
**Ricardo Aler Mur**

- This lecture shows how models can be improved by making ensembles of basic models.

- There are basically two types of ensembles: bagging and boosting (also Stacking).

- Bagging creates ensembles of models by resampling with replacement the original training dataset. This is done many times and a model is trained with each resample.

- Boosting creates models sequentially. Each model in the sequence focuses on the mistakes of the previous model

- There are some variants of Bagging and Boosting when trees are used as base models.

- Random forests is a variant of Bagging, where in addition to using resampling with replacement, different trees in the ensemble are created with different attributes, and each tree uses a much smaller subset of attributes (compared to the total set).

- Gradient Boosted Trees is a variant of Boosting, where trees are used as base models. It is shown how each new model is trained by learning the pseudo-residuals (the difference between the ensemble so far and the actual output). It is also shown that Boosting is prone to overfitting, and care must be taken to avoid it by controlling some of the hyper-parameters.

# *ENSEMBLES OF MODELS*

## *BAGGING & BOOSTING*

# *Ensembles of models*

- **Ensemble** = collection of models (collection of **base models**)
- Main motivation: all models make mistakes, but mistakes of some of the base models can be compensated by successes of the other base models, if the mistakes of the base models are not too correlated (i.e. it is unlikely that all the base models will all fail at the same time)
- They are usually more accurate than single models, even when the base models are "**weak learners**" (not very accurate)
- Main types:
  - Bagging: base models are trained in parallel with the same ML algorithm (e.g. several neural networks). Prediction is decided by majority voting (classification) or averaging (regression).
    - Random Forests: subtype with base model = decision tree
  - Boosting: base models are trained sequentially parallel with the same ML algorithm . Each model focuses in them mistakes of the previous model.
    - Boosted trees: subtype with base model = decision tree
  - Stacking: base models are trained in parallel, but each base model is trained with a different ML algorithm. A meta-model is used to carry out the final prediction (instead of majority voting or averaging)

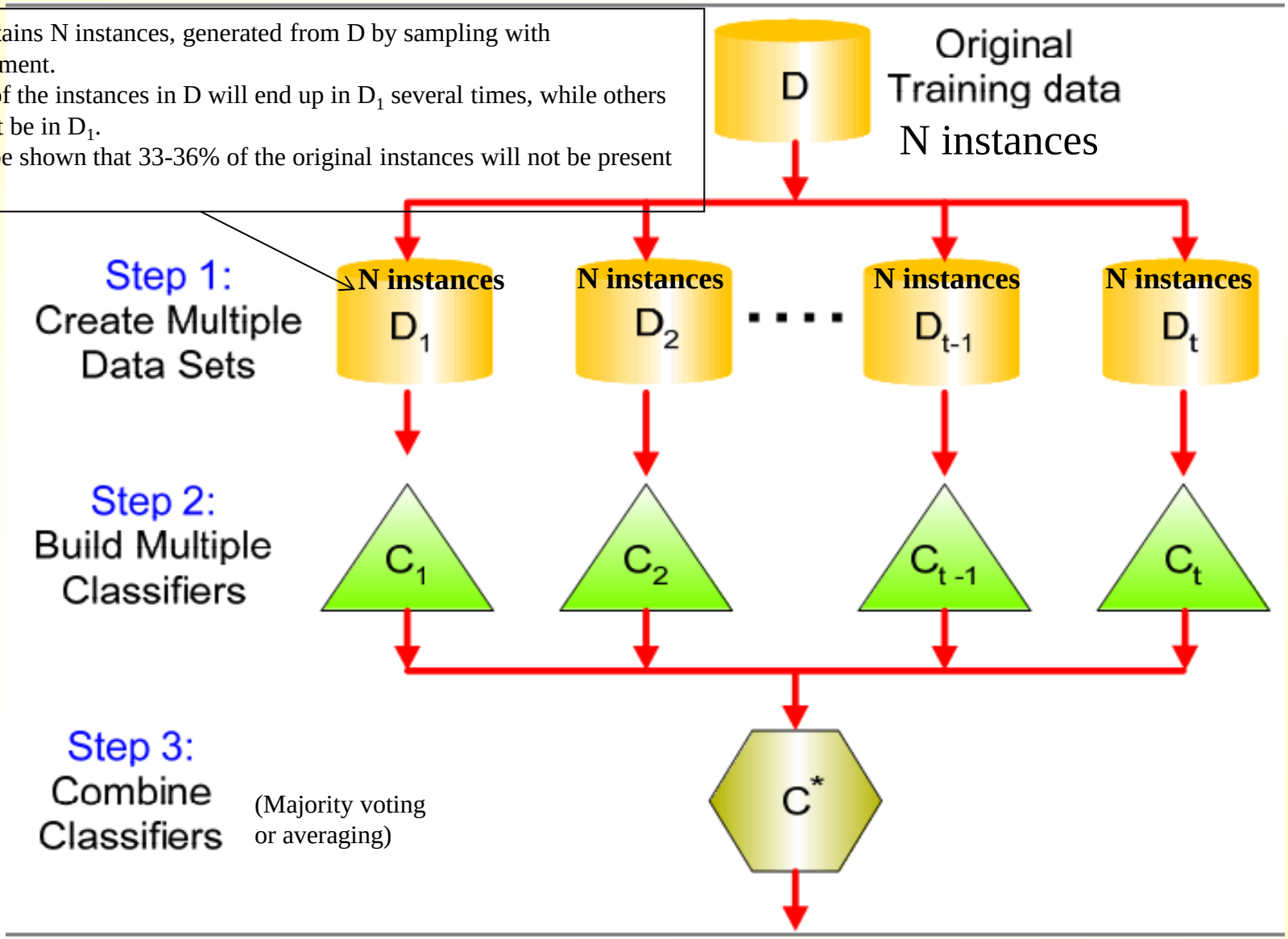
# *BAGGING*

# *Bagging (Bootstrap aggregating)*

- In order to generate an ensemble, Bagging takes advantage of:
  - In general, a ML learning algorithm generates a (slightly) different model if the training data is (slightly) different
  - For the so-called unstable ML algorithms, small differences in the training set cause important differences in the model
    - Unstable: neural networks, decision trees, decision stumps (decision trees with a single node), …
    - Stable: Nearest neighbours (KNN), Support Vector Machines (SVM), …
- Bagging generates lots of training sets and train a different model with each one. Prediction is decided by majority voting (classification) or averaging (regression).
- The collection of training sets is generated from the original available data by means of random sampling with replacement

# Bagging (Bootstrap aggregating)

- $D_1$ contains N instances, generated from D by sampling with replacement.
- Some of the instances in D will end up in $D_1$ several times, while others will not be in $D_1$.
- It can be shown that 33-36% of the original instances will not be present in $D_1$

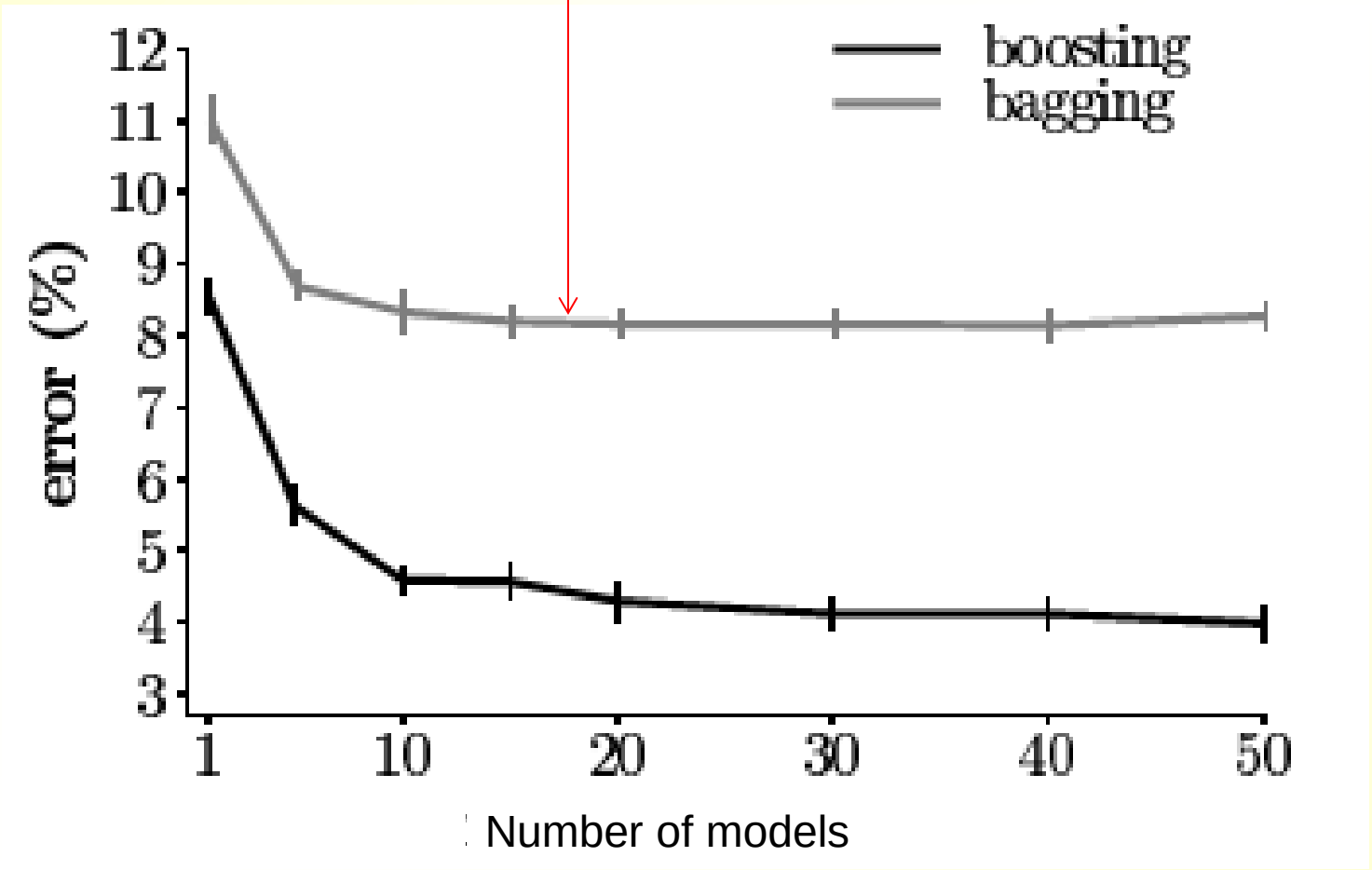


# *Randomization*

- Some of the ML algorithms are stochastic: the same algorithm can generate a (slightly) different model starting from the same training data (for instance, neural networks)
- But even if that is not the case, ML algorithms can be modified to make them stochastic
- Therefore, Randomization can be used to generate an ensemble of models (instead of sampling with replacement)
- We will see that Random Forests do both

# Bagging and error

Typically, the more models, the better (lower) the error



# *Why does it work?*

- Let's suppose that there are 25 classifiers for a two-class classification problem
- Let's suppose that all the classifiers have the same error: ε=0.35 (i.e. they fail 35% of the test instances. Success rate = 65%)
- If mistakes are independent - or not correlated -(i.e. if it is not the case that many classifiers fail at the same time), then the error of the ensemble is 6% (predictions are obtained by majority voting):

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

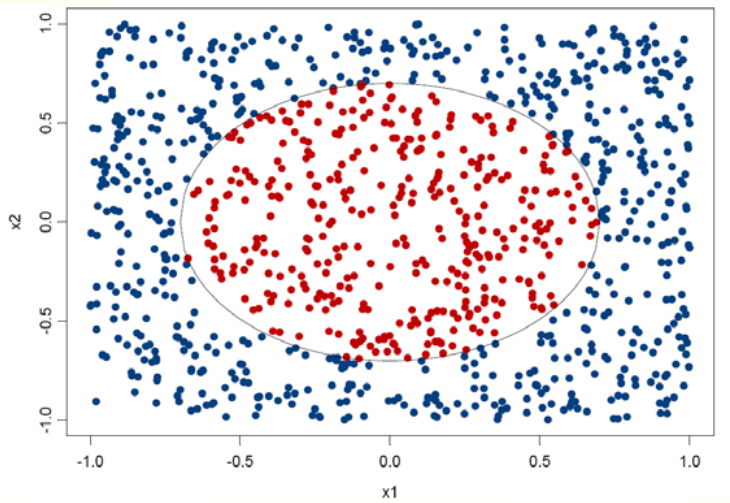
$\varepsilon^i * (1-\varepsilon)^{25-i}$ represents the case where i classifiers fail and 25-i succeed (two-class problem)

- This is the best case, because it is not easy to obtain classifiers whose mistakes are **completely** uncorrelated (independent), even through random sampling or randomization. Uncorrelation can be achieved only to some extent

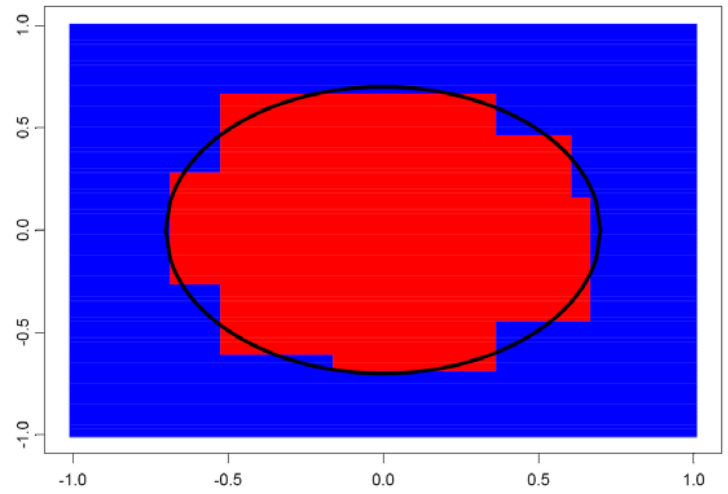
# *Why does it work?*

- A geometric view in instance space: the average of boundaries is more accurate than a single boundary

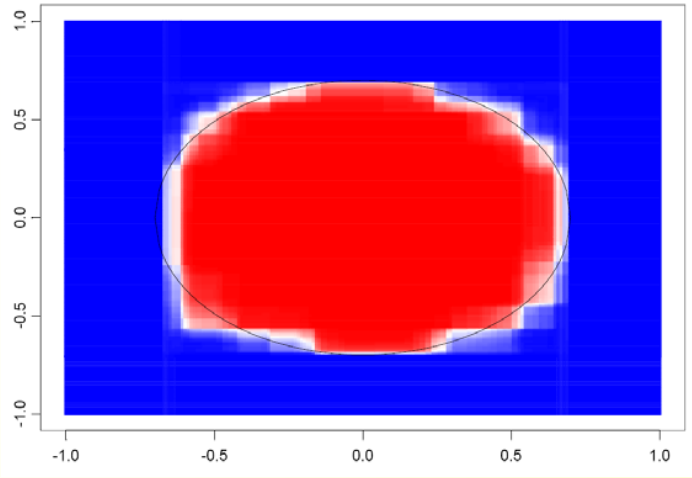
Original data




Decision tree

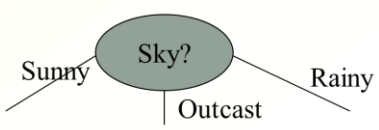


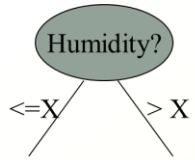

Ensemble of 100 decision trees

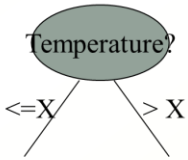


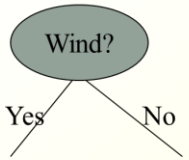
# *Random Forests (RF)*

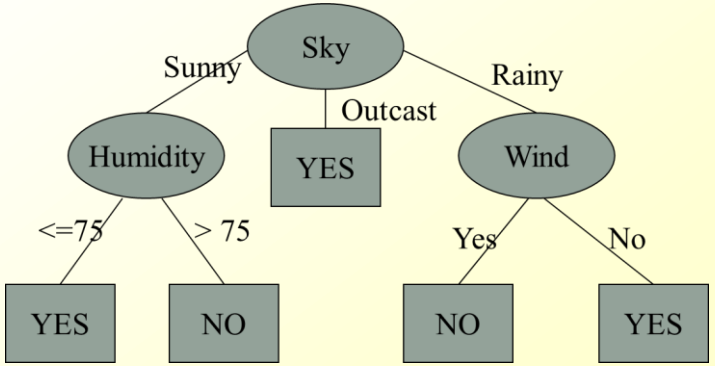
- RF = **Bagging** with decision trees. It uses:
  - Random sampling with replacement
  - Randomization: when choosing an attribute for a node, the best is not selected. Rather, the best is selected from a random subset of m attributes. For instance, if there are M=16 attributes and m=4, then 4 attributes are randomly chosen, and then the best of the four is selected.
    - Typically m=sqrt(M) for classification and m = M/3 for regression.
    - If m == M, then RF = Bagging



# Random Forests

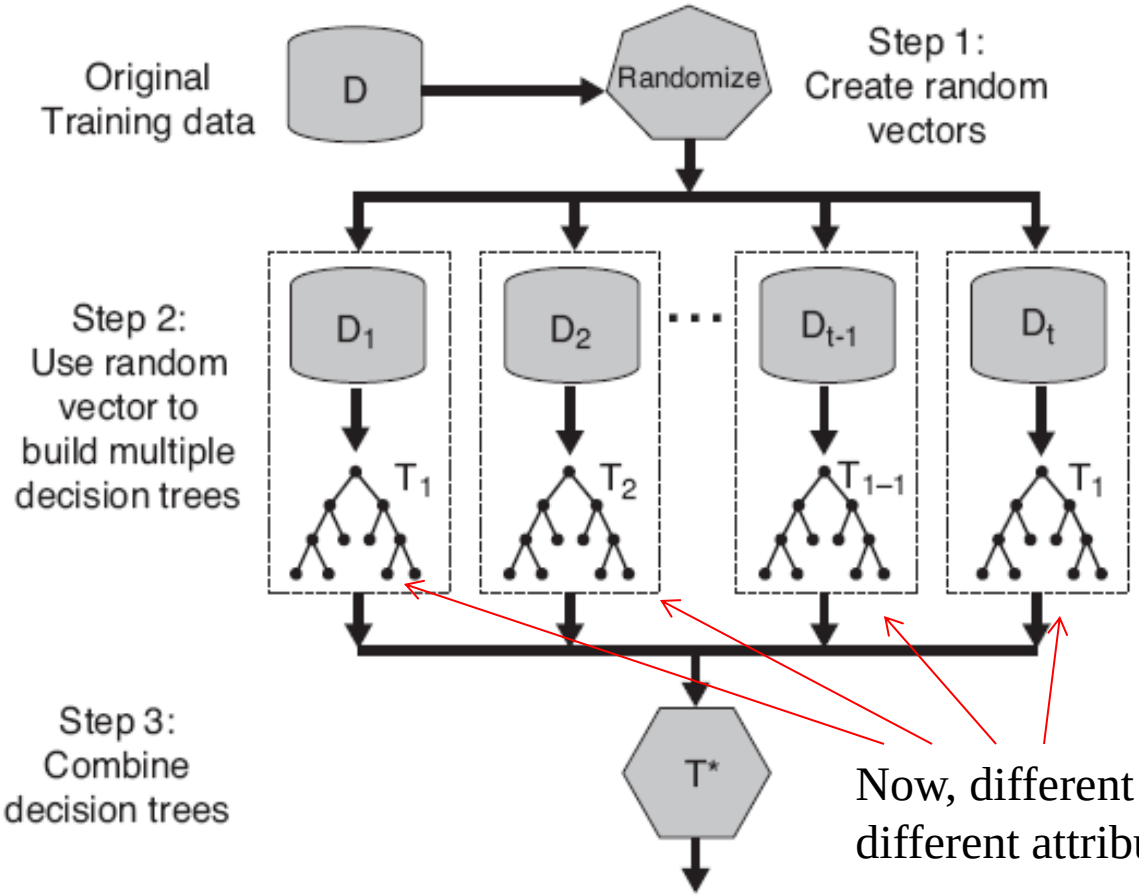



**Figure 5.40.** Random forests.

# *Results of Random Forests*

- It is quite common that RF outperform single decision trees

*Test set misclassification error (%)*

| **Data set** | **Forest** | **Single tree** |
|---|---|---|
| Breast cancer | 2.9 | 5.9 |
| Ionosphere | 5.5 | 11.2 |
| Diabetes | 24.2 | 25.3 |
| Glass | 22.0 | 30.4 |
| Soybean | 5.7 | 8.6 |
| Letters | 3.4 | 12.4 |
| Satellite | 8.6 | 14.8 |
| Shuttle $\times 10^3$ | 7.0 | 62.0 |
| DNA | 3.9 | 6.2 |
| Digit | 6.2 | 17.1 |

# *Random Forests. Out of bag estimation (OOB*

- We already know that in order to estimate the model success rate, it is necessary to use a different test set than the one used for training
- This is typically done by means of train/test (also called hold-out) and also by means of crossvalidation
- But RF are able to provide a success rate estimation without setting aside a test set
- It takes advantage that some instances are not present in some of the training sets $D_i$. Given that they have not been used for training, they can be used for testing the associated tree $T_i$
- The error of instance *x* will be computed by using the trees where *x* was not used for training

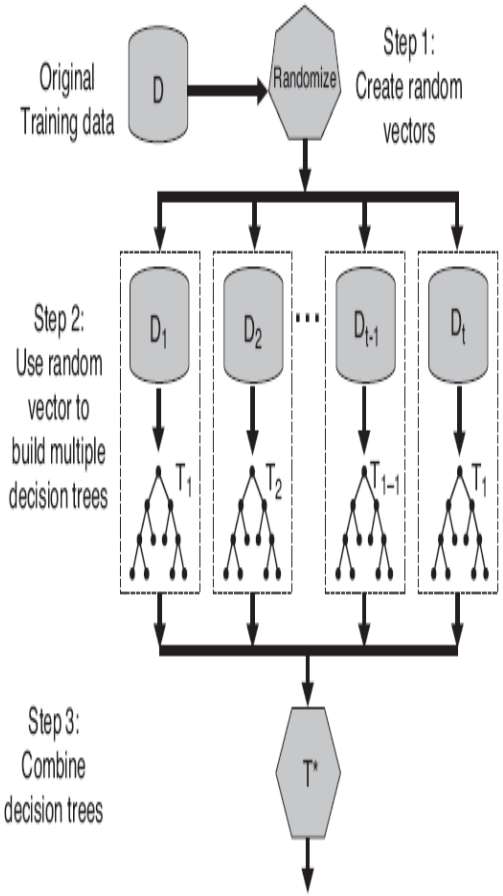



Figure 5.40. Random forests.

https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

# *Random Forests. Ranking attributes*

- Compute out-of-bag error $\hat{e}$
- Let's recall that an attribute $p_k$ is just a column of values in the training data table
- Values of column $p_k$ are shuffled (sorted randomly) and a new out-of-bag error is computed: $\hat{e}p_k$
- Attributes are sorted according to differences $\hat{e}p_k - \hat{e}$. Those with larger differences are more important for prediction

# *Random Forests. Summary*

- Only two new (hyper-)parameters: number of trees in the ensemble (*k*) and size of the attribute subset (*m*)
- Usually it outperforms single decision trees
- Faster than standard Bagging (because only m<<M attributes are considered for each node)
- It provides an estimation success rate called “out-of-bag”: in principle, it is not necessary to do train/test or crossvalidation.
- It ranks attributes (the most relevant first): similarly to attribute selection
- It is able to return probabilistic predictions: if 90 trees say “+” and 10 say “-”, probability of “+” is 0.9

# *BOOSTING*

# *Adaboost (boosting)*

- Like Bagging, Boosting trains different models with different training sets
- But Boosting constructs models sequentially
- We know that the training data is a list of instances (tuples):

  $\{(x_1,y_1), \ldots, (x_a,y_a), \ldots, (x_N,y_N)\}$
- In boosting, every instance *a* has a weight $w_a$. Initially all weights are the same for all instances $w_a=1/N$
- At every iteration, weights change, in order to give more importance to more difficult instances (and contrariwise for easy instances)

# *Adaboost (boosting)*

1. Initially, all training instances use the same weight ($w_a$=1/N)
2. A first classifier $h_0$ is trained. Its training error is $e_0$
3. Repeat while $0<e_i<0.5$
    1. Create a new training set by giving larger weights to difficult instances:
        1. If $h_{i-1}$ fails with $(x_a,y_a)$, increase $w_a = w_a * (1-e_{i-1})/e_{i-1}$
        2. If $h_{i-1}$ succeeds with $(x_a,y_a)$, decrease $w_a = w_a * e_{i-1}/(1-e_{i-1})$
    2. Train a new classifier $h_i$ with the new training set. Its error is $e_i$ (computed on the weighted training set)

The final classifier *f* is a linear combination of all $h_i$. The alpha coefficients depend on the accuracy of $h_i$

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

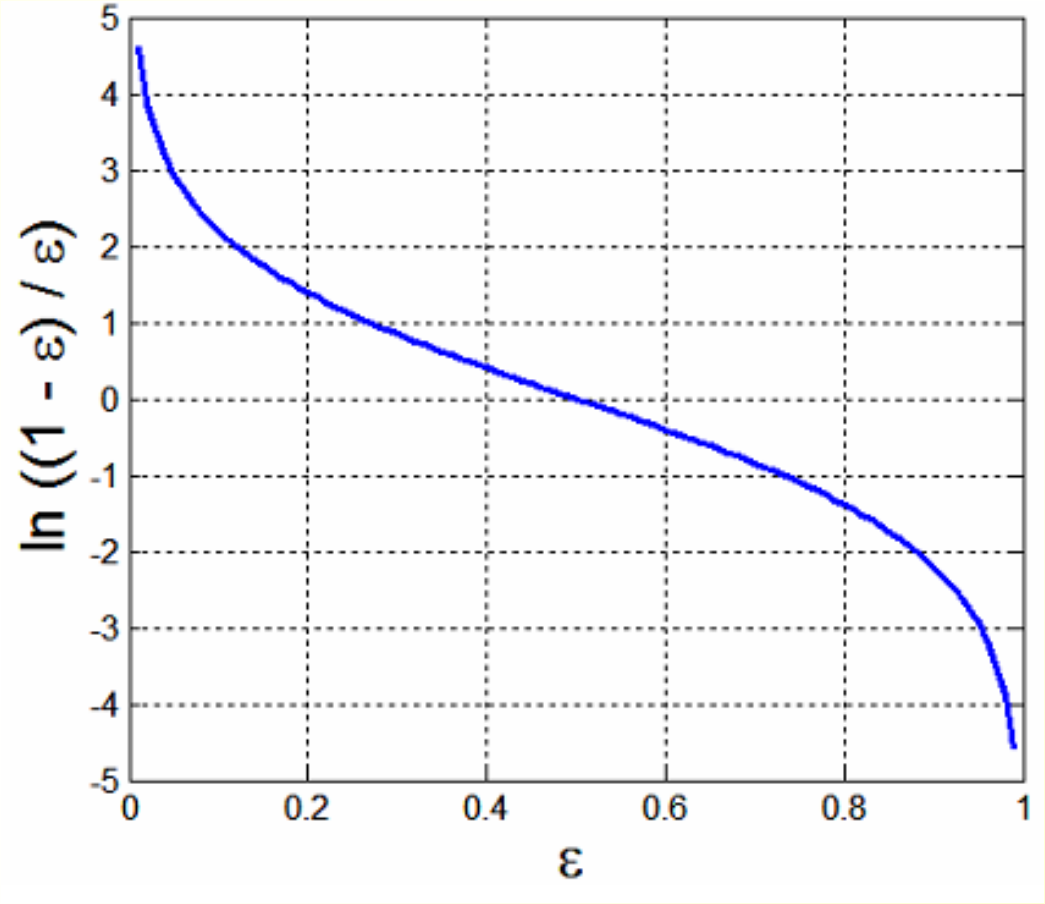
$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$

*Note: if error* $\varepsilon_i$ *small then* $\alpha_i$ *large, if error* $\varepsilon_i$ *large then* $\alpha_i$ *is close to 0*

*Note:* $h_i$ must return either *+1 or -1 (*positive / negative class in two-class problems), or an intermediate value.

# Computing alpha coefficients

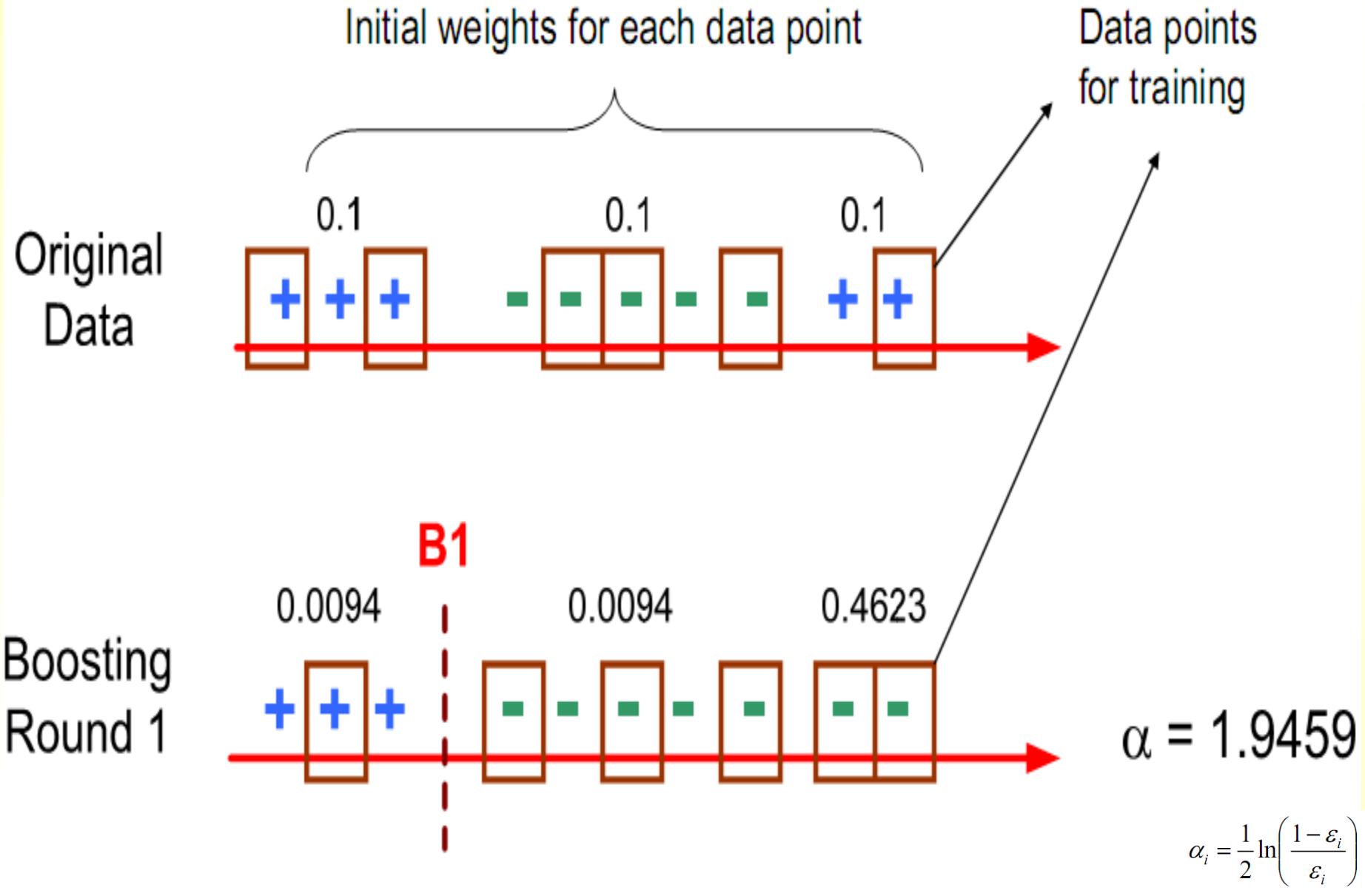
$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

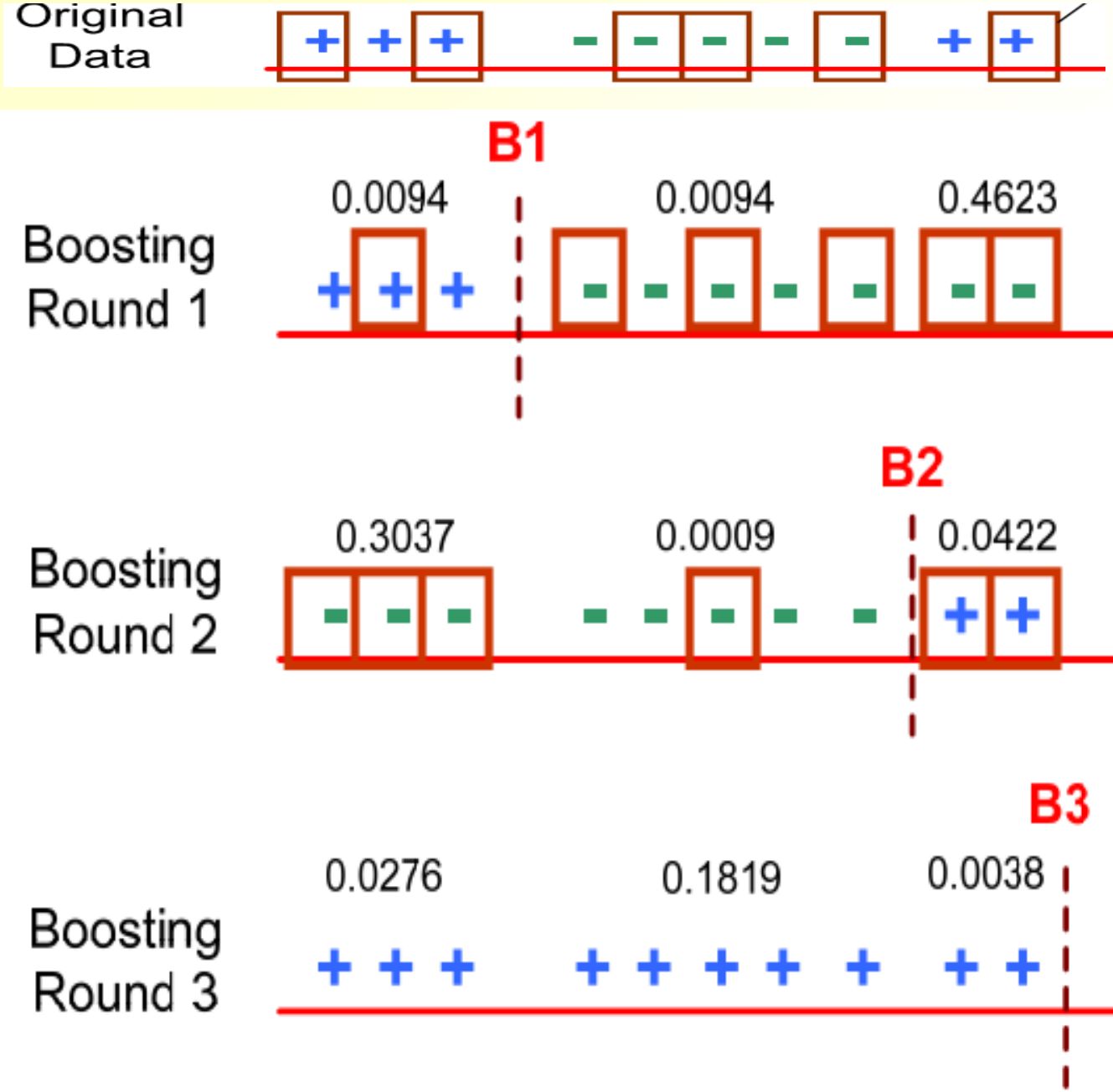


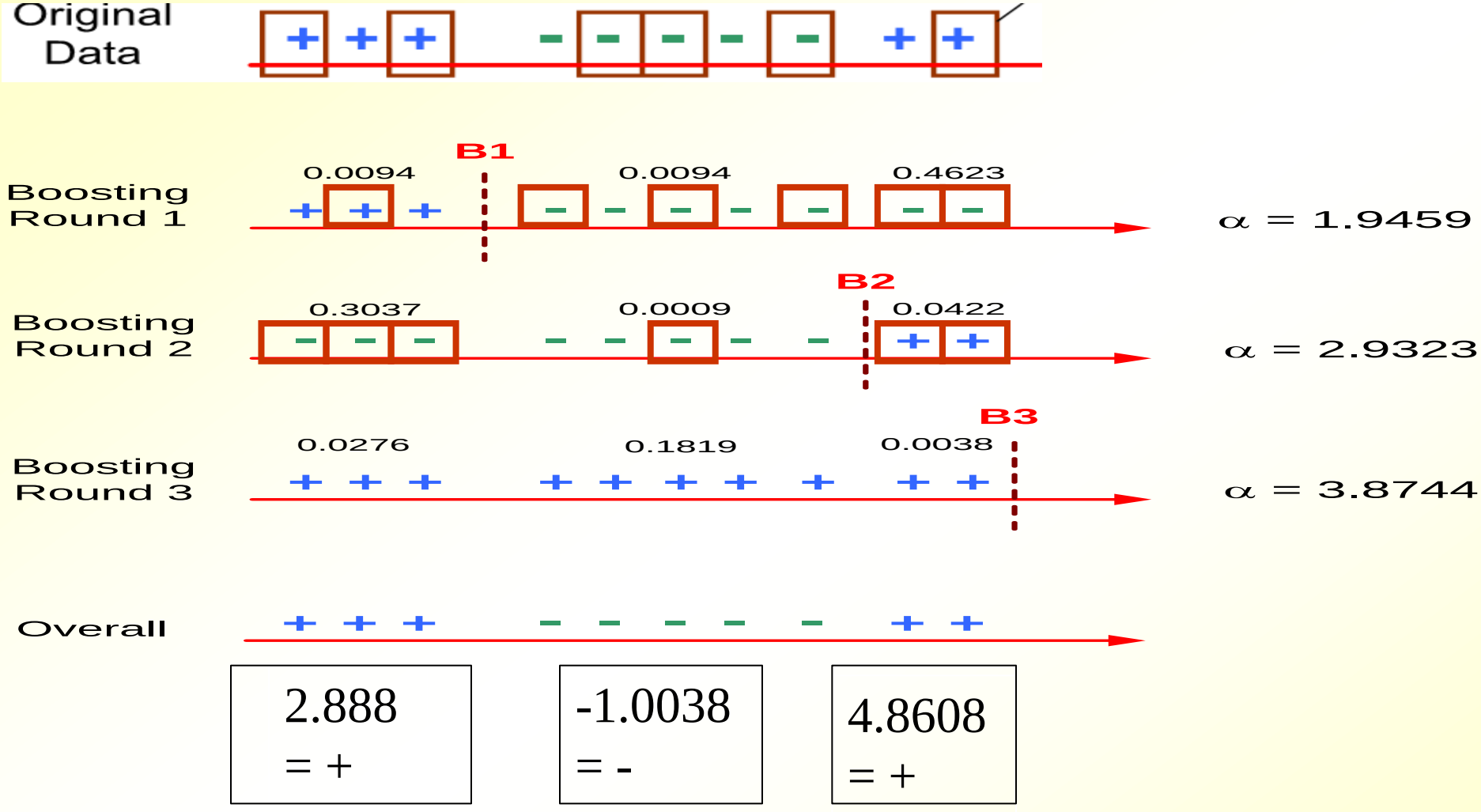
# *Creating the new training set*

- Some ML algorithms are able to use training sets with weights, so weights can be used directly (for instance, decision trees)
- If that is not the case, the new training set can be created by:
  - by multiplying instances according to their weights
  - by randomly sampling the training set. The probability that an instance is selected is proportional to its weight:

$$w_a = w_a \;/\; (w_1+w_2+\ldots+w_N)$$


Initial weights for each data point
Data points for training
Original Data
0.1
0.1
0.1
B1
Boosting Round 1
0.0094
0.0094
0.4623
α = 1.9459
$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$


Original Data
B1
0.0094
0.0094
0.4623
Boosting Round 1
α = 1.9459
B2
0.3037
0.0009
0.0422
Boosting Round 2
α = 2.9323
B3
0.0276
0.1819
0.0038
Boosting Round 3
α = 3.8744



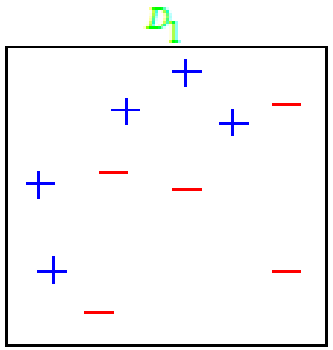

1.9459*B1(x) + 2.9323*B2(x)+ 3.8744*B3(x) = f(x)

1.9459*(+1) + 2.9323*(-1)+ 3.8744*(+1) = 2.888 > 0

1.9459*(-1) + 2.9323*(-1)+ 3.8744*(+1) = -1.0038 < 0

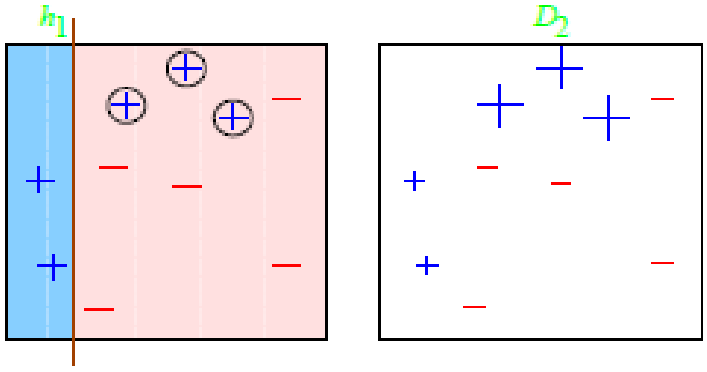
1.9459*(-1) + 2.9323*(+1)+ 3.8744*(+1) = 4.8608 > 0

# Example of boosting in 2D instance space

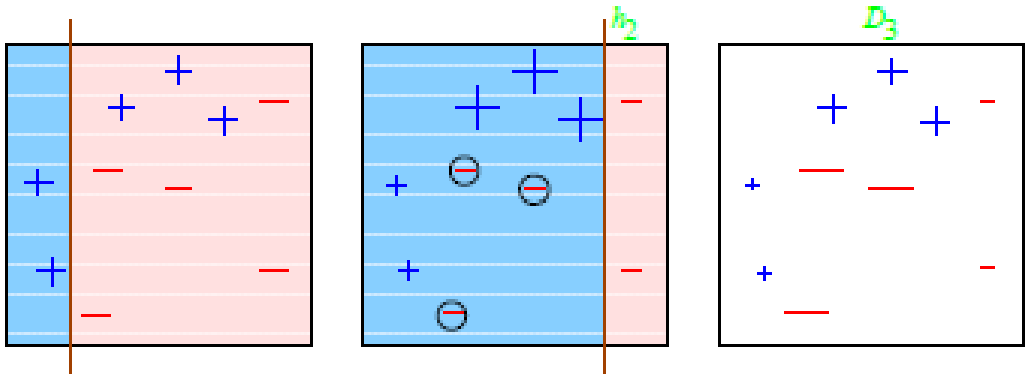



Weak hypotheses == vertical or horizontal half-planes

Round 1



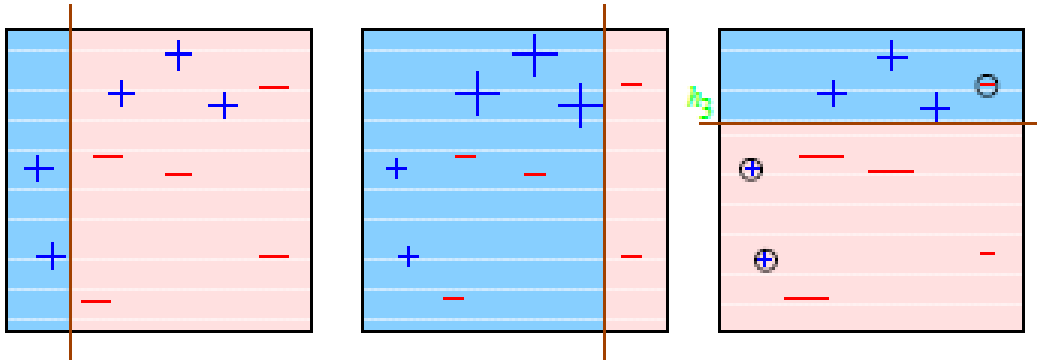
## Round 2




$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

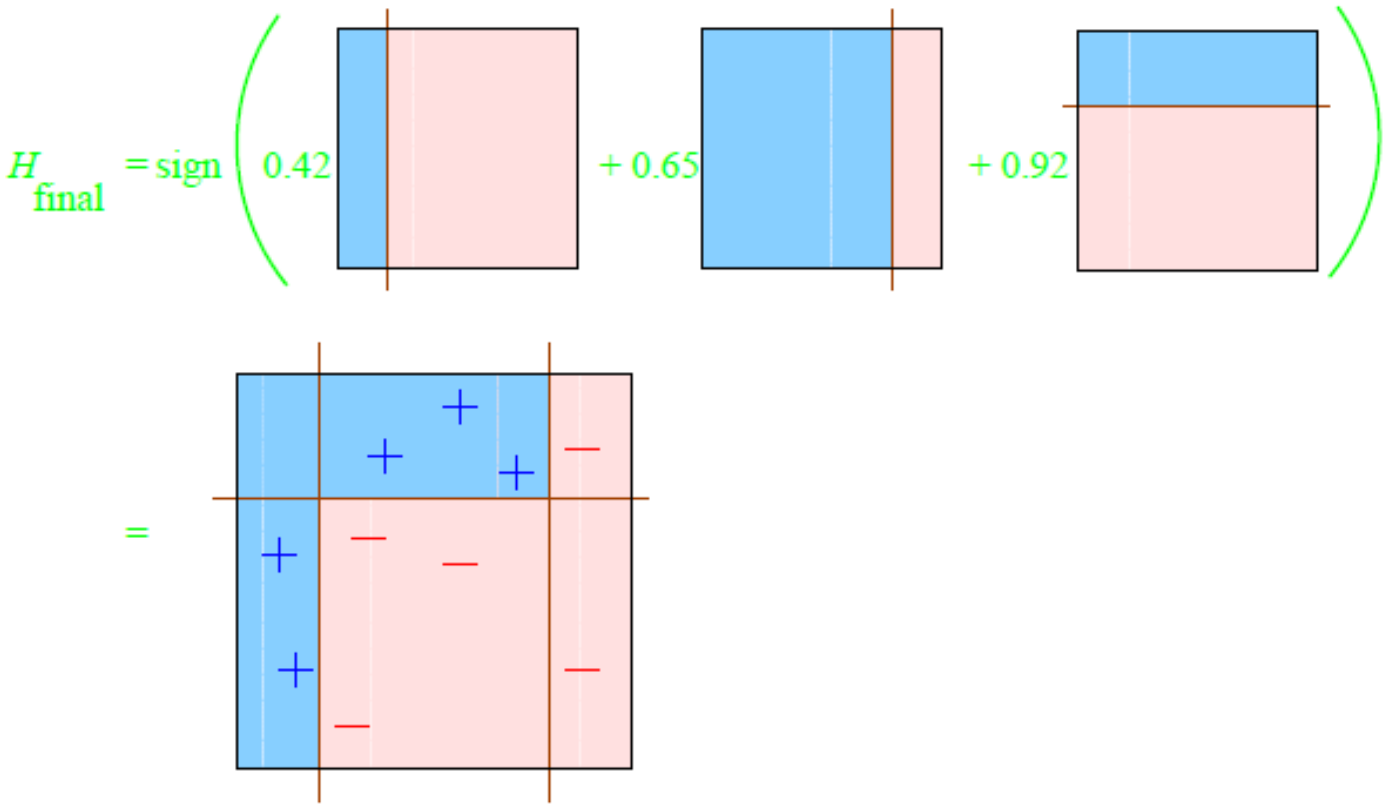
## Round 3




$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

## Final Hypothesis




Source: Singer and Lewis Tutorial

# *Boosting. Summary*

- It is one of the best ML algorithms, but it is sensitive to noise (class overlap). The reason is that Boosting models focuses on instances difficult to classify by the previous model. But noisy instances are always difficult to classify. Hence, Boosting will try to classify instances that cannot be classified, by memorizing them (and therefore, will incur in overfitting).
- It has other advantages similar to Bagging (probabilistic predictions, attribute ranking, out-of-bag estimations, …)

# *GRADIENT BOOSTING*

- Adaptation of Boosting for regression
- Other names: Gradient Boosting, Gradient Boosting Machines, …
- In every iteration, it trains a model to predict the difference between the actual output and the output of the ensemble trained so far (pseudo-residuals)

# GRADIENT BOOSTING

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

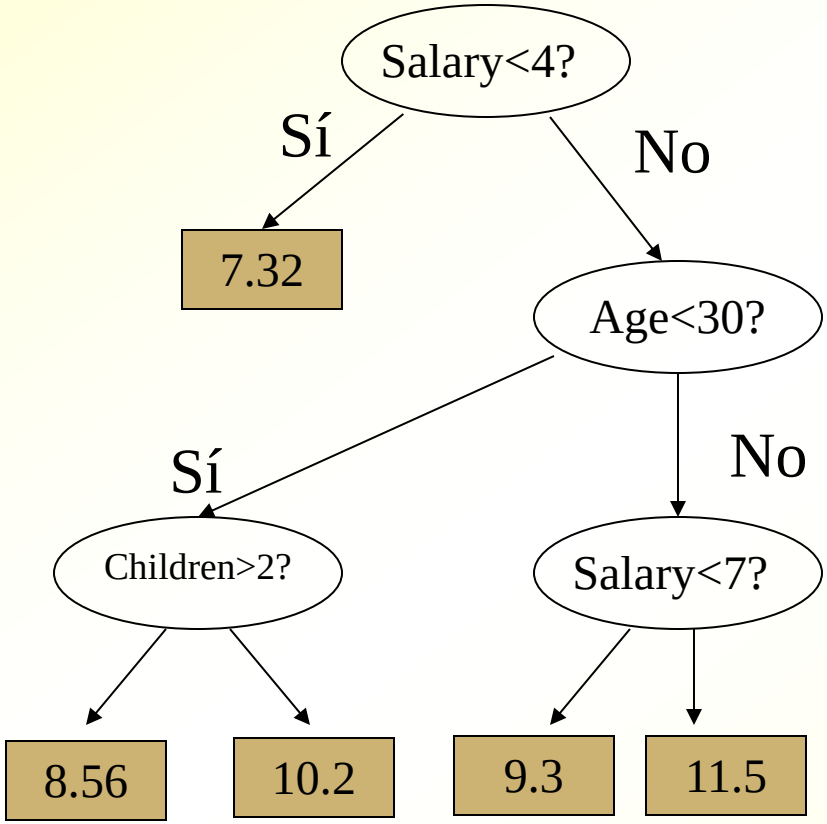
- Example, for minimizing mean squared error

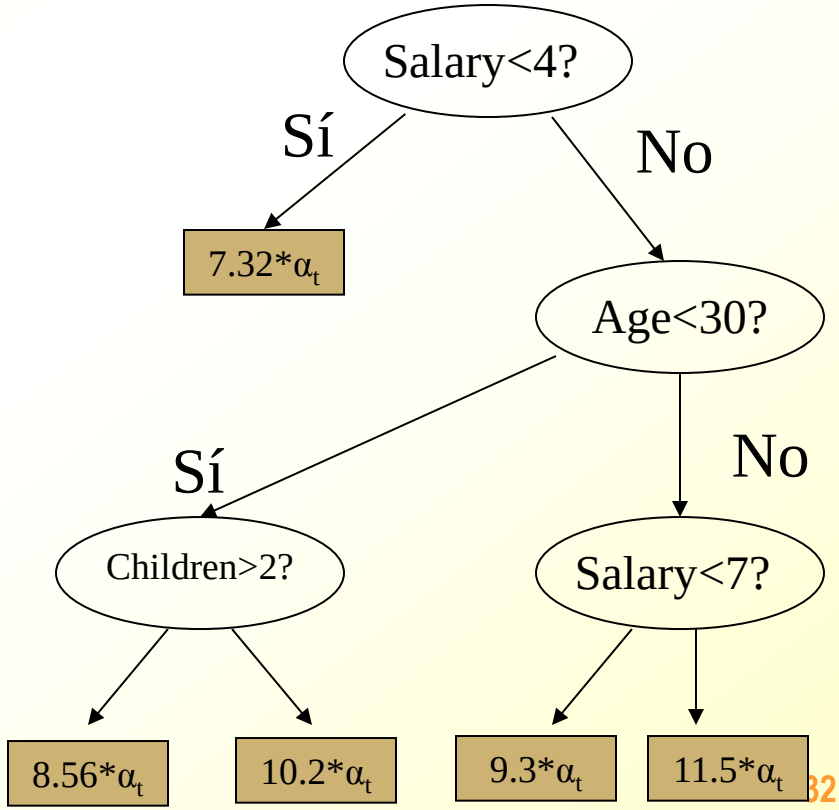
1. Initialize $h_0(x)$ = average of outputs
2. FOR t = 1 TO T
   - Compute a new training set, whose outputs are the "pseudo-residuals", where for each $(x_i, y_i)$ we get an instance $(x_i, y_i - f_{t-1}(x))$
     - Where $f_{t-1}(x) = h_0(x) + \alpha_1 h_1(x) + \ldots + \alpha_{t-1} h_{t-1}(x)$ ; the ensemble so far
   - Train a new model $h_t(x)$ that fits the pseudo-residuals
   - Compute $\alpha_t$ in order to minimize the error of the ensemble:

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

# GRADIENT BOOSTED TREES

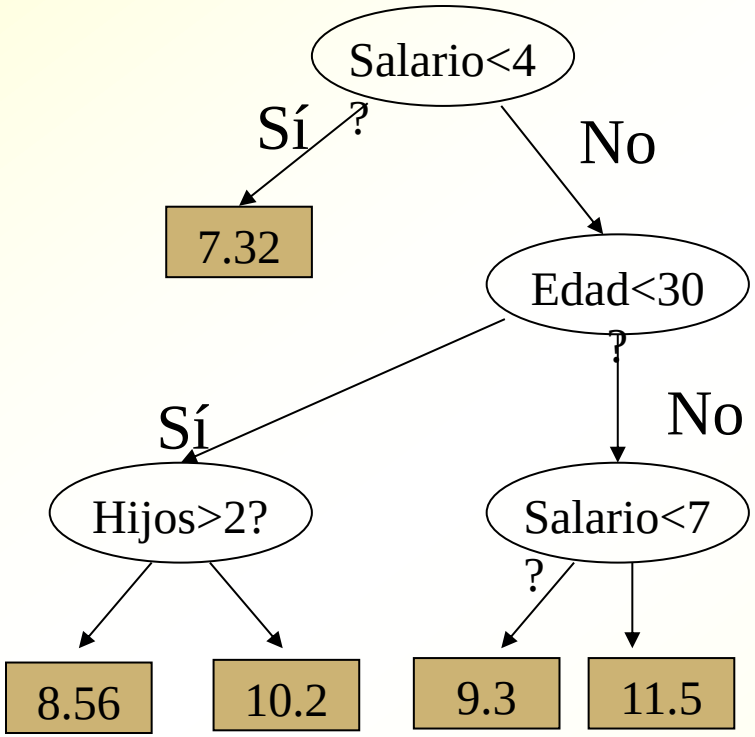
- It is Gradient Boosting with regression trees, but instead of finding the optimal $\alpha_t$ for $f_{t-1}(x)+\alpha_t h_t(x)$ …

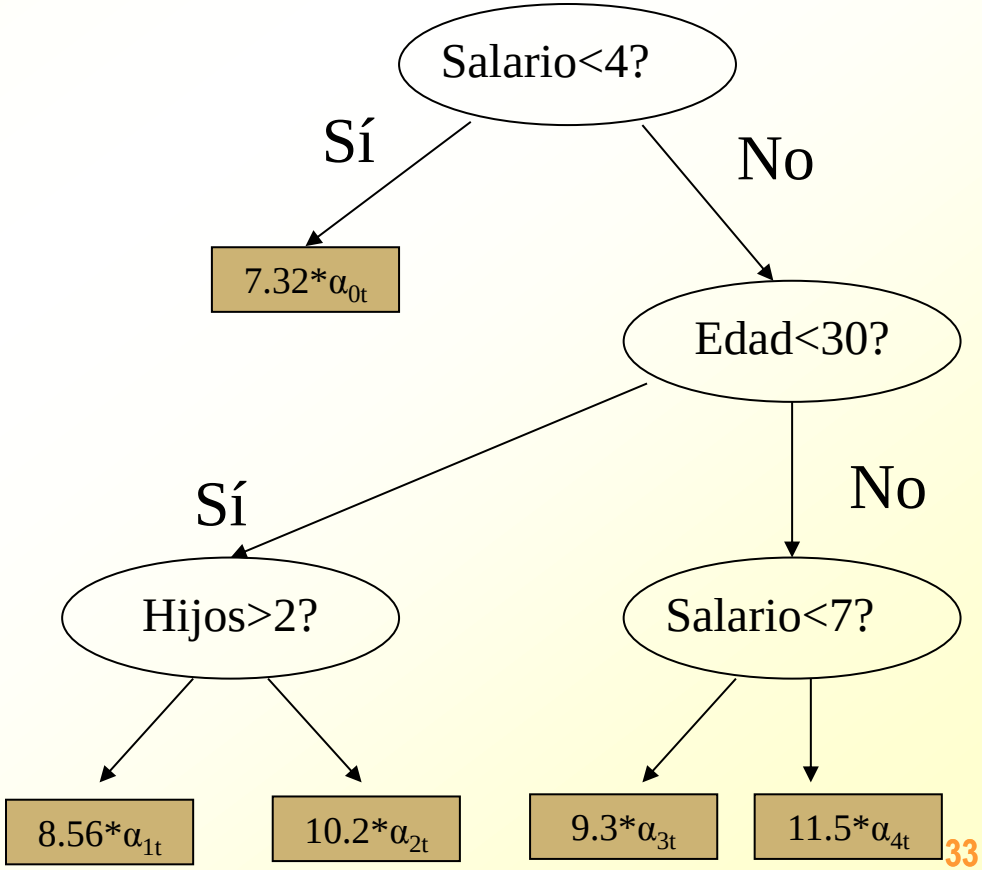


# GRADIENT BOOSTED TREES

- … a different alpha is optimized for every leave: $\alpha_0\ \alpha_1\ \alpha_2\ \alpha_3\ \alpha_4$



# *GBT and overfitting*

- All trees in the ensemble contain the same number of leaves
- In order to avoid overfitting:
  - Control the number of trees T in the ensemble (small T => small overfitting)
  - Shrinkage: $f_t(x) = f_{t-1}(x)+v*\alpha_t h_t(x)$
    - Use a learning rate $0<v<1$
    - This decreases the weight of the model ht, and therefore it is more difficult for the model to overfit the data
    - But small v makes training slower
  - Minimum number of instances in the leaves

# *Stochastic Gradient Boosting*

- In every iteration, a random sample **without replacement** is used, instead of the whole dataset.
- Example: with a subsample of 0.8, only 80% of the original dataset is used (and instances are not repeated)
- It avoids overfitting, because training data is different for every model in the sample
- It provides out-of-bag estimation