle high dimensional data without much problem

Random Forests

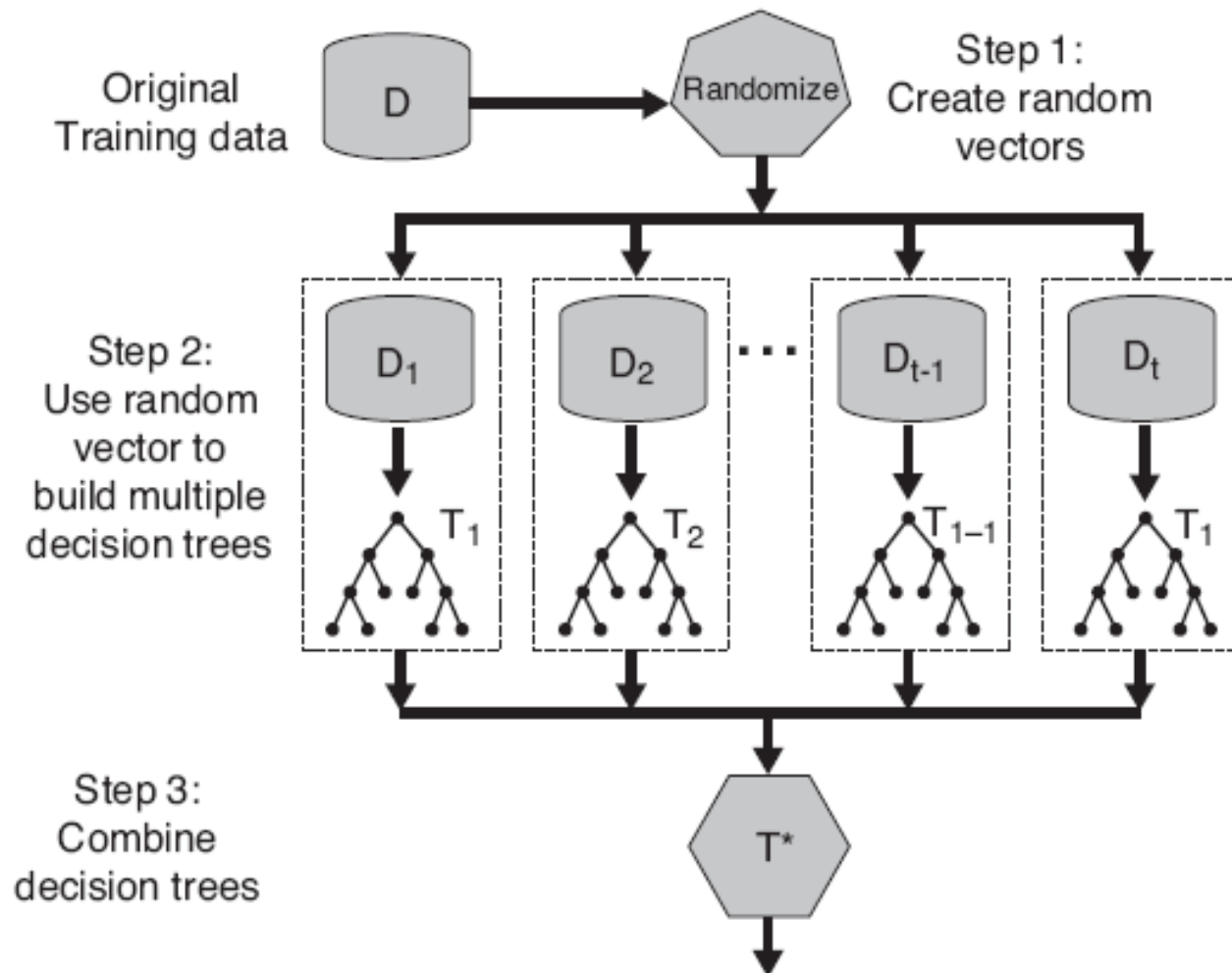


Figure 5.40. Random forests.

Boosting

- Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

- Simple with theoretical foundation

- **Use training set re-weighting**

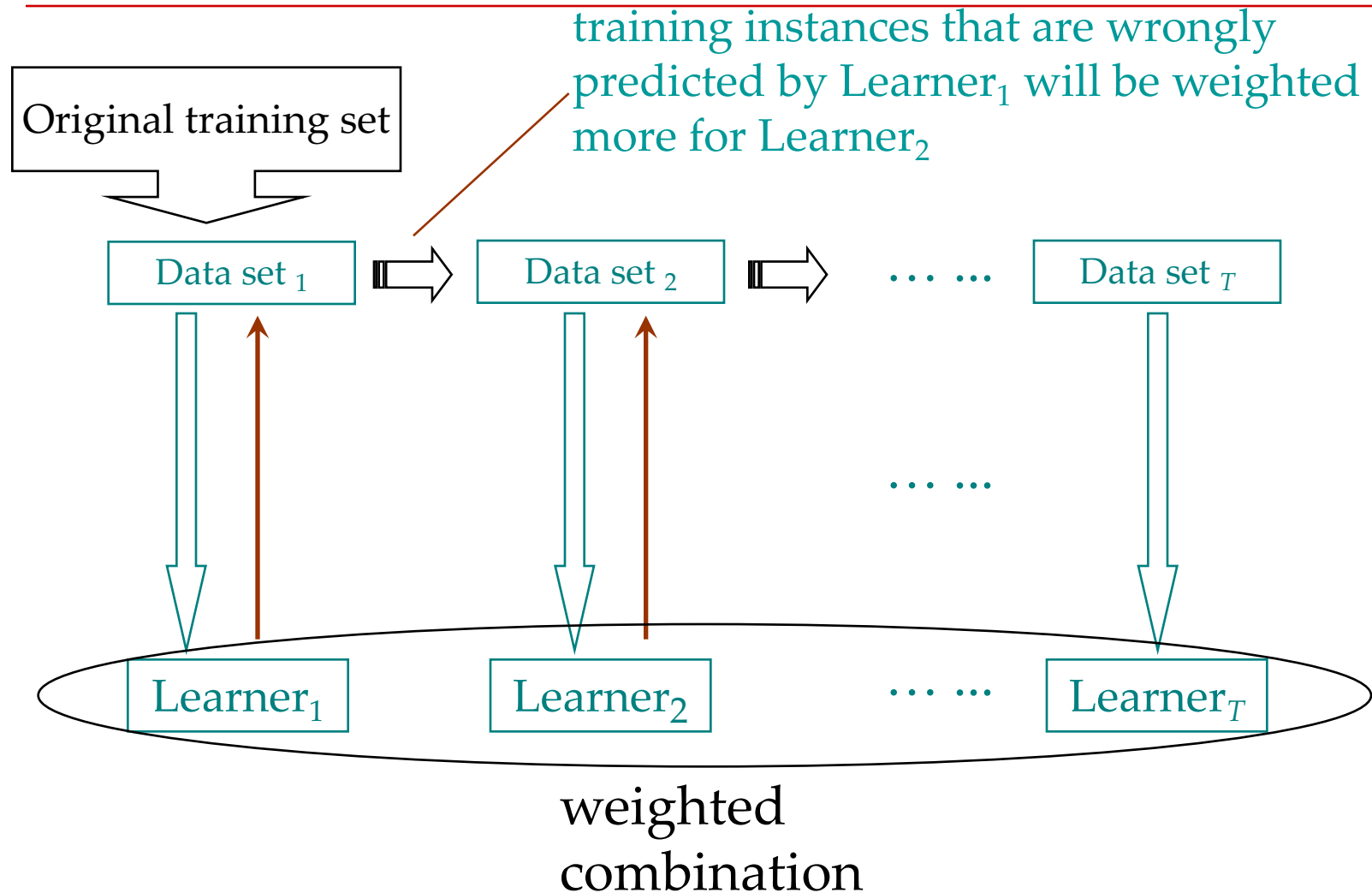
- Each training sample uses a weight to determine the probability of being selected for a training set.

- **AdaBoost** is an algorithm for constructing a “strong” classifier as linear combination of a sequence of “simple” “weak” classifier

$$f(x) = \sum_{t=1}^T \alpha_t h_t(x)$$

- **A weak classifier** is built based on the previous weak classifiers

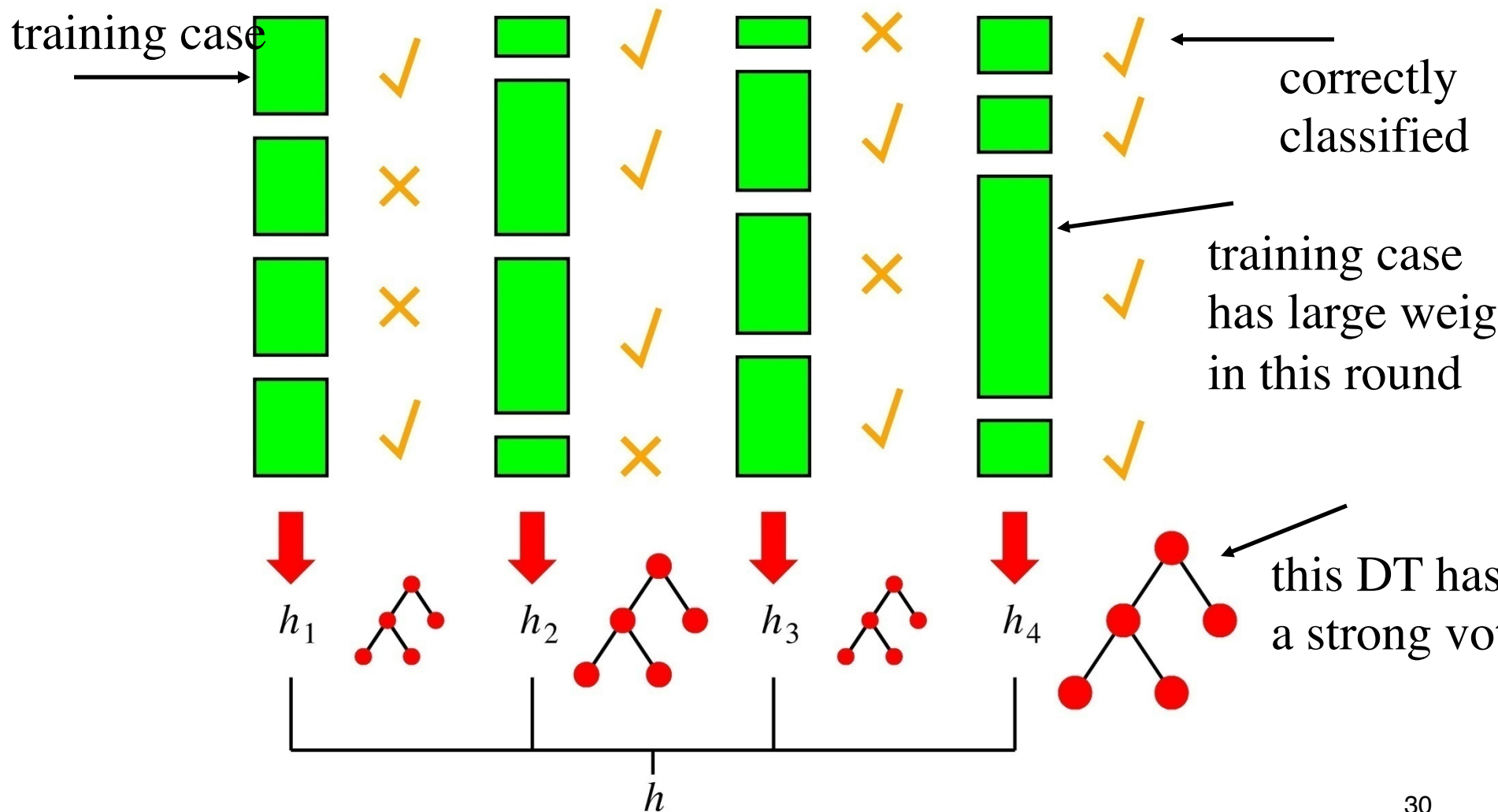
AdaBoost: An Easy Flow



Adaboost Terminology

- $h_t(x)$... “**weak**” or **basis classifier**
 - $< 50\%$ error over any distribution
- $H(x) = \text{sign}(f(x))$... “**strong**” or **final classifier**
 - For binary classification: Positive vs negative
 - thresholded linear combination of weak classifier outputs

And in a Picture



Key idea of AdaBoost

- Given training set $X = \{(x_1, y_1), \dots, (x_m, y_m)\}$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- Initialize distribution $D_1(i) = 1/m$; (weight of training cases)
- for $t = 1, \dots, T$:
 - Find a weak classifier (“rule of thumb”)
 $h_t : X \rightarrow \{-1, +1\}$
with small error ε_t on D_t :
 - Update distribution D_t on $\{1, \dots, m\}$ so that $D_{t+1}(i)$ becomes bigger for wrongly classified cases and smaller for correctly classified cases
 - how about by a factor of $1/\varepsilon_t - 1$
 - how about by a factor of $\ln(1/\varepsilon_t - 1)$
 - how about by a factor of $\sqrt{1/\varepsilon_t - 1}$
- Output final hypothesis

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

AdaBoost.M1

- Given **training set** $X = \{(x_1, y_1), \dots, (x_m, y_m)\}$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- Initialize **distribution** $D_1(i) = 1/m$; (weight of training cases)
- for $t = 1, \dots, T$:

- Find a **weak classifier** (“rule of thumb”)

$$h_t : X \rightarrow \{-1, +1\}$$

with small error ε_t on D_t :

- Update distribution D_t on $\{1, \dots, m\}$. $\alpha_t = 0.5 \ln(1/\varepsilon_t - 1)$

$y_i * h_t(x_i) > 0$, if correct
 $y_i * h_t(x_i) < 0$, if wrong

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

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

Namely sum of $D_{t+1} = 1$

- Output **final hypothesis**

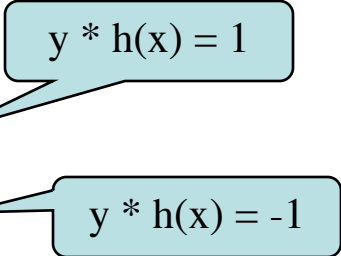
$$H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$$

Reweighting

Effect on the training set

Reweighting formula:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

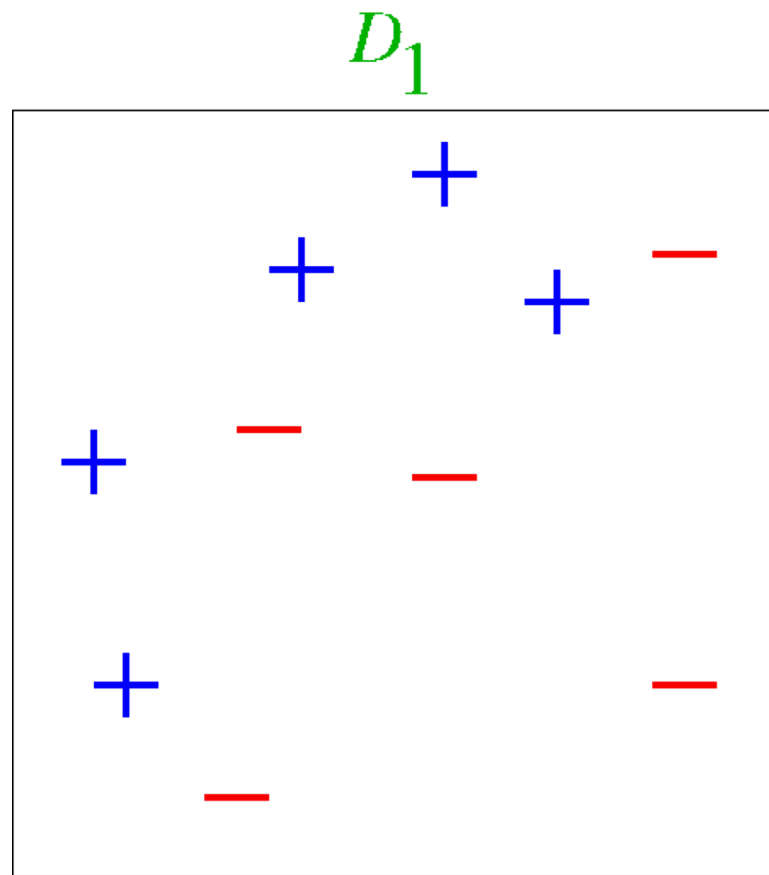
$$\exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y_i = h_t(x_i) \\ > 1, & y_i \neq h_t(x_i) \end{cases}$$


⇒ Increase (decrease) weight of wrongly
(correctly) classified examples

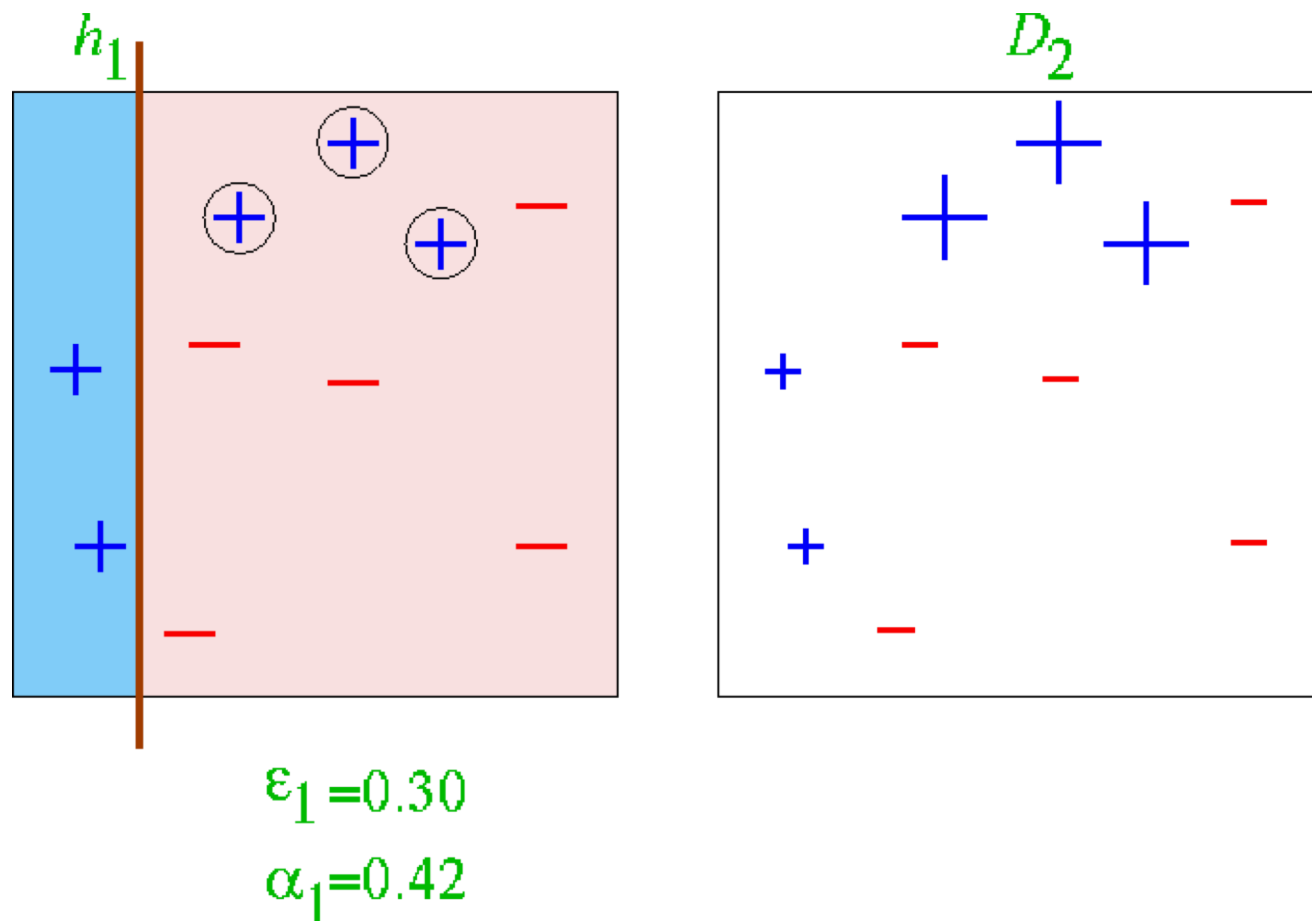
$$\text{Exp}(0.5 \ln(1/\epsilon_t - 1)) = \text{sqrt}(1 / \epsilon_t - 1)$$

$$\text{Exp}(-0.5 \ln(1/\epsilon_t - 1)) = 1 / \text{sqrt}(1 / \epsilon_t - 1)$$

Toy Example



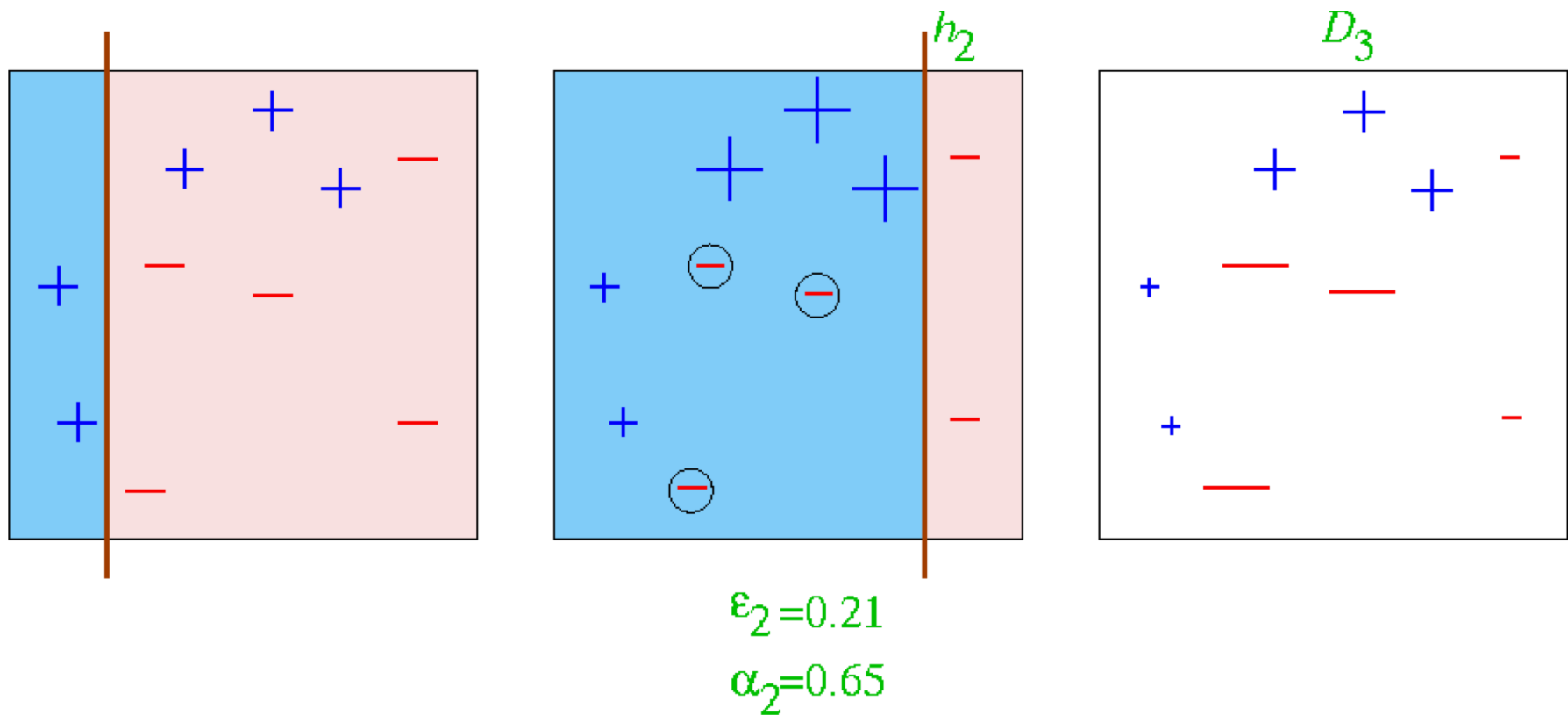
Round 1



Error rate is 30% $\alpha_1 = 0.5 \ln(1/\epsilon_1 - 1) = 0.4236$

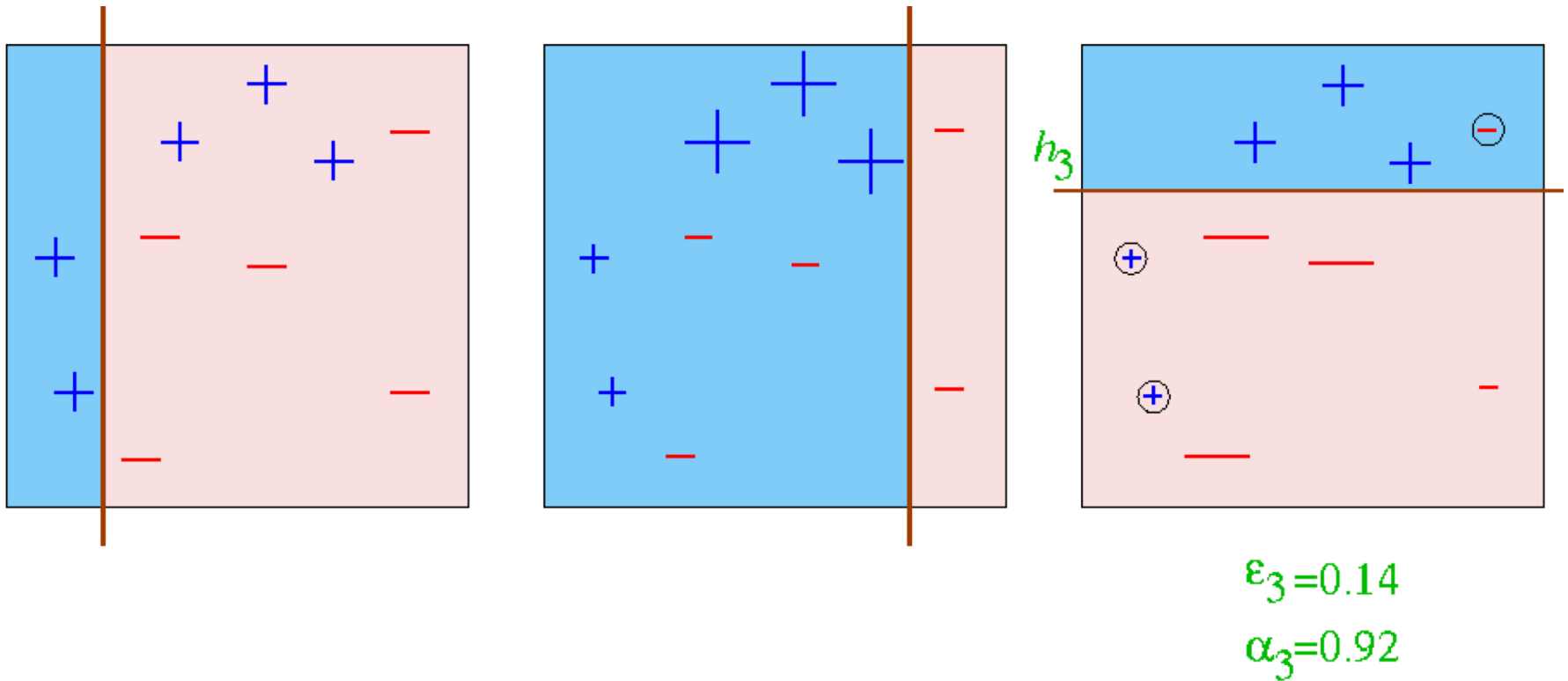
Weak classifier: if $h_1 < 0.2 \rightarrow 1$ else -1

Round 2



Weak classifier: if $h_2 < 0.8 \rightarrow 1$ else -1

Round 3



Weak classifier: if $h_3 > 0.7 \rightarrow 1$ else -1

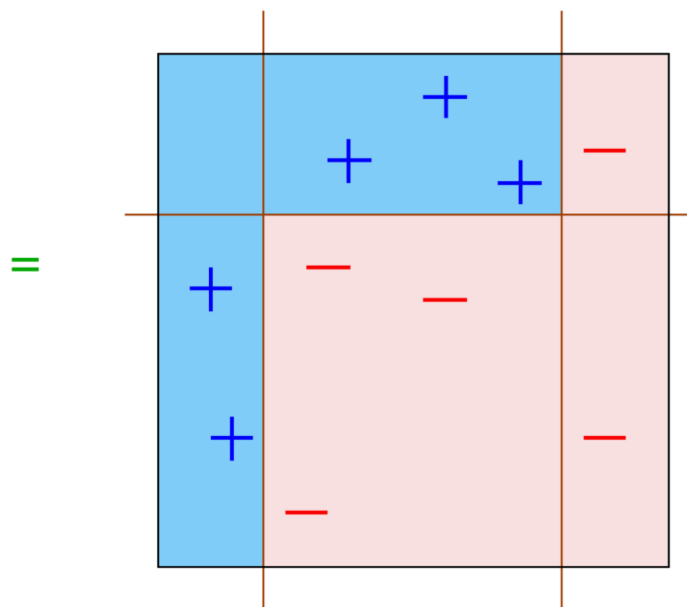
Final Combination

if $h_1 < 0.2 \rightarrow 1$ else -1

if $h_2 < 0.8 \rightarrow 1$ else -1

$$H_{\text{final}} = \text{sign} \left(0.42 \begin{array}{|c|} \hline \text{blue} \\ \hline \text{pink} \end{array} + 0.65 \begin{array}{|c|} \hline \text{blue} \\ \hline \text{pink} \end{array} + 0.92 \begin{array}{|c|} \hline \text{blue} \\ \hline \text{pink} \end{array} \right)$$

if $h_3 > 0.7 \rightarrow 1$ else -1



Pros and Cons of AdaBoost and Extension

Advantages

- Very simple to implement
- Does feature selection resulting in relatively simple classifier
- Fairly good generalization

Disadvantages

- Suboptimal solution
 - Sensitive to noisy data and outliers
-
- **RankBoost** extends AdaBoost for pairwise correctness of document ranking
 - +1: Correctly ordered for a pair of documents
 - -1: Incorrectly ordered

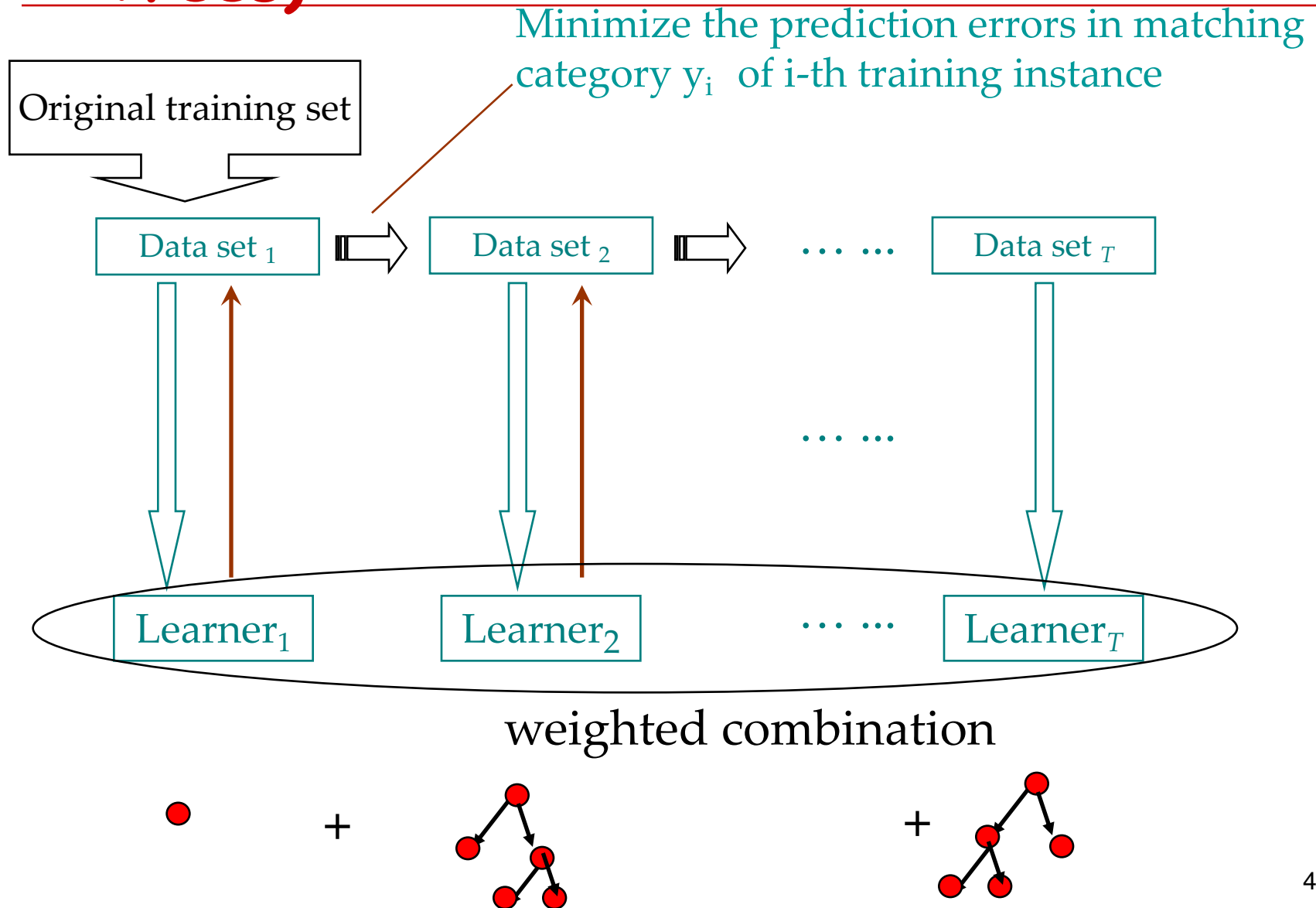
Rank Algorithms and Opensource Library

- **Linear**
 - RankSVM
 - SVM based weight computation
 - As an extension of AdaBoost/AdaRank, AdaRank is optimized for ranking based on NDCG cost metrics
- **Nonlinear Tree Ensembles**
 - GBRT (Gradient Boosting Regression Trees)
 - LambdaMART
 - Additive tree boosting
 - Optimzied based on NDCG
 - RandomForest
 - Bagging on top of GBRT or LambdaMART
- **Opensource Rank Library**
 - RankLib

Some References on Ranking & Boosting

- AdaRank,
 - Jun Xu and Hang Li. 2007. AdaRank: a boosting algorithm for information retrieval. In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '07)
 - Generalization from Adaboost for NDCG optimization
- LambdaMart:
 - C.J.C. Burges, K.M. Svore, P.N. Bennett, A. Pastusiak and Q. Wu, *Learning to Rank Using an Ensemble of Lambda-Gradient Models*. Journal of Machine Learning Research: Workshop and Conference Proceedings, vol. 14, pp. 25-35, 2011
- RandomForest
 - M. Ibrahim and M. Carman. Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank. ACM Transactions on Information Systems (TOIS), 34(4):20, 2016.
- **GBRT:**
 - [A. Mohan, Z. Chen, K. Weinberger, Wearch Ranking with Initialized Gradient Boosted Regression Trees](#) Journal of MLR, 2011

GBRT(Gradient Boosting Regression Trees)



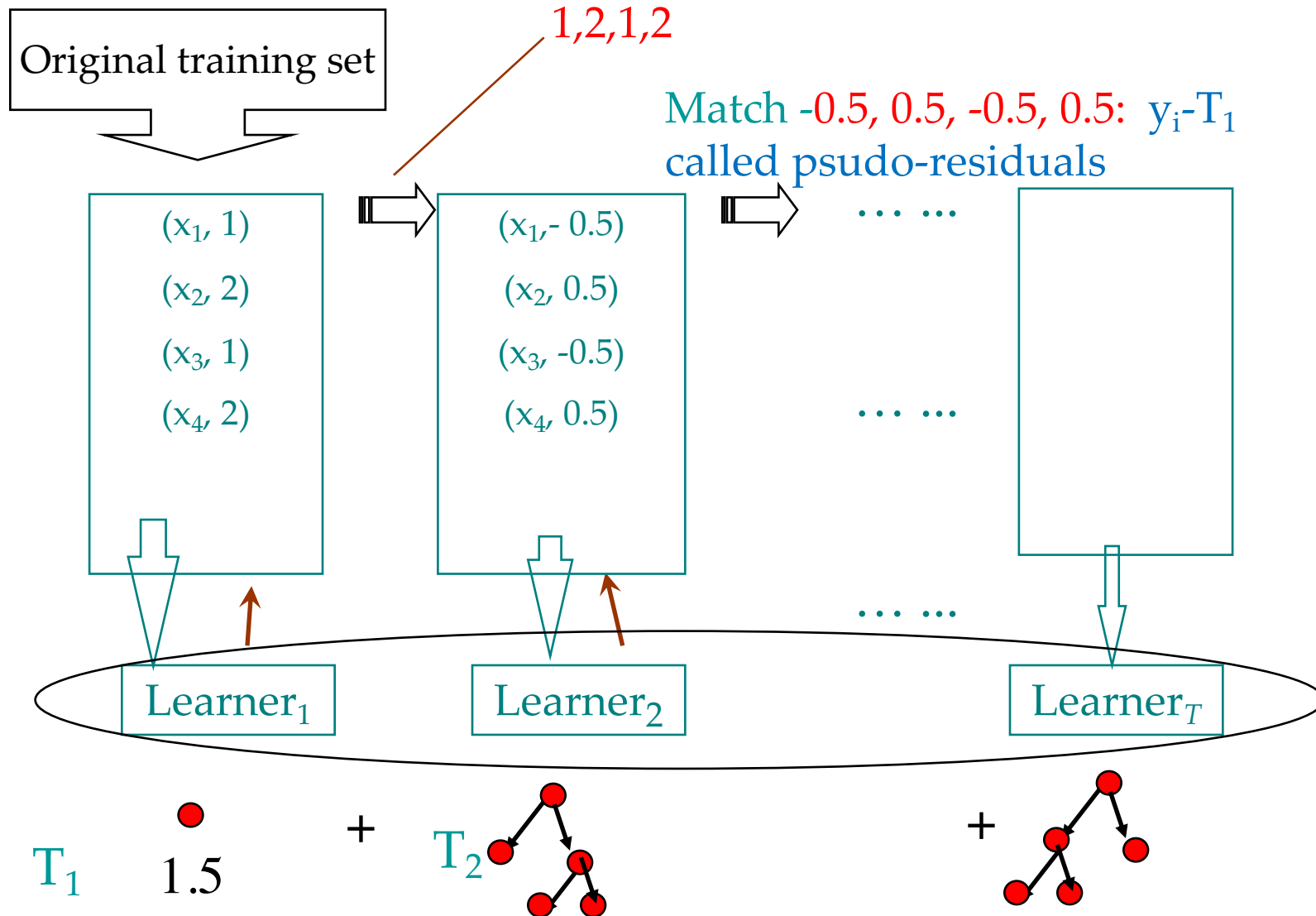
GBRT example with 4 training instances

Learned Function = $T_1 + T_2 + T_3 + \dots$

Minimize prediction errors to match

1,2,1,2

Match -0.5, 0.5, -0.5, 0.5: $y_i - T_1$
called pseudo-residuals



Summary

- **Supervised machine learning techniques**
 - Decision trees, boosting trees, learning ensembles
- **Typically based on hand-crafted features:**
 - Content (e.g. term frequencies, document lengths)
 - Meta-data (e.g.: PageRank scores)
- **Boosting regression trees were popular for ranking and classification in 2018-2018.**
 - LambdaMART
 - Faster than neural nets (machines were slow at that time. Limited GPU support)
- **Weakness**
 - Require hand-crafted features
 - Less effective for handling query-document vocabulary mismatching