

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)

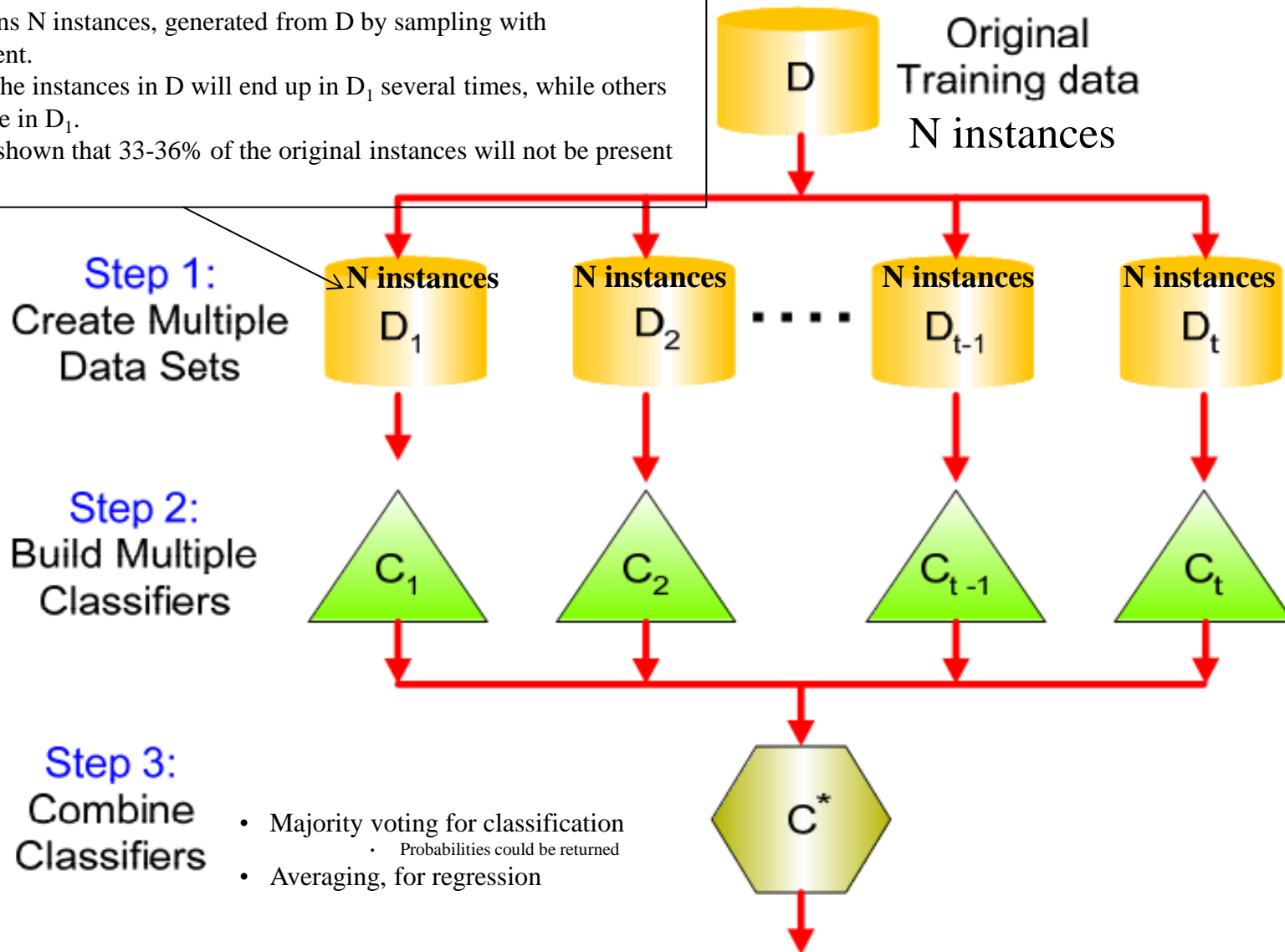
BAGGING

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

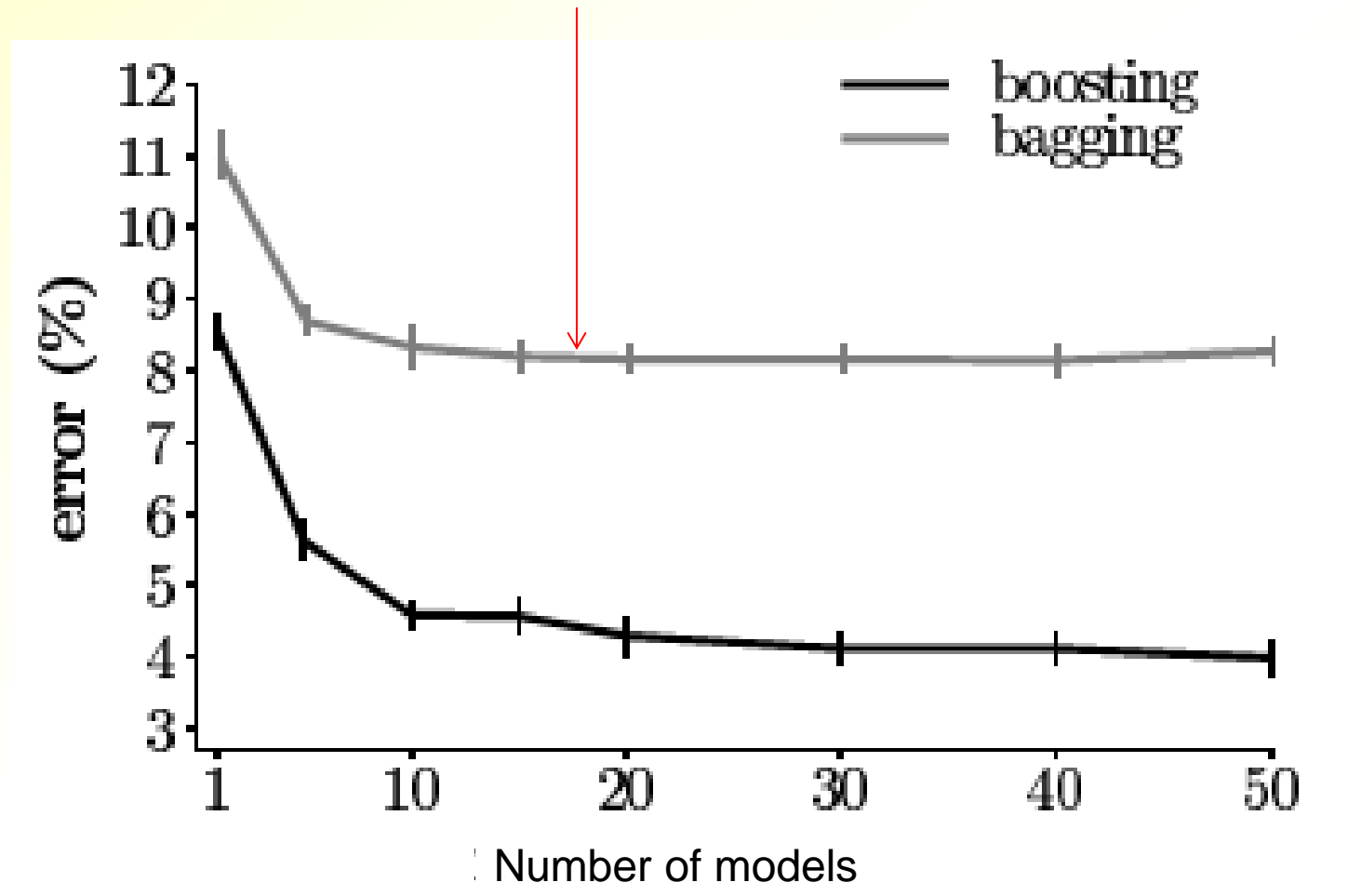


Randomization

- 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: $\varepsilon=0.35$ (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):

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

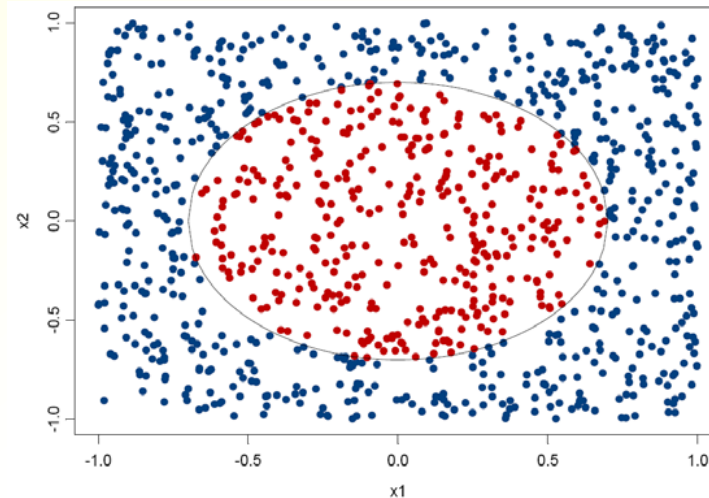
$\varepsilon^i * (1-\varepsilon)^{25-i}$ represents the case where i classifiers fail and $25-i$ succeed (two-class problem)

- 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

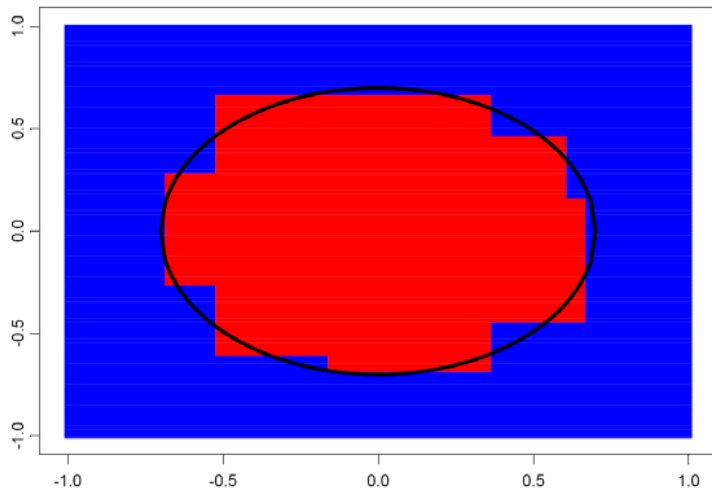
Why does it work?

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

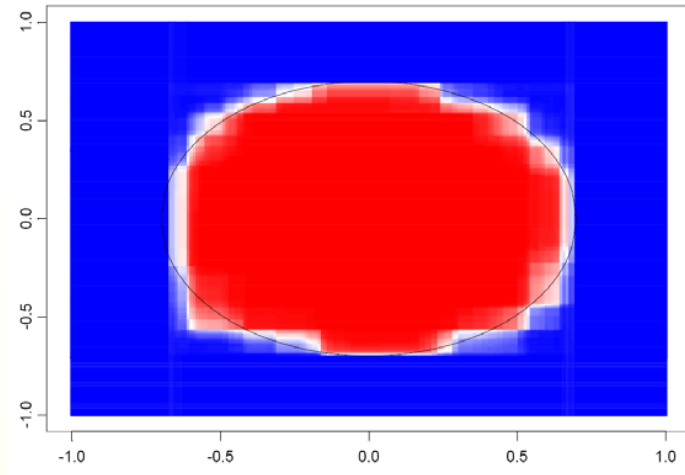
Original data



Decision tree

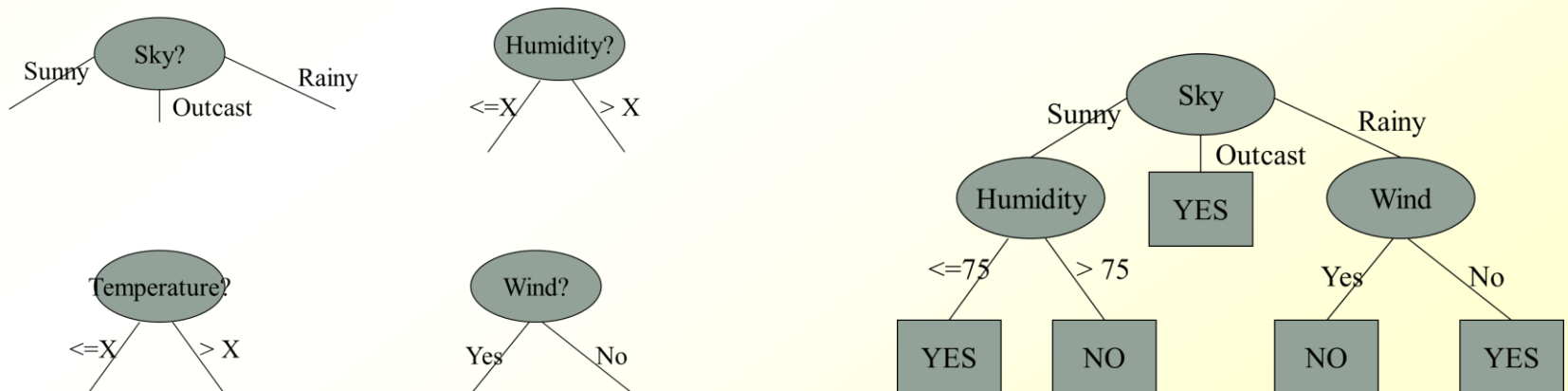


Ensemble of 100 decision trees



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

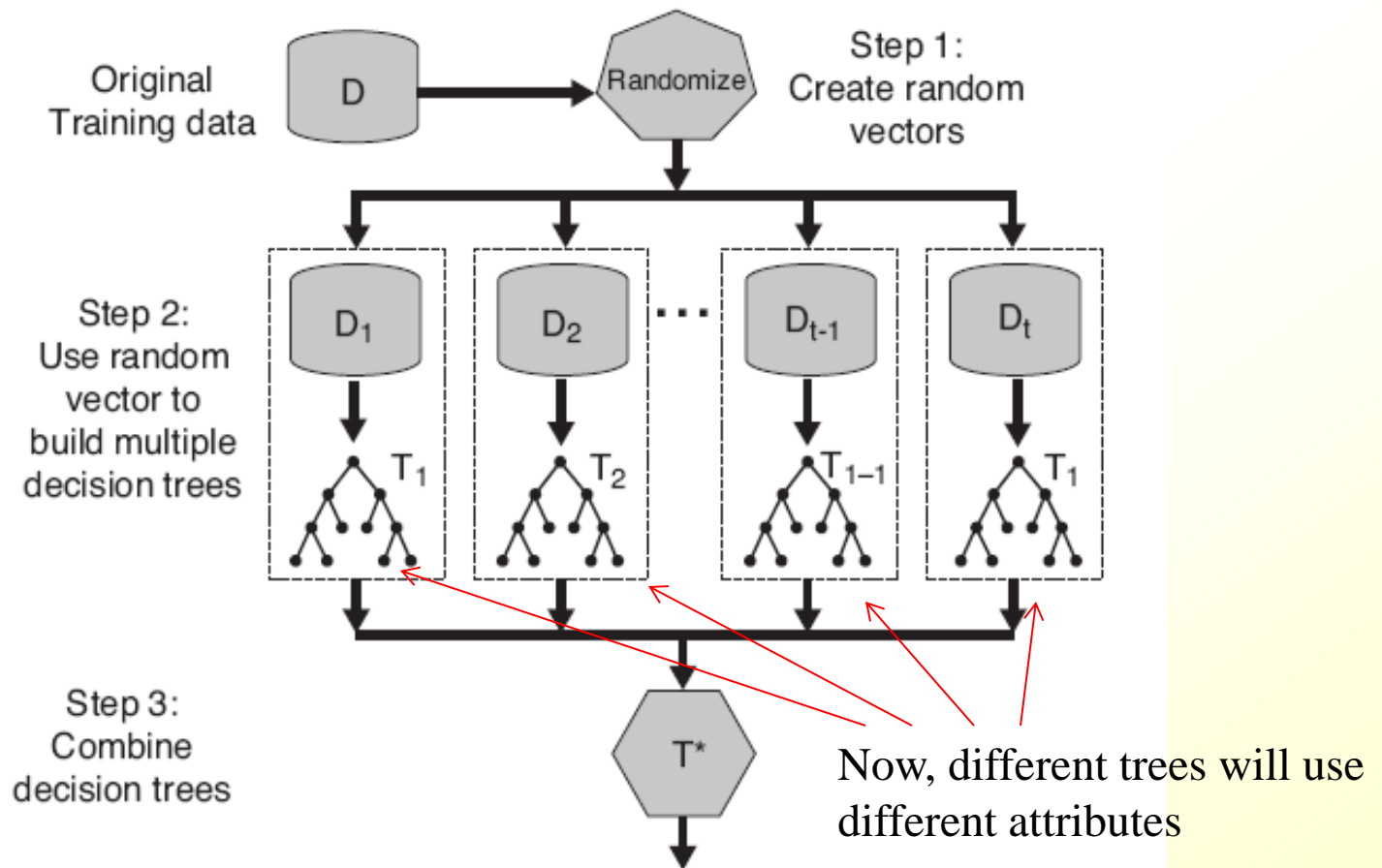


Figure 5.40. Random forests.

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

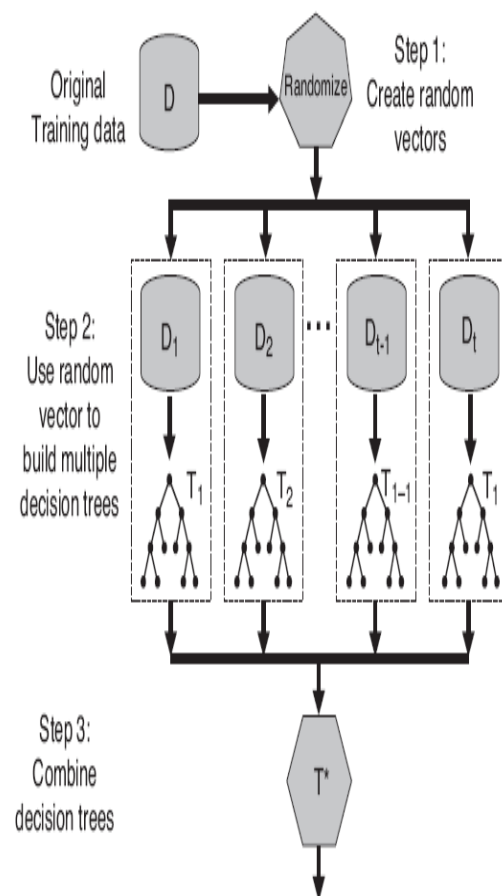


Figure 5.40. Random forests.

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 $m \ll M$ 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):
 $\{(x_1, y_1), \dots, (x_a, y_a), \dots, (x_N, y_N)\}$
- In boosting, every instance a has a weight w_a . Initially all weights are the same for all instances $w_a = 1/N$
 - $\{(x_1, y_1, w_1 = 1/N), \dots, (x_a, y_a, w_a = 1/N), \dots, (x_N, y_N, w_N = 1/N)\}$
- 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

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

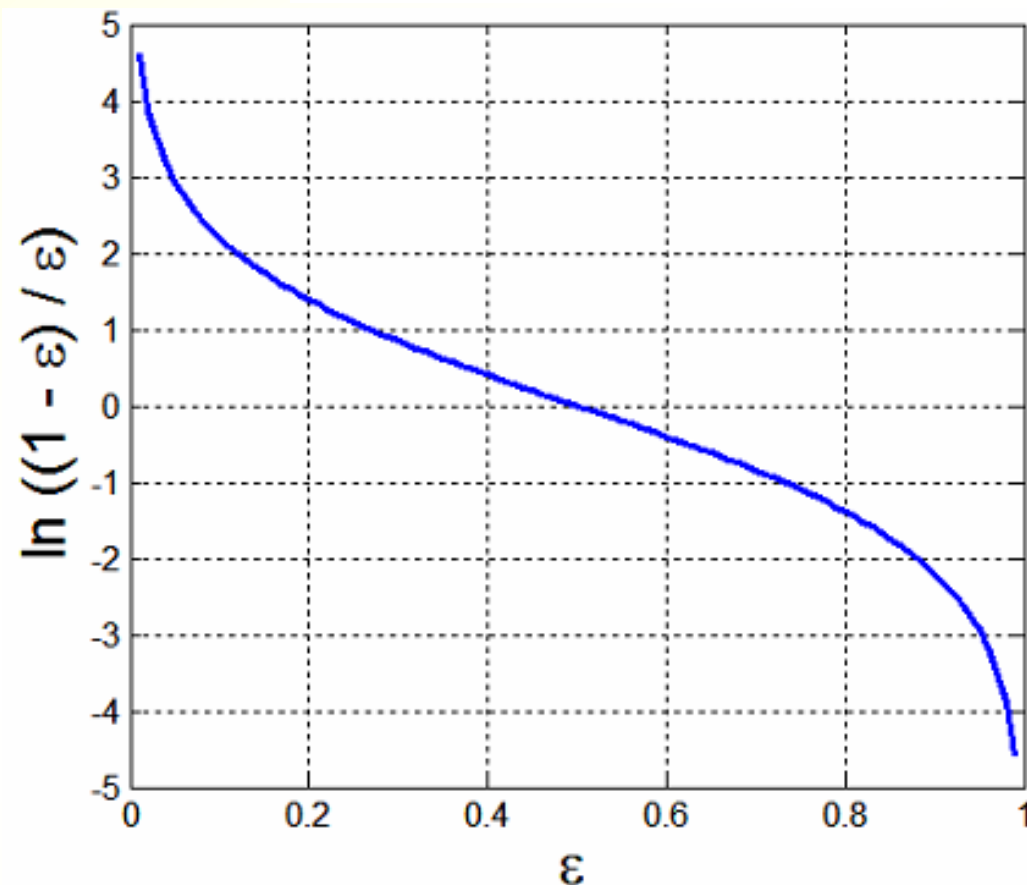
$$f(\mathbf{x}) = \sum_{t=1}^T \alpha_t h_t(\mathbf{x}).$$

Note: if error ε_i small then α_i large, if error ε_i large then α_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

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



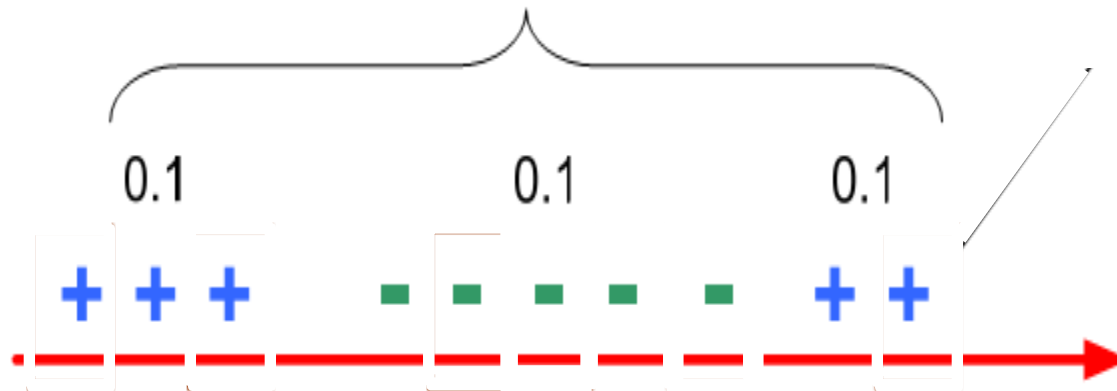
Creating the new training set

- 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 randomly sampling the training set. The probability that an instance is selected is proportional to its weight:

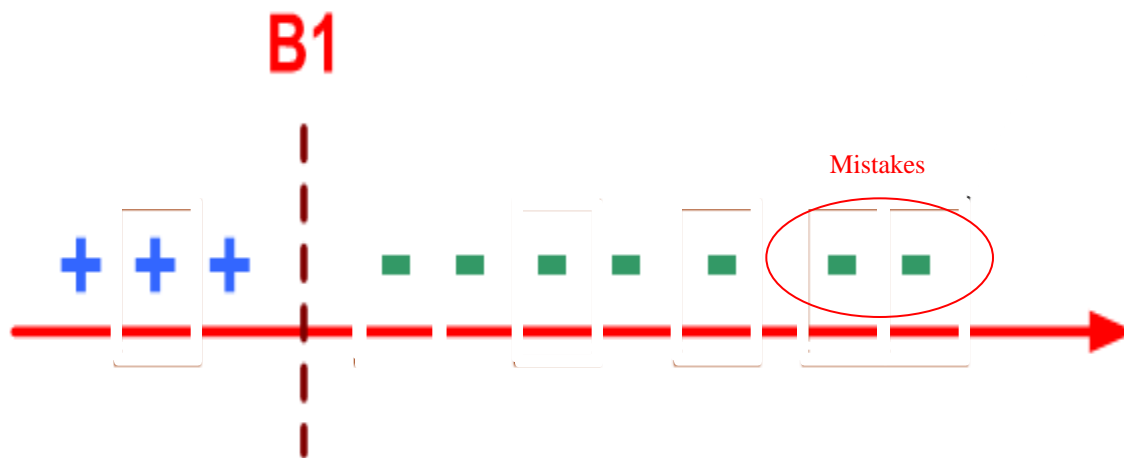
$$\text{prob}_a = w_a / (w_1 + w_2 + \dots + w_N)$$

Initial weights for each data point

Original
Data



Boosting
Round 1

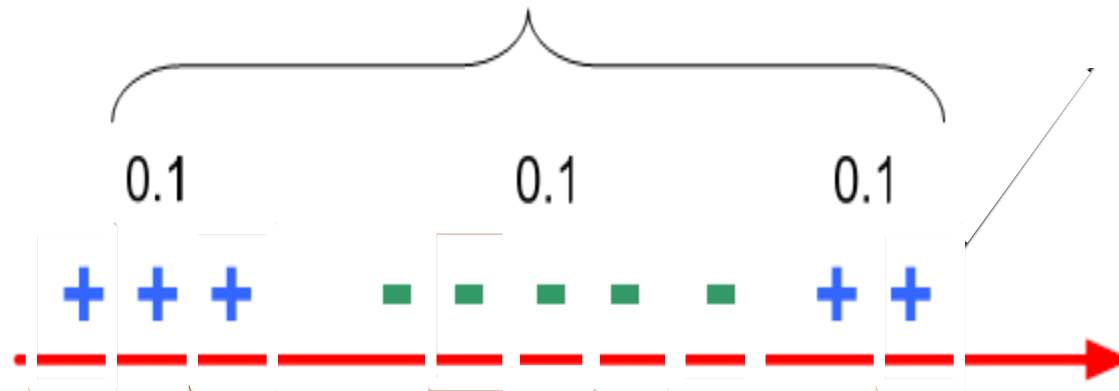


$$\alpha = 1.9459$$

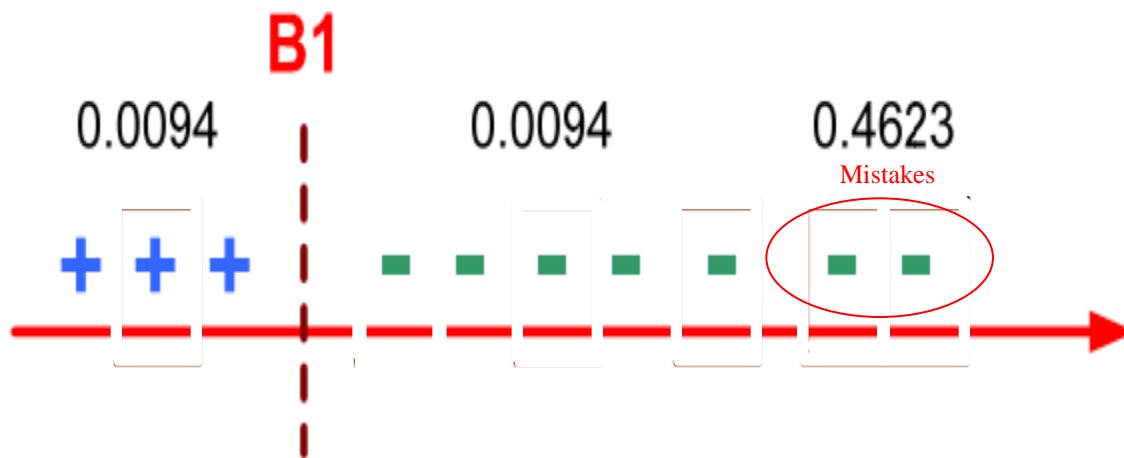
$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$

Initial weights for each data point

Original
Data



Boosting
Round 1



$$\alpha = 1.9459$$

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

Original
Data

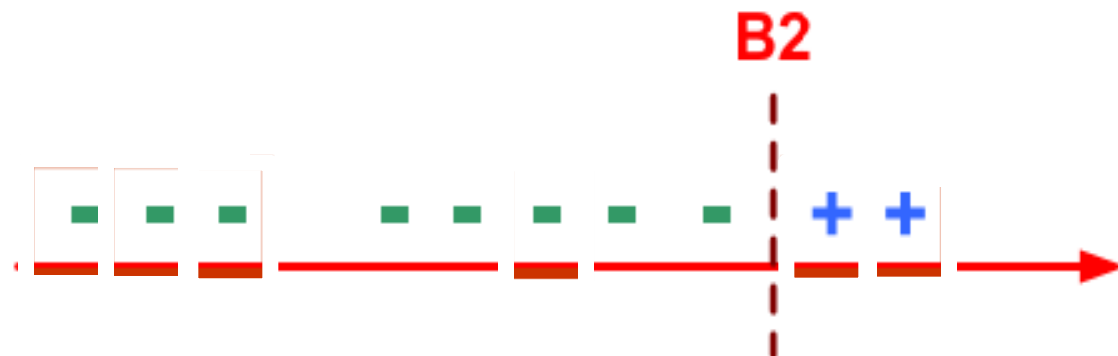


Boosting
Round 1



$$\alpha = 1.9459$$

Boosting
Round 2



$$\alpha = 2.9323$$

Boosting
Round 3



$$\alpha = 3.8744$$

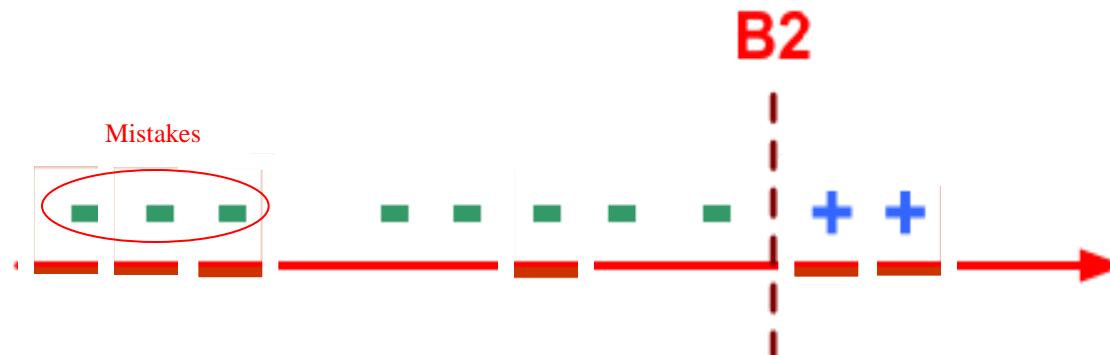
Original
Data



Boosting
Round 1



Boosting
Round 2



Boosting
Round 3



Original
Data

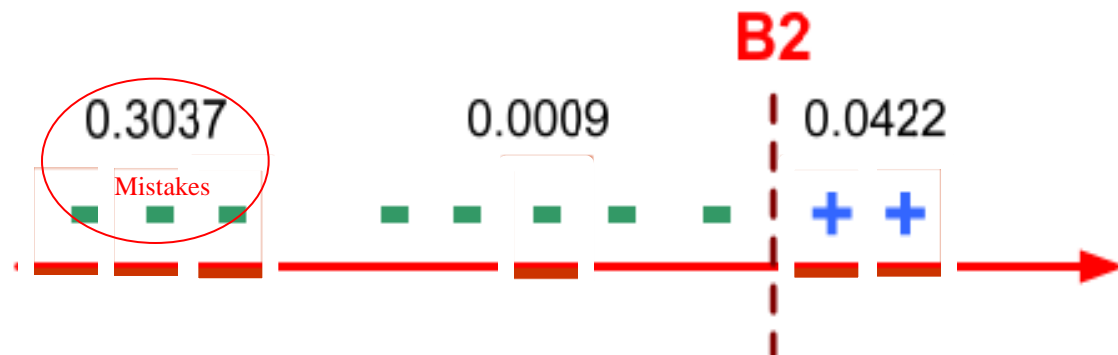


Boosting
Round 1



$$\alpha = 1.9459$$

Boosting
Round 2



$$\alpha = 2.9323$$

Boosting
Round 3



$$\alpha = 3.8744$$

Original
Data

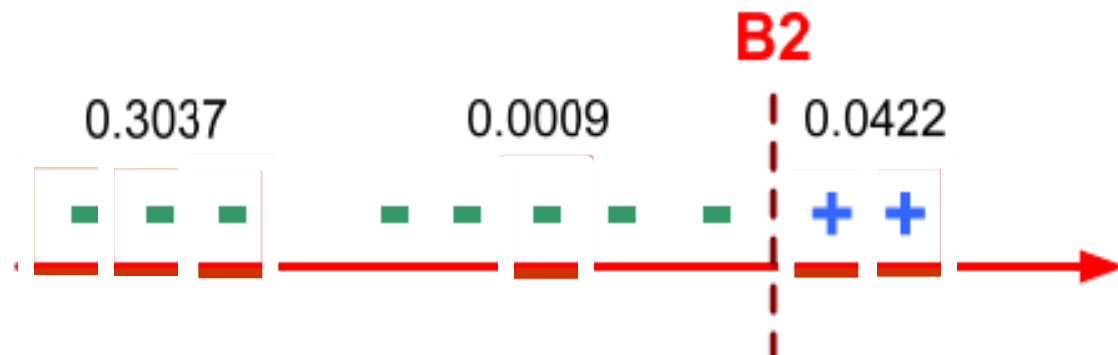


Boosting
Round 1



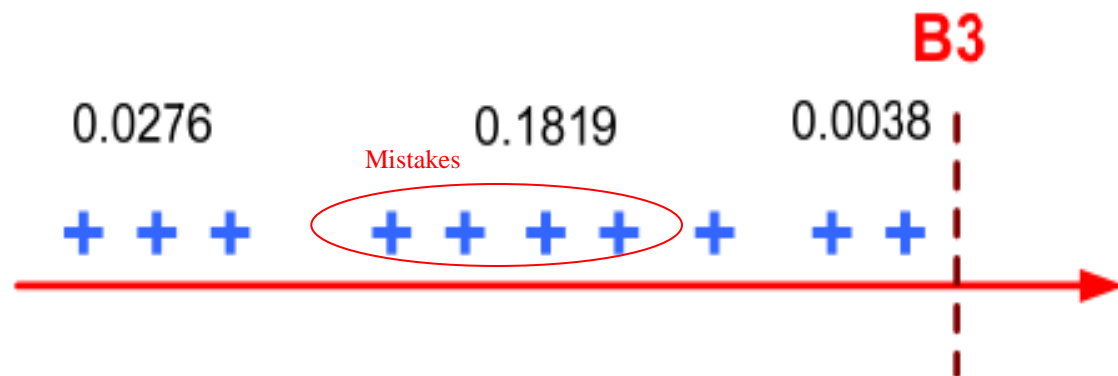
$$\alpha = 1.9459$$

Boosting
Round 2



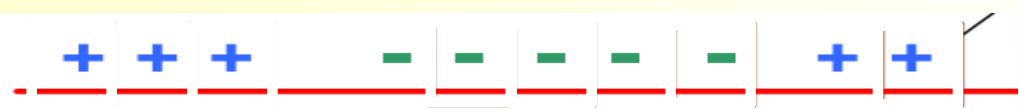
$$\alpha = 2.9323$$

Boosting
Round 3



$$\alpha = 3.8744$$

Original
Data



B1

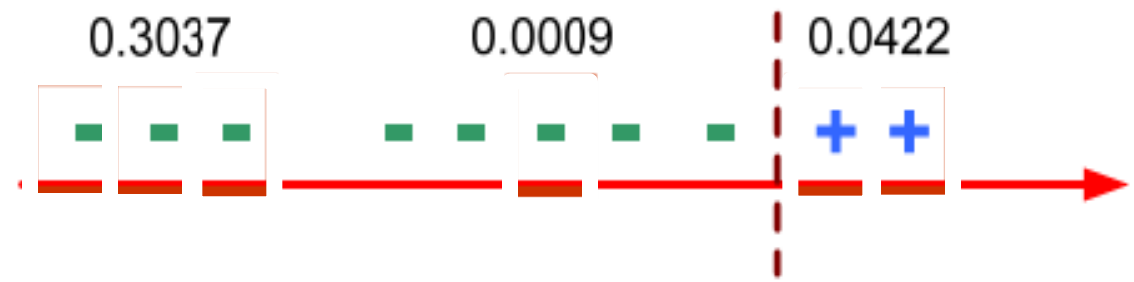
Boosting
Round 1



$$\alpha = 1.9459$$

B2

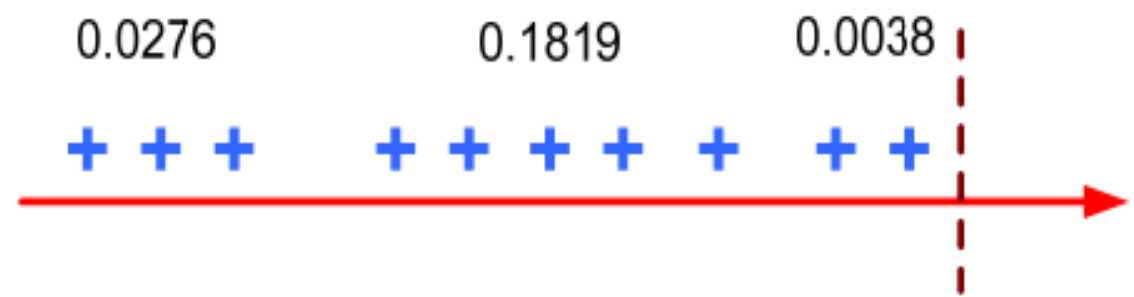
Boosting
Round 2



$$\alpha = 2.9323$$

B3

Boosting
Round 3



$$\alpha = 3.8744$$



Boosting Round 1	0.0094 + + +	0.0094 - - - - -	0.4623 - -	$\alpha = 1.9459$
Boosting Round 2	0.3037 - - -	0.0009 - - - - -	0.0422 + +	$\alpha = 2.9323$
Boosting Round 3	0.0276 + + +	0.1819 + + + + +	0.0038 + +	$\alpha = 3.8744$
Overall	+ + +	- - - - -	+ +	
	2.888 = +	-1.0038 = -	4.8608 = +	

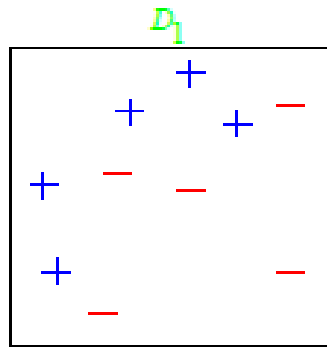
$$1.9459*B1(x) + 2.9323*B2(x) + 3.8744*B3(x) = f(x)$$

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

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

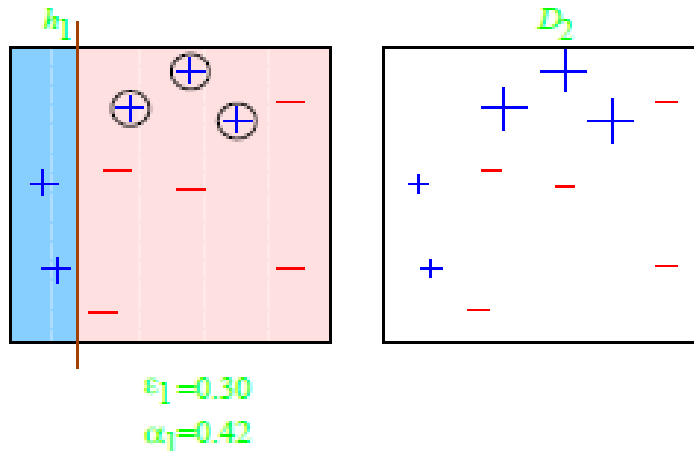
$$1.9459*(-1) + 2.9323*(+1) + 3.8744*(+1) = 4.8608 > 0$$

Example of boosting in 2D instance space

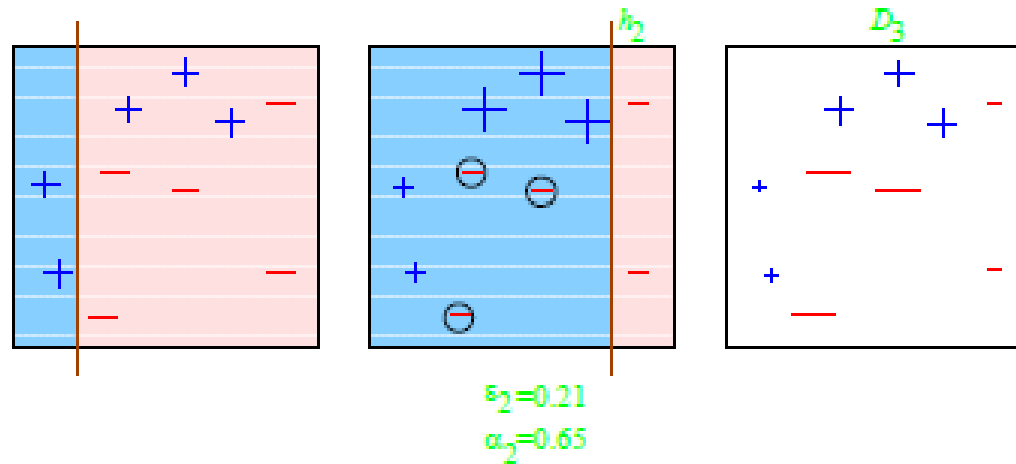


Weak hypotheses == vertical or horizontal half-planes

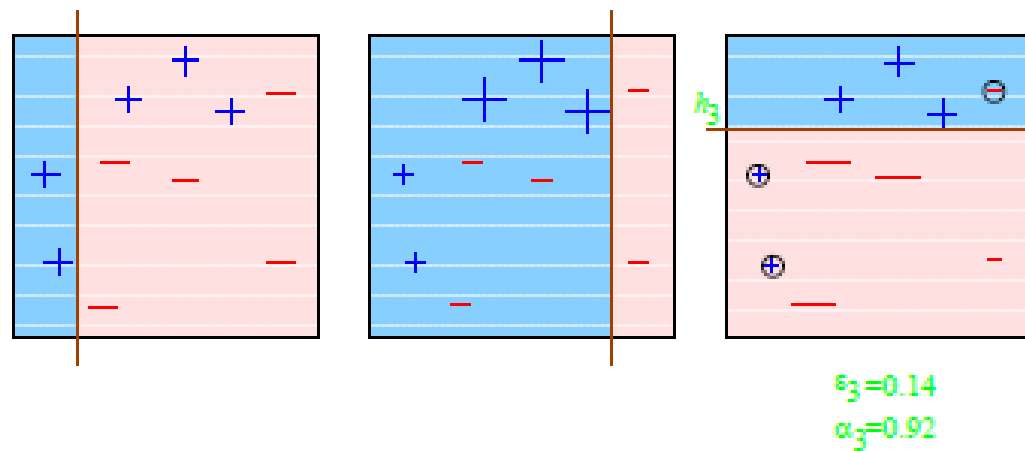
Round 1



Round 2



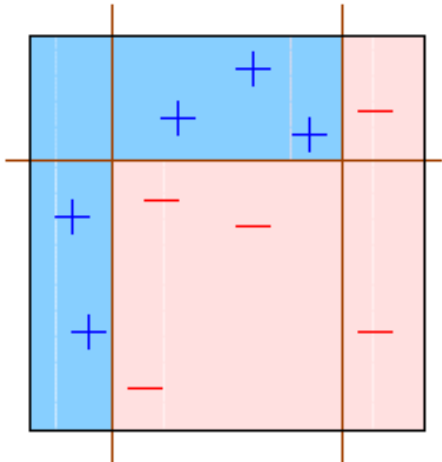
Round 3



Final Hypothesis

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

=



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

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

■ Example, for minimizing mean squared error

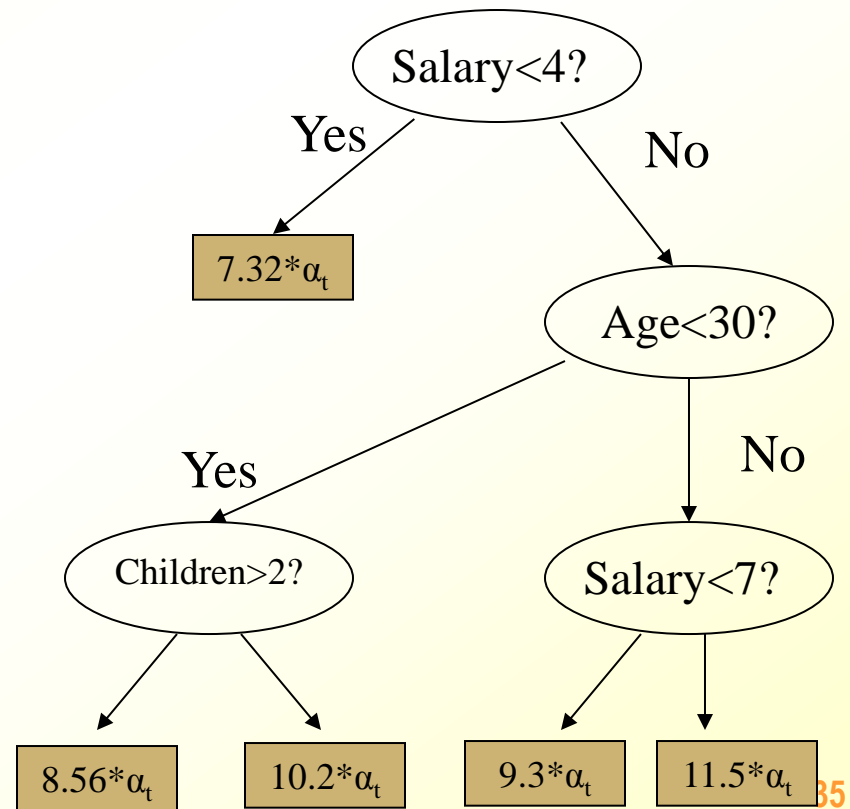
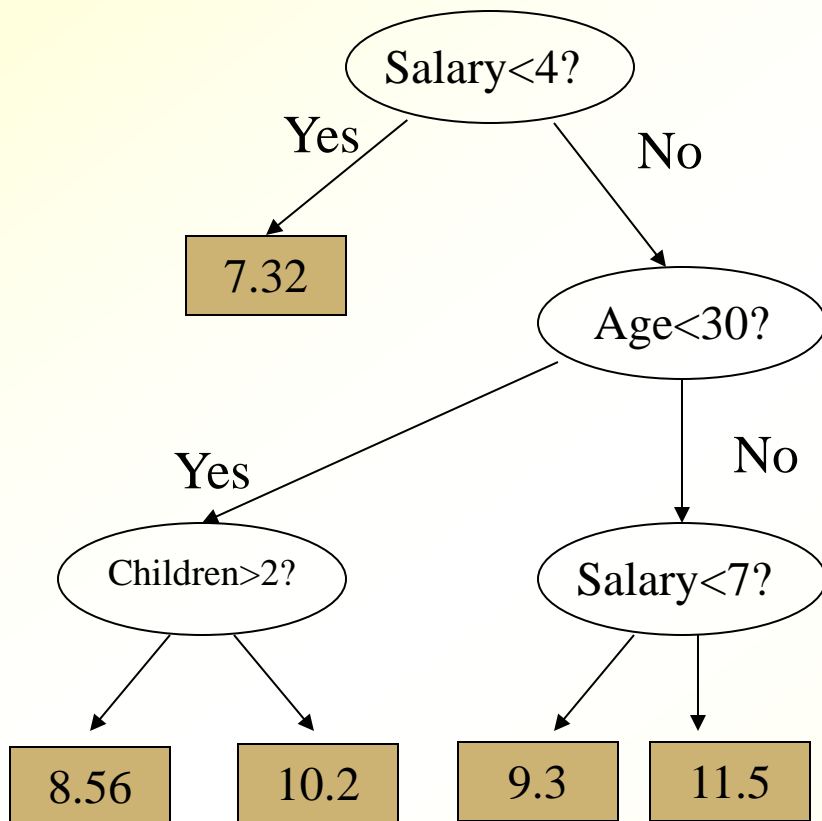
1. Initialize $h_0(\mathbf{x})$ = average of outputs

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 $f_{t-1}(x) = h_0(x) + \alpha_1 h_1(x) + \dots + \alpha_{t-1} h_{t-1}(x)$; the ensemble so far
- Train a new model $h_t(x)$ that fits the pseudo-residuals
- Compute α_t in order to minimize the error of the ensemble:
 $f_{t-1}(x) + \alpha_t h_t(x)$
- Update the ensemble: $f_t(x) = f_{t-1}(x) + \alpha_t h_t(x)$

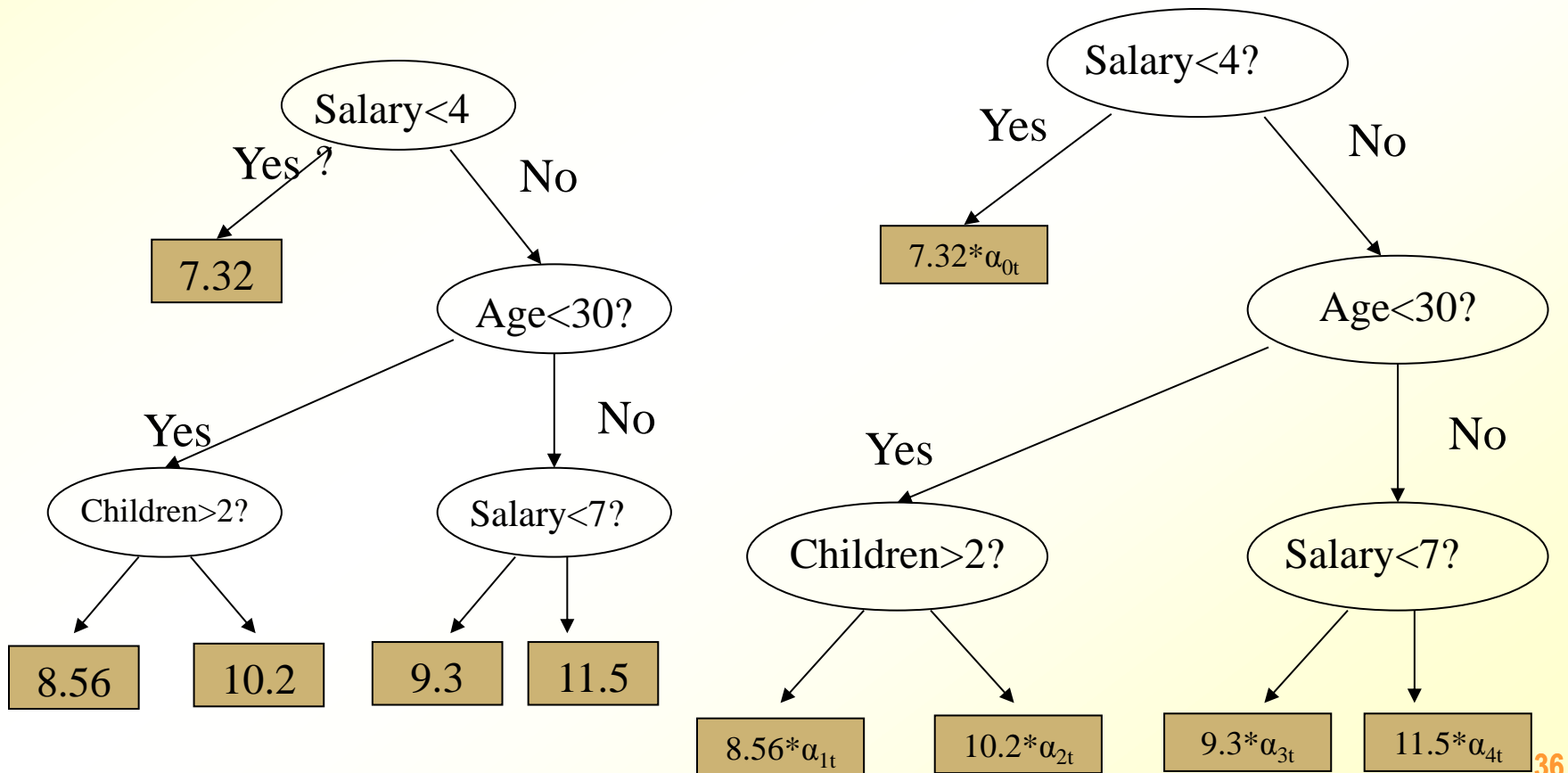
GRADIENT BOOSTED TREES

- It is Gradient Boosting with regression trees, but instead of finding the optimal α_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

■ In order to control overfitting:

- Hyper-parameter T (number of trees in the ensemble) (T small => less complex model / ensemble)
- Shrinkage (or learning rate): $f_t(x) = f_{t-1}(x) + v * \alpha_t h_t(x)$
 - Learning rate $0 < v < 1$
 - Each new tree has a smaller weight on the ensemble, and therefore less overfitting to data.
 - But v small usually implies using more trees on the ensemble (i.e. training takes longer).
- Max tree depth (or minimum number of instances in the nodes)

Stochastic Gradient Boosting

- Every iteration uses a smaller random sample (without replacement) of the training dataset
 - E.g. use 80% of the original training dataset
- Avoids overfitting, to some extent, because the sample is slightly different at each iteration.
- It also allows out-of-bag estimation

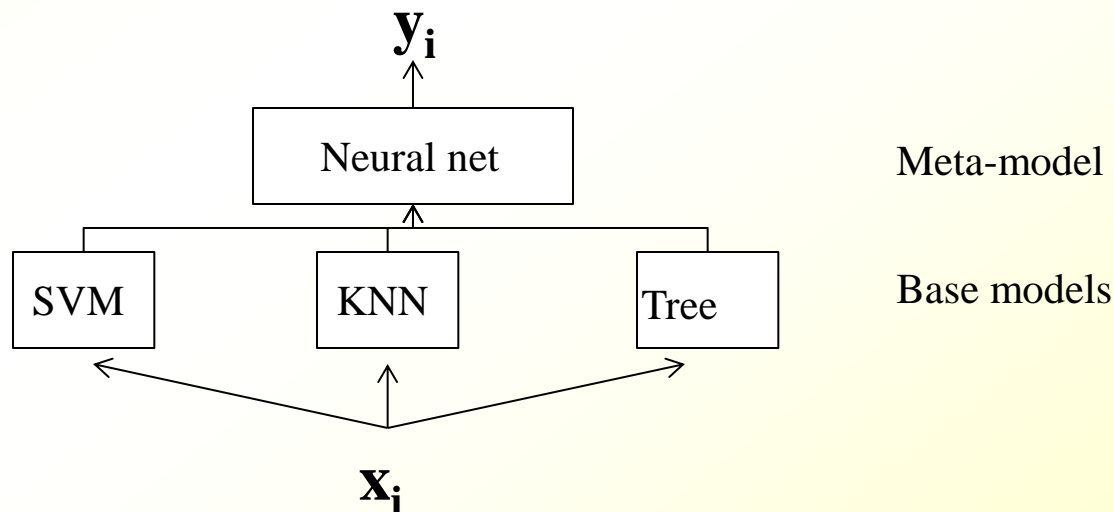
XGBOOST (eXtreme Gradient Boosting)

- On a practical level, XGBOOST is one of the best implementations of Gradient Tree Boosting: accurate, fast, scalable (multi-core) and distributed (clusters, Hadoop, Spark, ...)

<https://github.com/dmlc/xgboost>

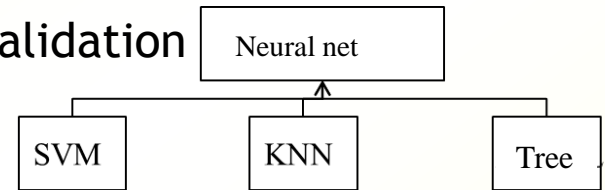
Stacking (Stacked Generalization)

- So far, the final result of an ensemble is carried out by means of a fixed rule: majority voting or (weighted) averaging.
- In Stacking, the rule is also learned from data
- The classifier that implements the combination rule is called the **meta-model**.
- The base models can be heterogeneous (unlike in Bagging and Boosting)
- The meta-model learns when to trust each base model, taking into account the outputs of the other base models.



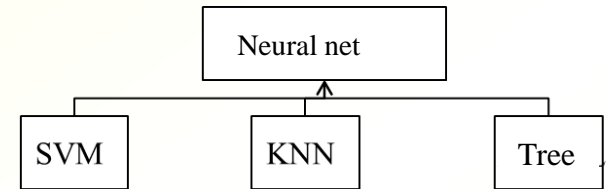
Stacking (Stacked Generalization)

- Base models and meta-model are trained with crossvalidation



- For example, for base model SVM:
 - 10-fold crossvalidation. For each fold:
 - SVM is trained with 9 folds and predictions are made for the remaining fold
- This is done for the other base models too.
- A new training dataset is constructed, where the inputs are the outputs of the base models, and the outputs are the original outputs.
- The aim of this new training set is to obtain “realistic” outputs of the base classifiers (i.e. outputs generated with data not used during training).
- The meta-model is trained with the new training set.
- Finally, the base-models are re-trained with the complete training set.

Stacking



- For classification problems, it is recommended that base-models output probabilities
 - E.g. for a two class problem (“+” and “-”)
 - and instance with input attributes x
 - output $(p(+|x), p(-|x))$
- Many ML algorithms output probabilities (or something close): decision trees, calibrated SVMs, neural networks, naive bayes, ensembles, ...
- Then, train a Multi-response linear regression meta-model (Ting & Witten 1999)
 - For a classification problem with M classes, train M linear regression models (LR_j).
 - The inputs of each LR_j are the probabilities of the base models. The output is the class (1 if instance belongs to class j , 0 otherwise)
 - $LR_+(x) = a_{svm} p_{svm}(+|x) + a_{knn} p_{knn}(+|x) + a_{tree} p_{tree}(+|x) + a_0$
 - $LR_-(x) = b_{svm} p_{svm}(-|x) + b_{knn} p_{knn}(-|x) + b_{tree} p_{tree}(-|x) + b_0$
 - New instances x are classified as $\max_j LR_j(x)$

Mixtures of experts

- Now, the meta-model has access to the input attributes of the instance to be classified.
- Typically, the meta-model is a linear combination of the base models. A gating network computes the weights of the base models.
- Mixtures of experts allow different base-models to become specialized in different regions of instance space.

