ENSEMBLES OF MODELS

BOOSTING

BOOSTING

Boosting

- Motivation: to iteratively improve (to boost) weak models (weak learners)
  - Weak learner: method that trains models better than random, but not much better
  - Trees with small depth are weak learners (e.g. decision stumps are trees with max depth = 1)
- Boosting adds **sequentially** a new model h_t to the ensemble

  $\{h_0, h_1\} + h_3 \Rightarrow \{h_0, h_1, h_3\} \ldots$
- The general idea is that the new base model h_t improves the ensemble trained so far
- Adaboosting was one of the first versions of boosting

Adaboosting (adaptive boosting)

- We know that the training data is a list of instances (tuples):

  $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_a,y_a), \ldots, (\mathbf{x}_N,y_N)\}$

- In boosting, every instance *a* has a weight w_a. Initially all weights are the same for all instances $w_a=1/N$

  $\{(\mathbf{x}_1,y_1,w_1=1/N), \ldots, (\mathbf{x}_a,y_a,w_a=1/N), \ldots, (\mathbf{x}_N,y_N,w_N=1/N)\}$

- At every iteration, weights change, in order to give more importance to more difficult instances (and contrarywise for easy instances)

w (weight)	S	T	H	V	Play
2	Sun	85	85	N	No
1.5	Sun	80	90	Y	No
1	Sun	72	95	N	No
1	Sun	69	70	N	Yes
0.5	Sun	75	70	Y	Yes

Adaboost (for classification)

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 ε_0
1. Repeat for t=1 to t=T-1 (T-1 times) (and while $\varepsilon_{t-1} < 0.5$)
    1. Create a new training set by giving larger weights to difficult instances:
        1. If h_{t-1} missclassifies (x_a,y_a), then increase $w_a = w_a * (1- \varepsilon_{t-1}) / \varepsilon_{t-1}$
        2. If h_{t-1} succeeds with (x_a,y_a), decrease $w_a = w_a * \varepsilon_{t-1} / (1- \varepsilon_{t-1})$
    2. Train a new classifier h_t with the new training set. **We expect this model to focus on instances with large weights.** Its error is ε_t ((computed on the weighted training set)

Result: ensemble f = $\{h_0, h_1, \ldots, h_{T-1}\}$

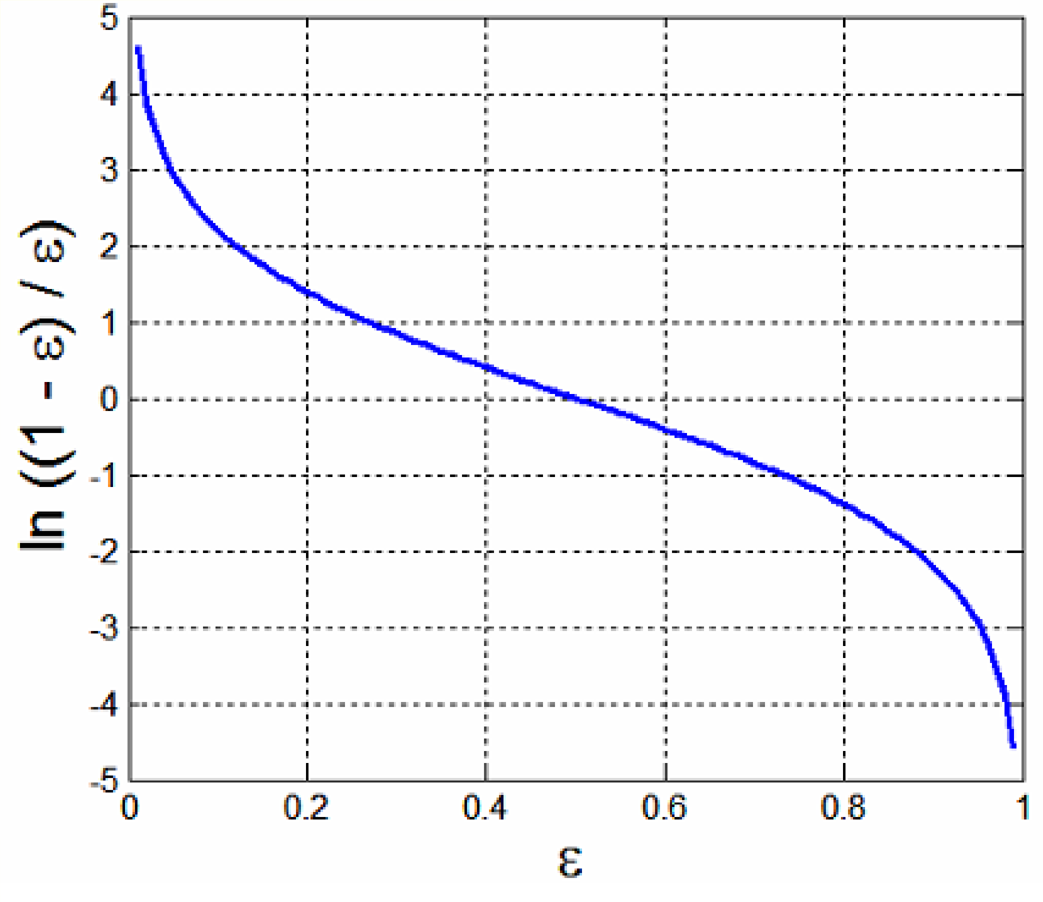
The final classifier *f* is a linear combination of all h_t. The alpha coefficients depend on the accuracy of h_t. if error ε_t small then $\boldsymbol{\alpha}_i$ large, if error ε_t large then $\boldsymbol{\alpha}_t$ is close to 0.

$$f(\boldsymbol{x}) = \sum_{t=0}^{T-1} \alpha_t h_t(\boldsymbol{x}) \qquad \alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

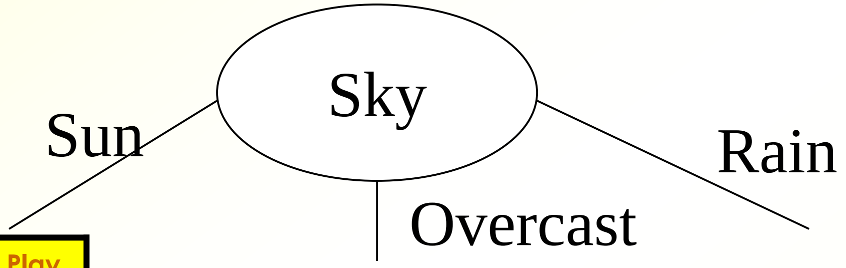
Note: h_i must return either *+1 or -1* (positive / negative class in two-class problems), or an intermediate value.

Computing alpha coefficients

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$



How can models use weights w_a?



w (weight)	S	T	H	V	Play
2	Sun	85	85	N	No
1.5	Sun	80	90	Y	No
1	Sun	72	95	N	No
1	Sun	69	70	N	Yes
0.5	Sun	75	70	Y	Yes

Not considering weights:

- 3 no; 2 yes

Considering weights:

- no: 2+1.5+1= 4.5 no
- yes: 1+0.5= 1.5 yes

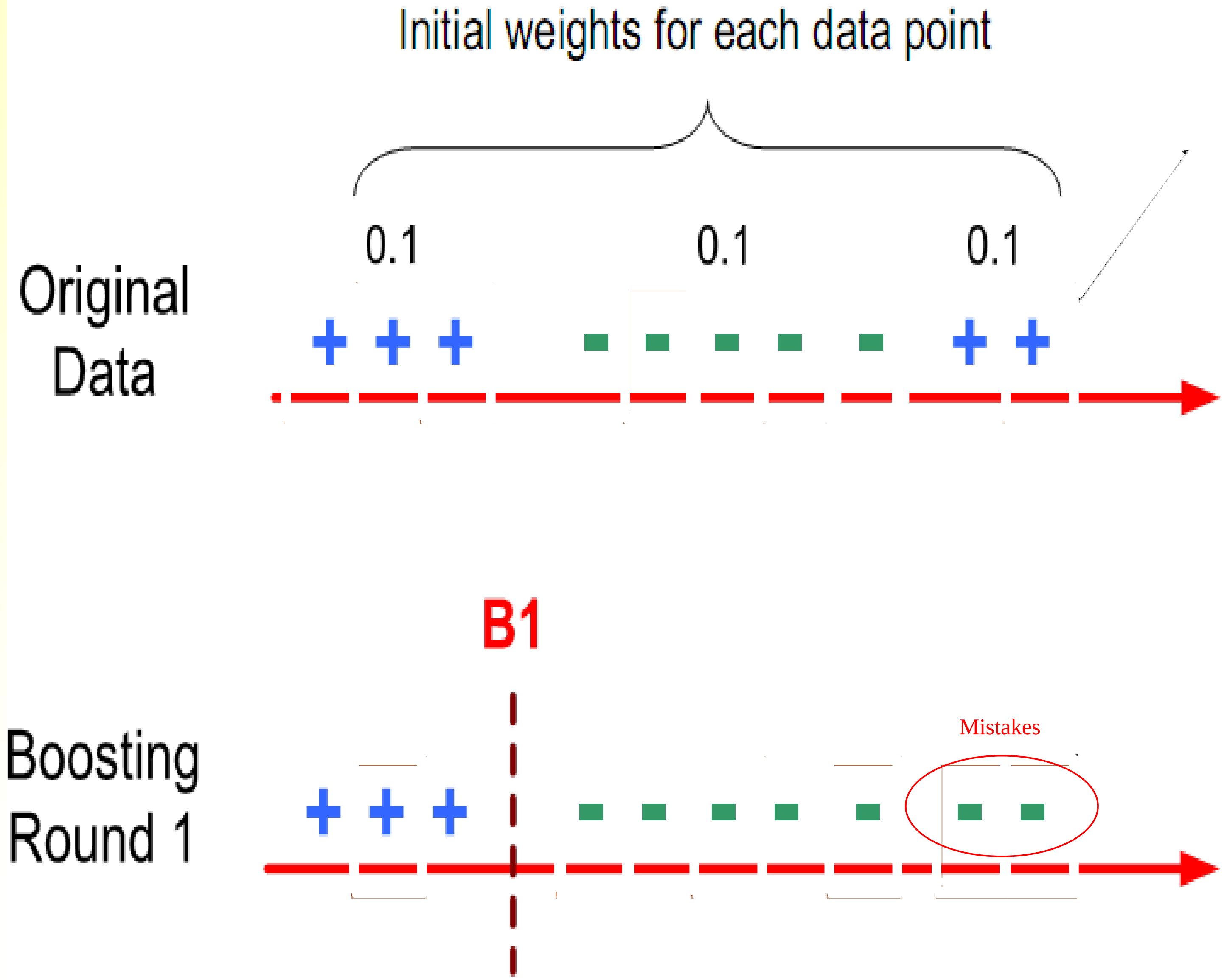
How can models use weights w_a?

- 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 **with replacement**. The probability that an instance is selected is proportional to its weight:

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

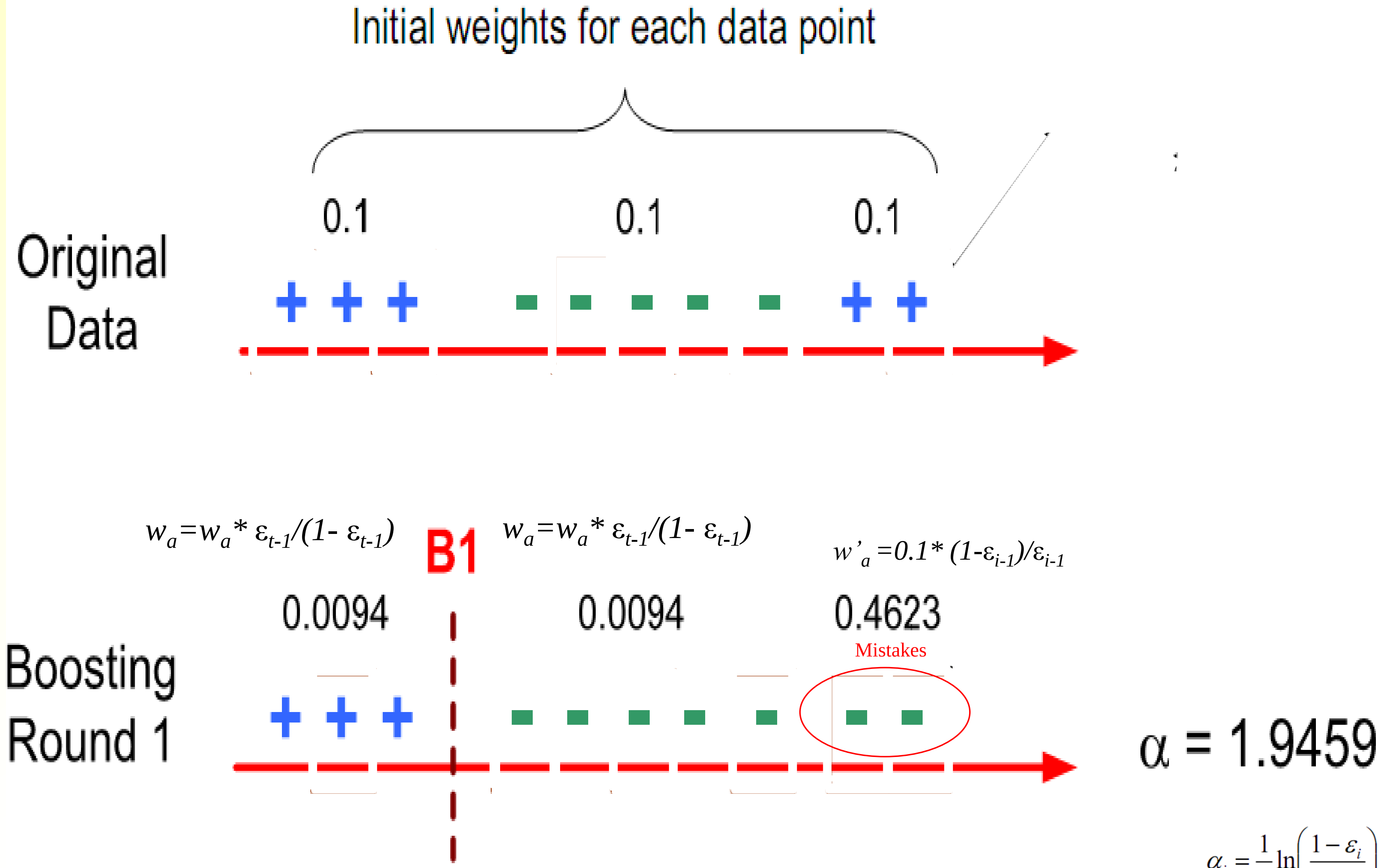
- Thus, instances with higher weights, will be sampled twice, thrice, …, and others (with small weights) will not be sampled, hence they will have no influence on the model.

$$f(x) = \sum_{i=0}^{T-1} \alpha_i f_i(x)$$



α = 1.9459

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$



Initial weights for each data point

0.1 0.1 0.1

Original Data

Boosting Round 1

$w_a = w_a * \varepsilon_{t-1}/(1- \varepsilon_{t-1})$

B1

$w_a = w_a * \varepsilon_{t-1}/(1- \varepsilon_{t-1})$

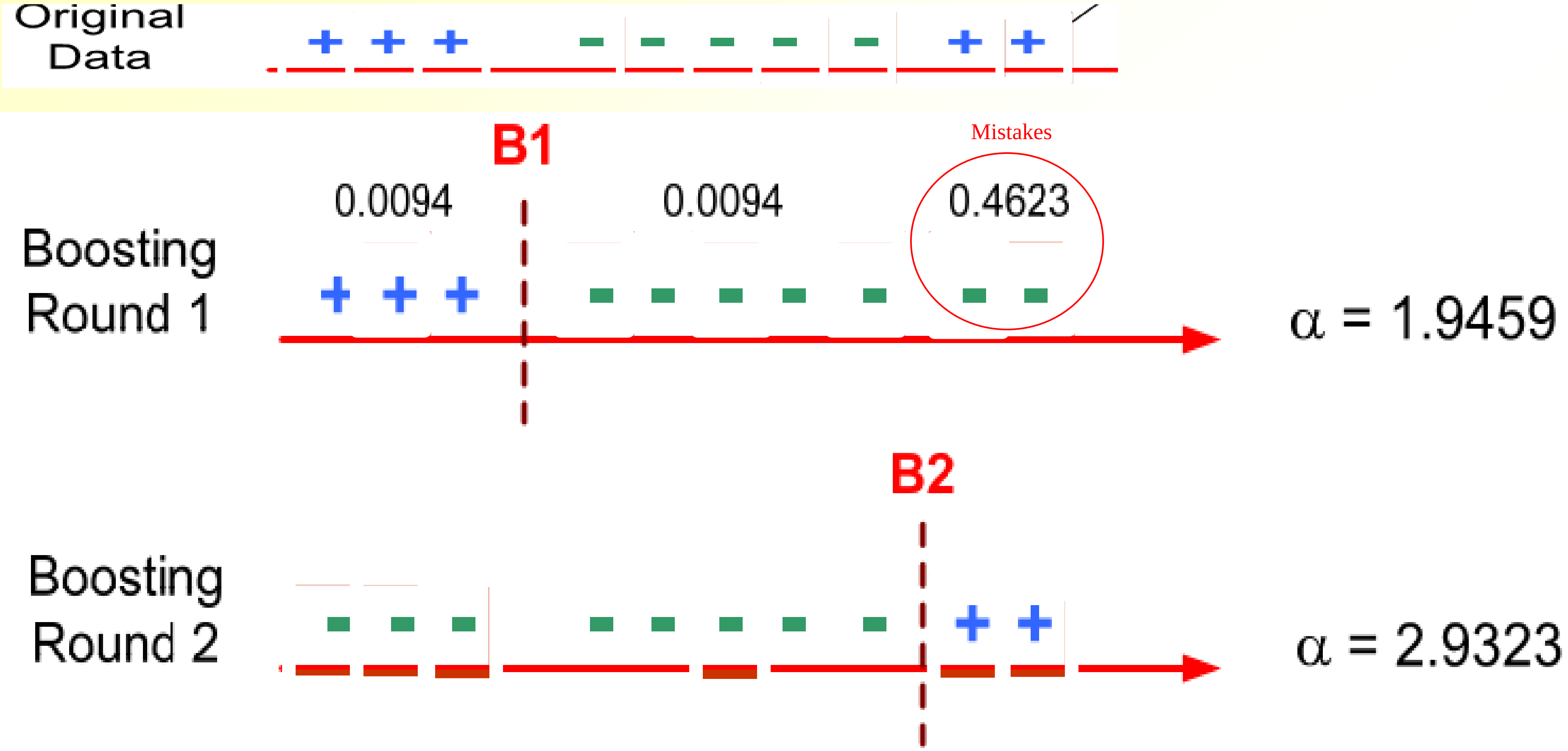
$w'_a = 0.1 * (1-\varepsilon_{i-1})/\varepsilon_{i-1}$

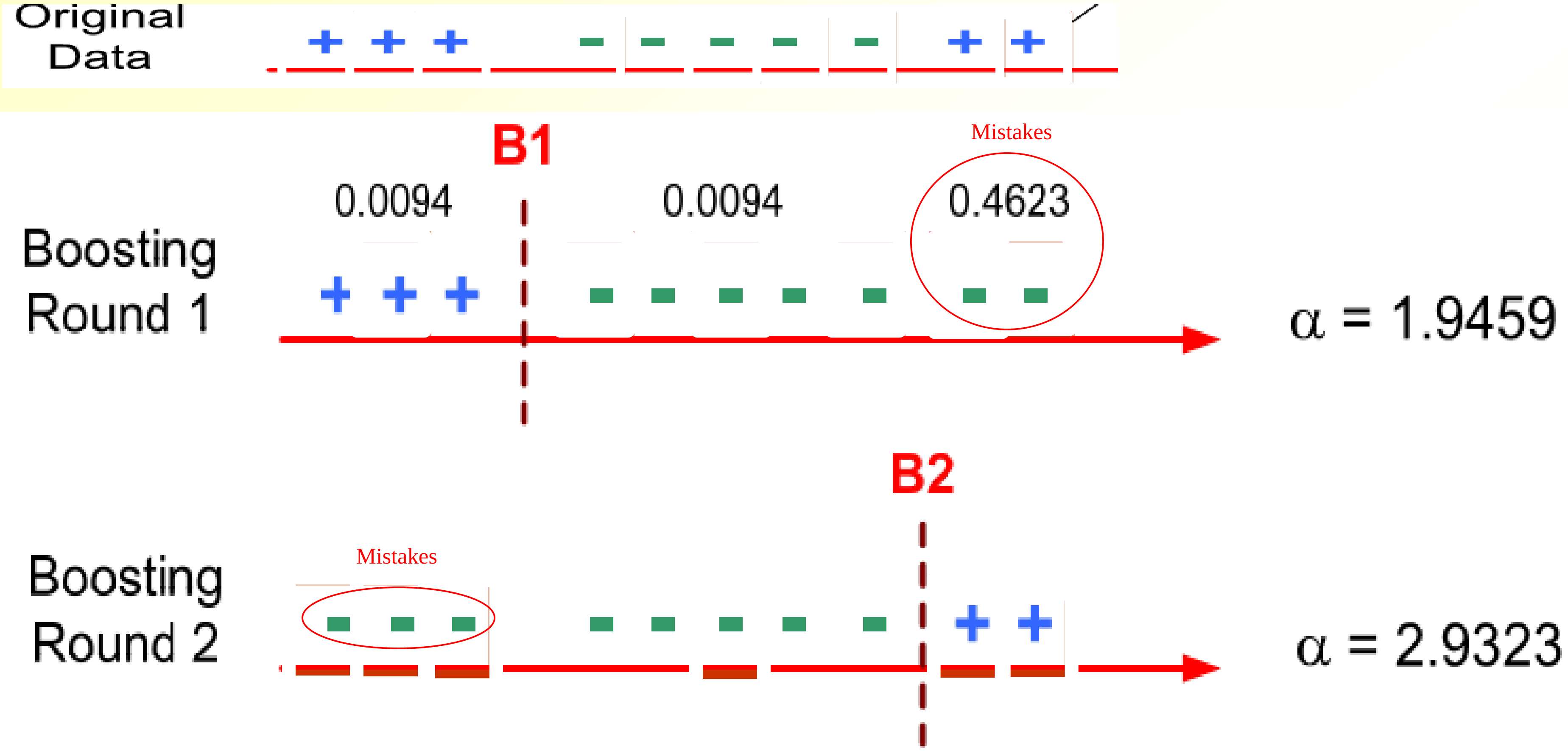
0.0094 0.0094 0.4623

Mistakes

α = 1.9459

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


Original Data
+ + + - - - - - - + +
B1
Mistakes
0.0094
0.0094
0.4623
Boosting Round 1
+ + + - - - - - - -
α = 1.9459
B2
Boosting Round 2
- - - - - - - - + +
α = 2.9323



Original Data

Boosting Round 1

B1

0.0094

0.0094

Mistakes

0.4623

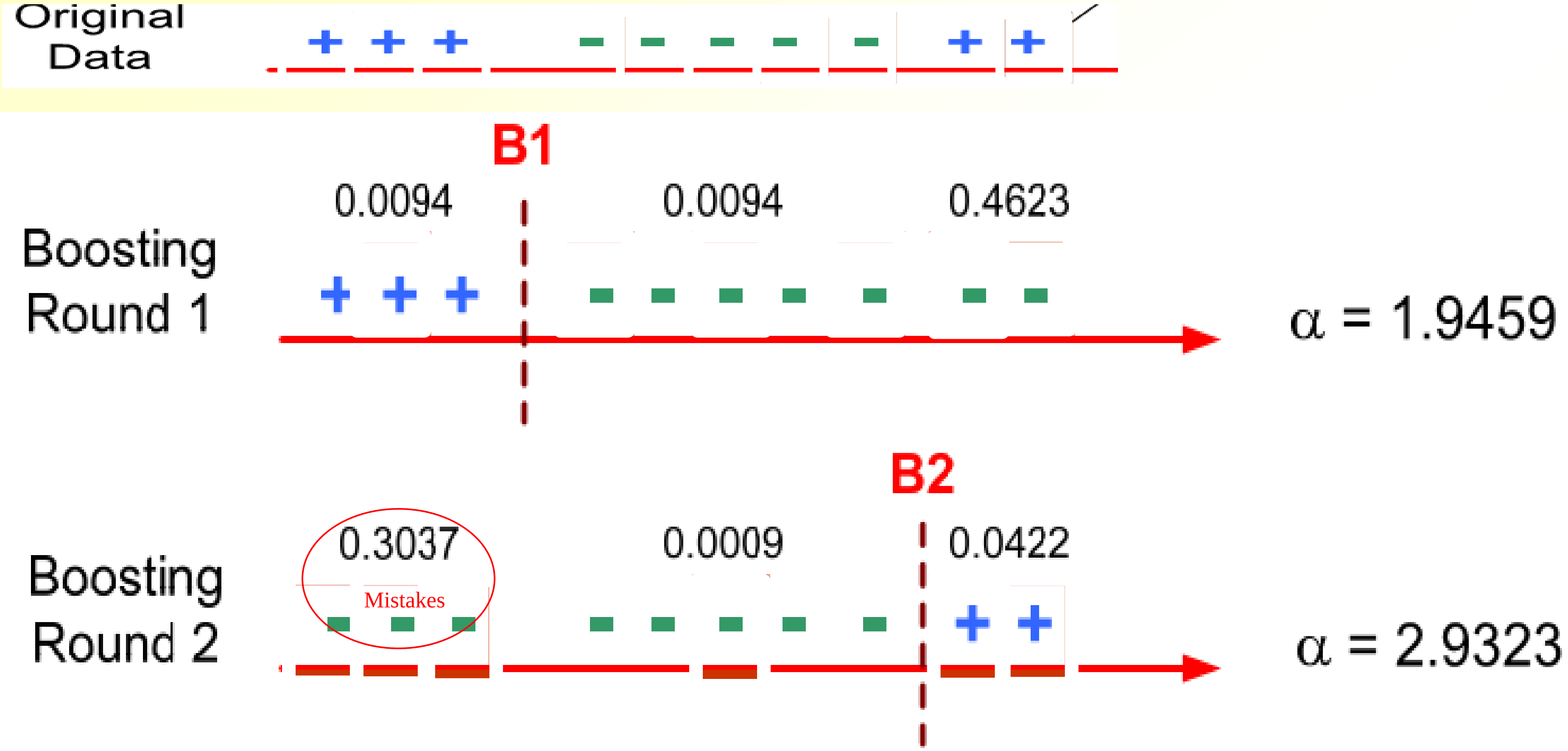
α = 1.9459

Boosting Round 2

B2

Mistakes

α = 2.9323



Original Data

B1

0.0094 0.0094 0.4623

Boosting Round 1

α = 1.9459

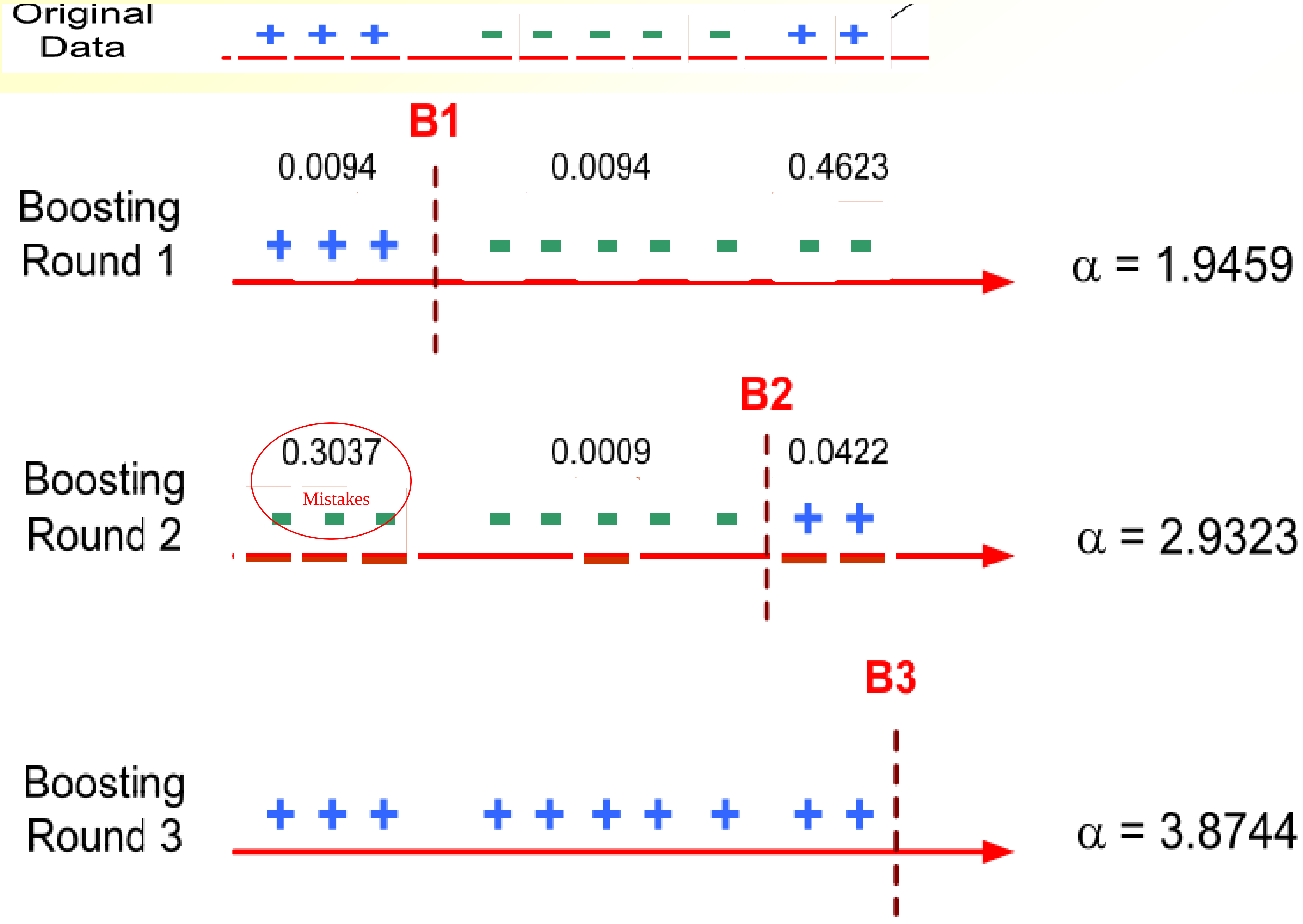
B2

0.3037 0.0009 0.0422

Mistakes

Boosting Round 2

α = 2.9323



Original Data

B1

0.0094 0.0094 0.4623

Boosting Round 1

$\alpha = 1.9459$

B2

0.3037 0.0009 0.0422

Mistakes

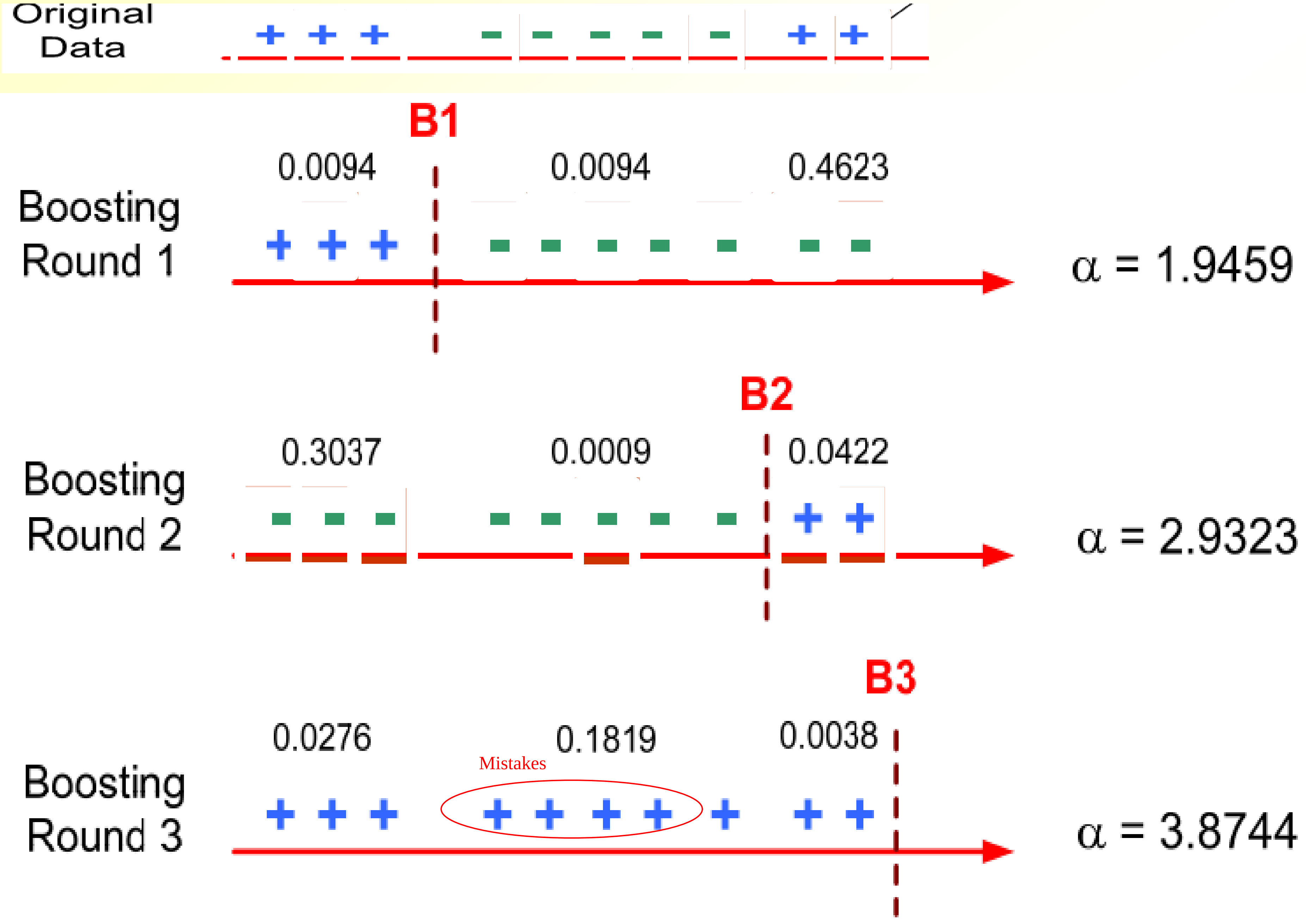
Boosting Round 2

$\alpha = 2.9323$

B3

Boosting Round 3

$\alpha = 3.8744$



Original Data

Boosting Round 1

B1

0.0094 0.0094 0.4623

α = 1.9459

Boosting Round 2

B2

0.3037 0.0009 0.0422

α = 2.9323

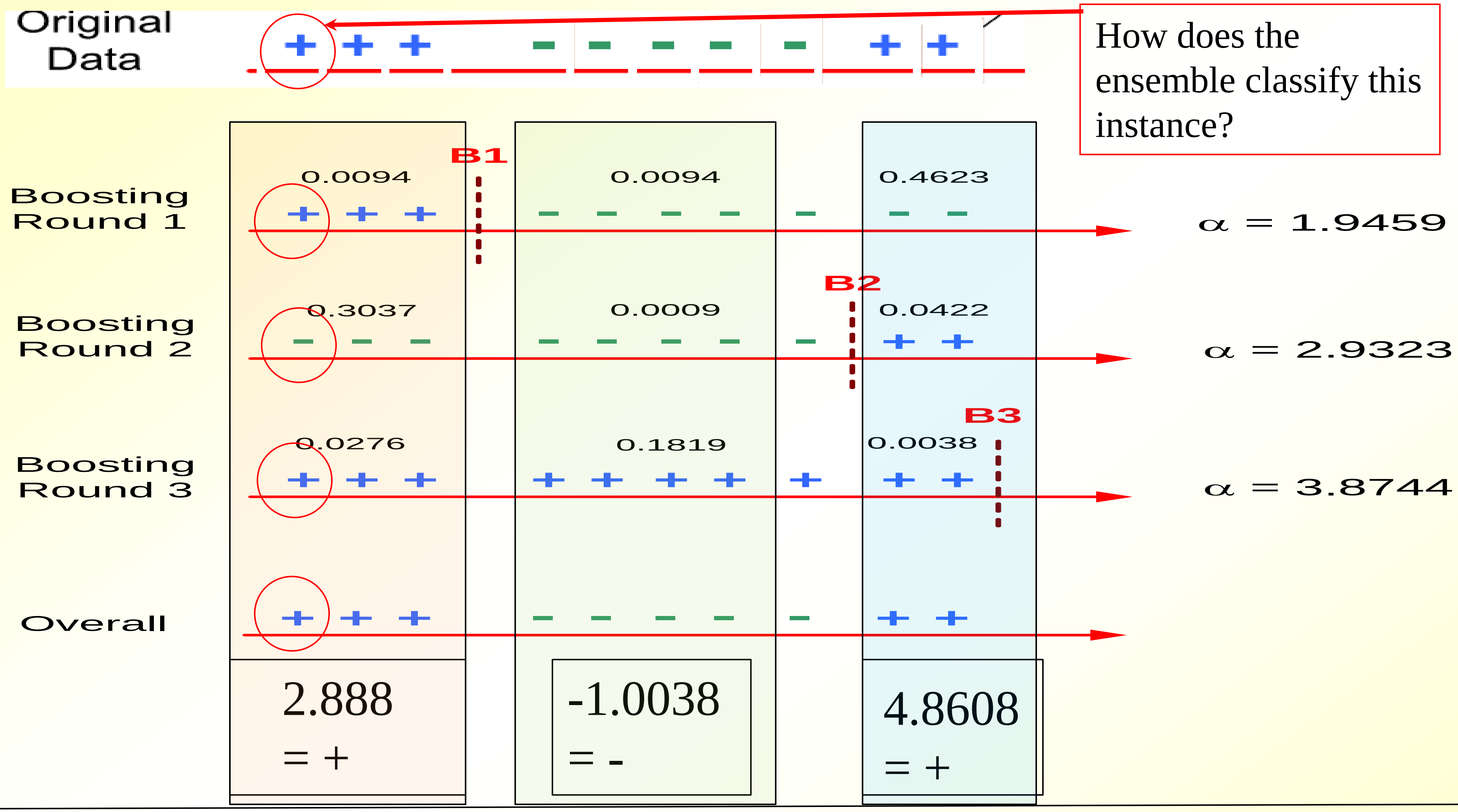
Boosting Round 3

B3

0.0276 0.1819 0.0038

Mistakes

α = 3.8744



Original Data

How does the ensemble classify this instance?

Boosting Round 1: 0.0094 | B1 | 0.0094 | 0.4623 — $\alpha = 1.9459$

Boosting Round 2: 0.3037 | 0.0009 | B2 | 0.0422 — $\alpha = 2.9323$

Boosting Round 3: 0.0276 | 0.1819 | B3 | 0.0038 — $\alpha = 3.8744$

Overall

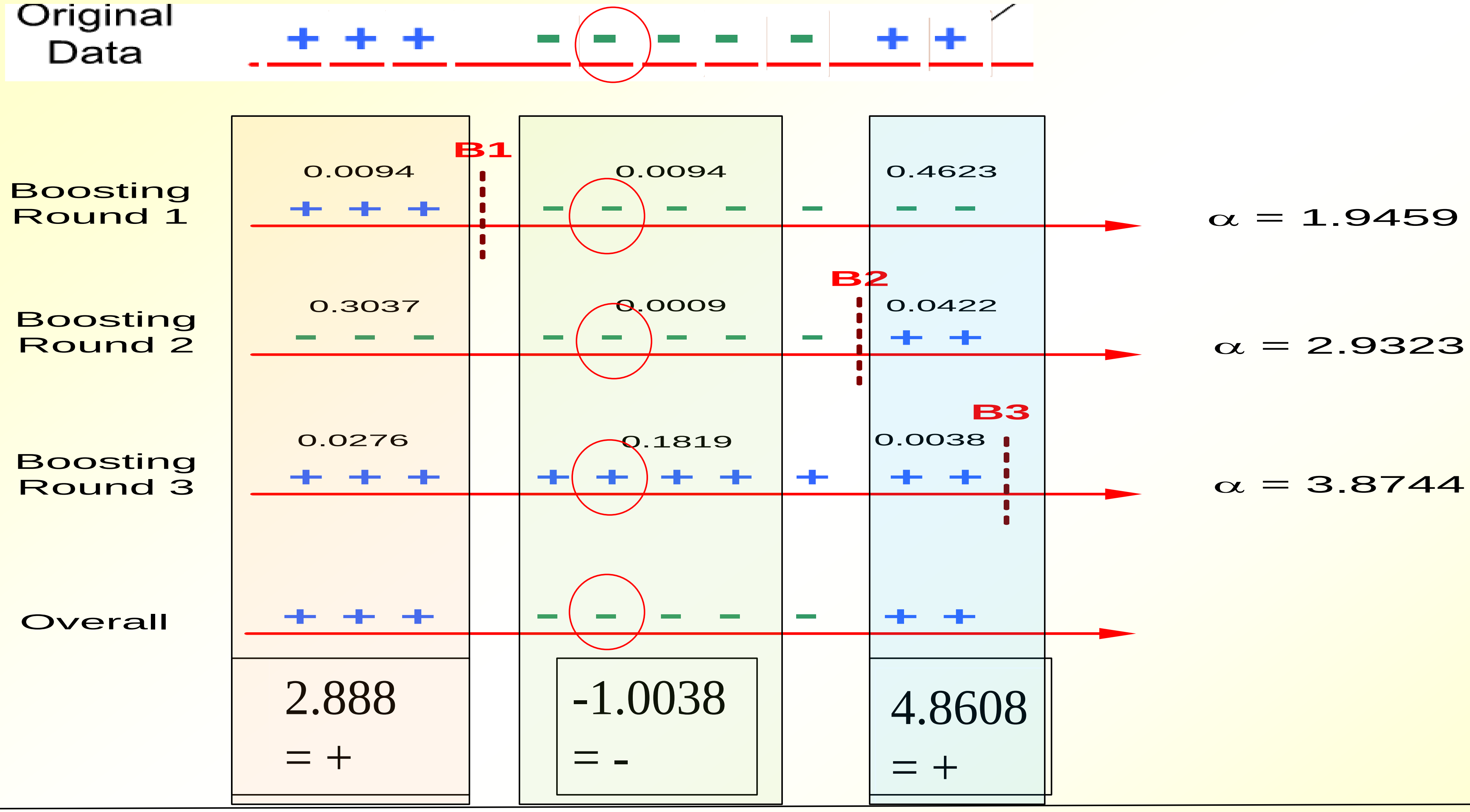
2.888 = +

-1.0038 = -

4.8608 = +

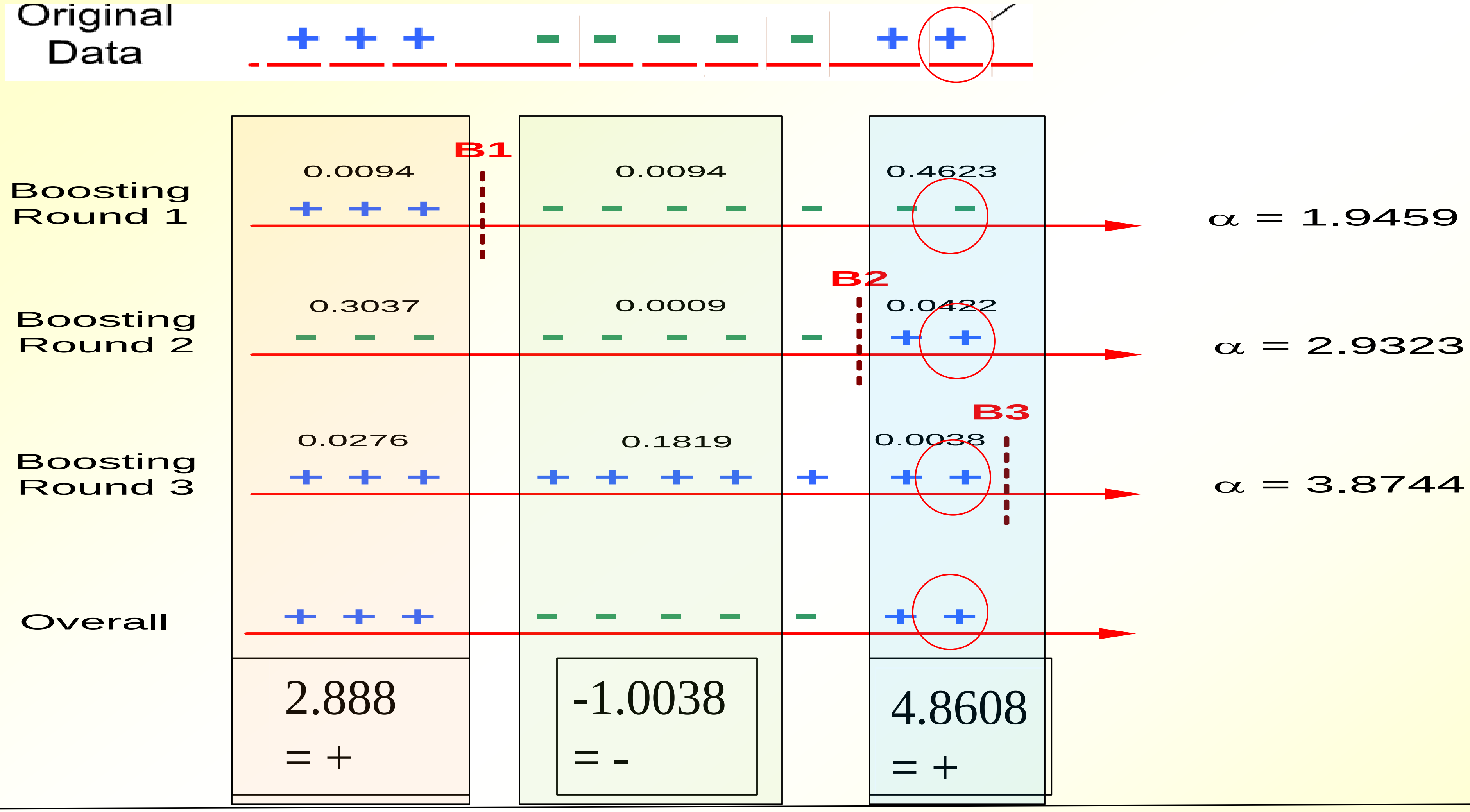
1.9459*B1(x) + 2.9323*B2(x)+ 3.8744*B3(x) = f(x)

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



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

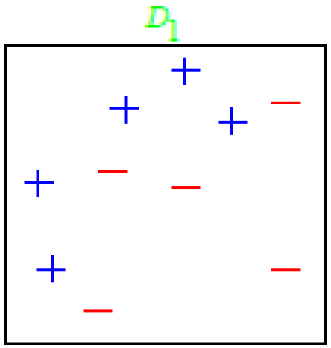
1.9459*(-1) + 2.9323*(-1)+ 3.8744*(+1) = -1.0038 < 0



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

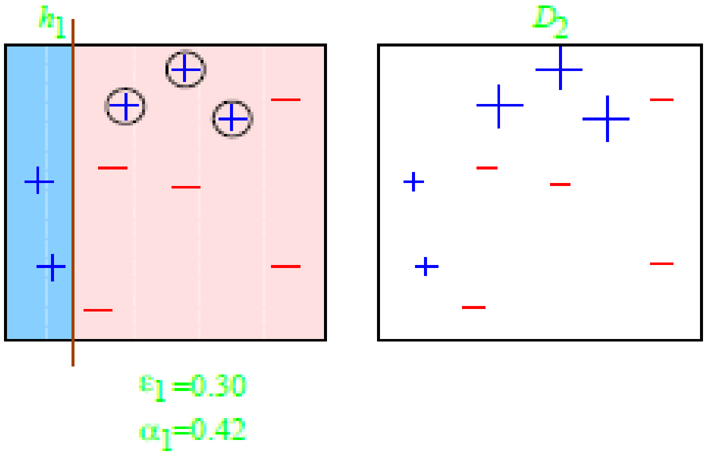
1.9459*(-1) + 2.9323*(+1)+ 3.8744*(+1) = 4.8608 > 0

Example of Adaboost in 2D

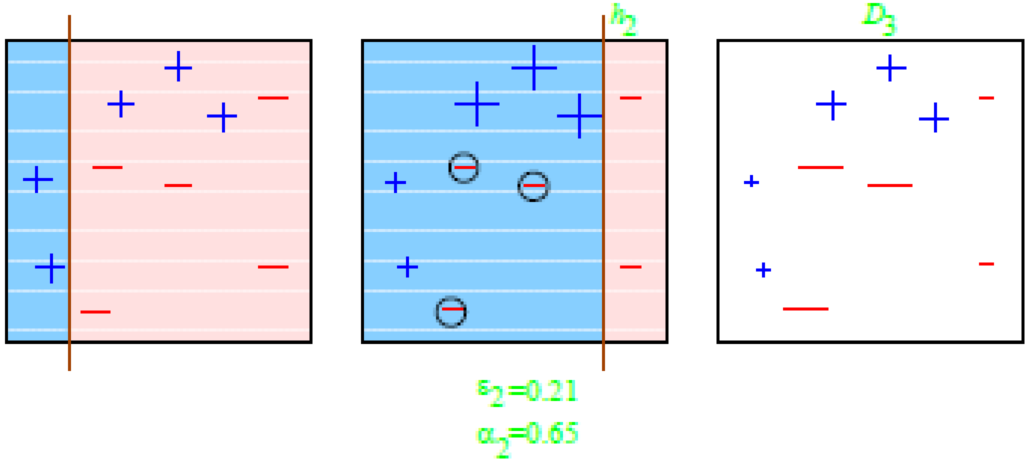


Weak hypotheses == vertical or horizontal half-planes

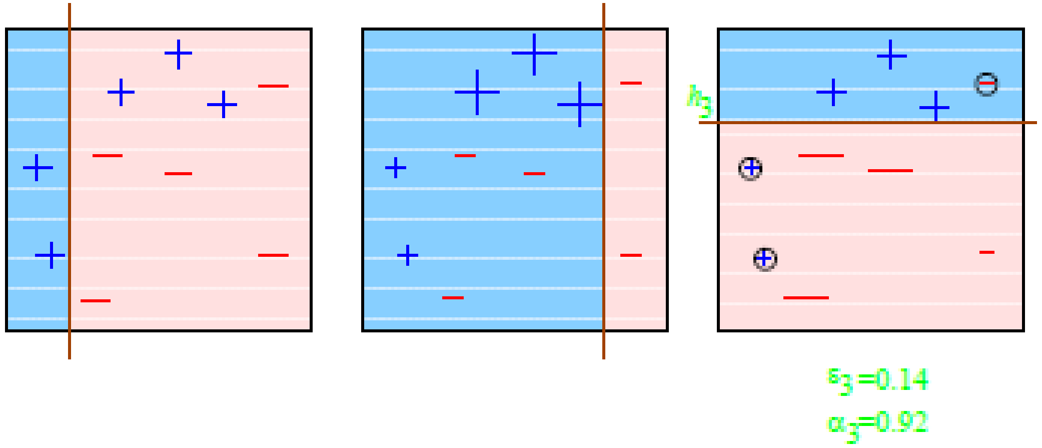
Round 1



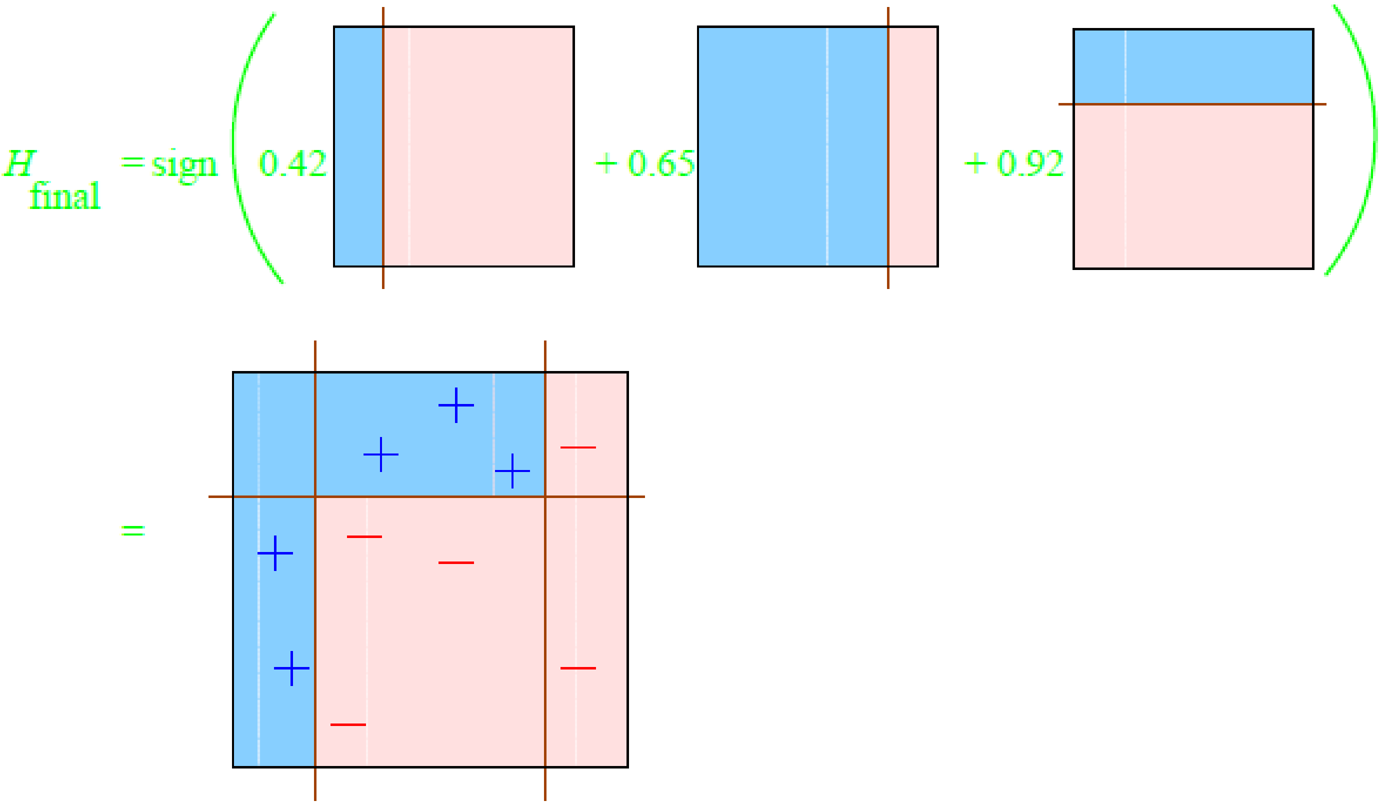
Round 2



Round 3



Final Hypothesis



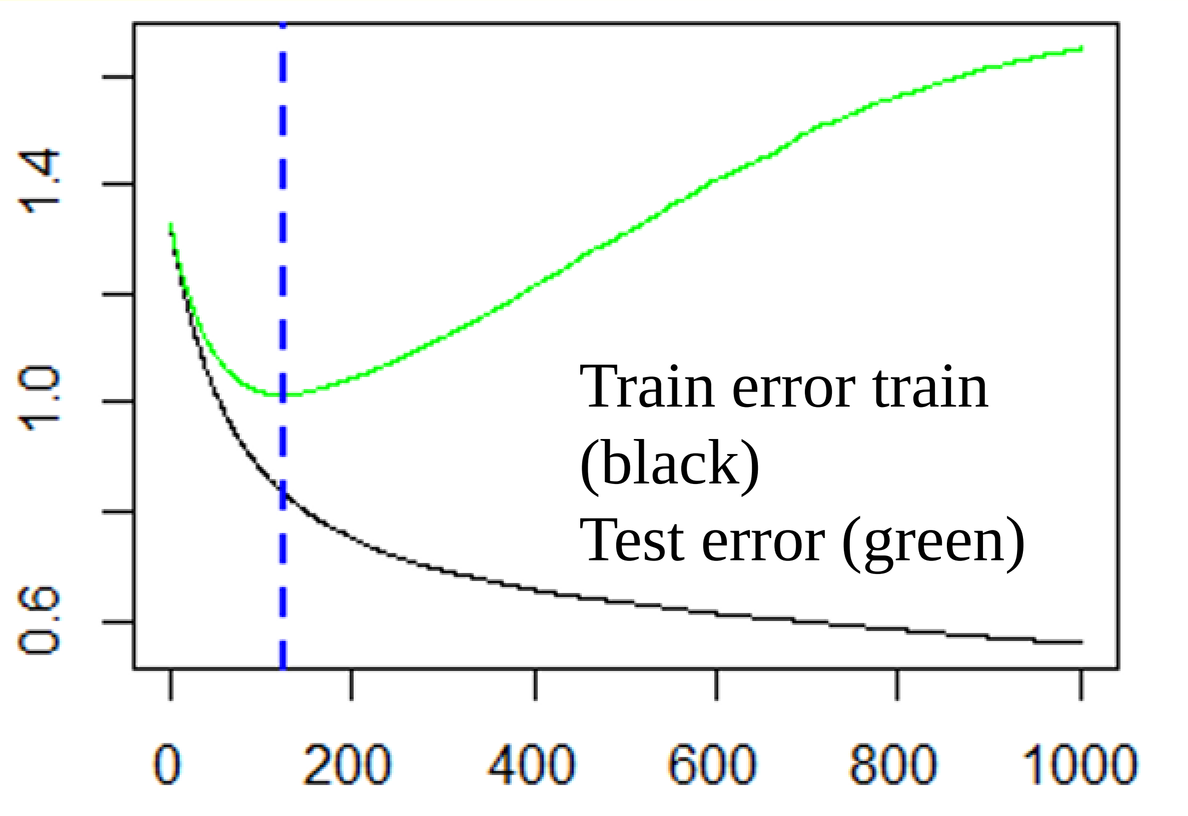
Source: Singer and Lewis Tutorial

Boosting. Summary

- It is one of the best ML algorithms, but it is sensitive to noise (class overlap). The reason is that Boosting models focus on instances difficult to classify by the previous model. But noisy instances are always difficult to classify. Hence, Boosting will insist in classifying instances that cannot be correctly classified, by memorizing them (and therefore, will incur in overfitting).

Possible overfitting

If number of base models is large. Noisy instances will eventually get very large weights (after all, they are difficult to classify) and eventually, introducing more and more base models in the ensemble will memorize the noise (overfitting)



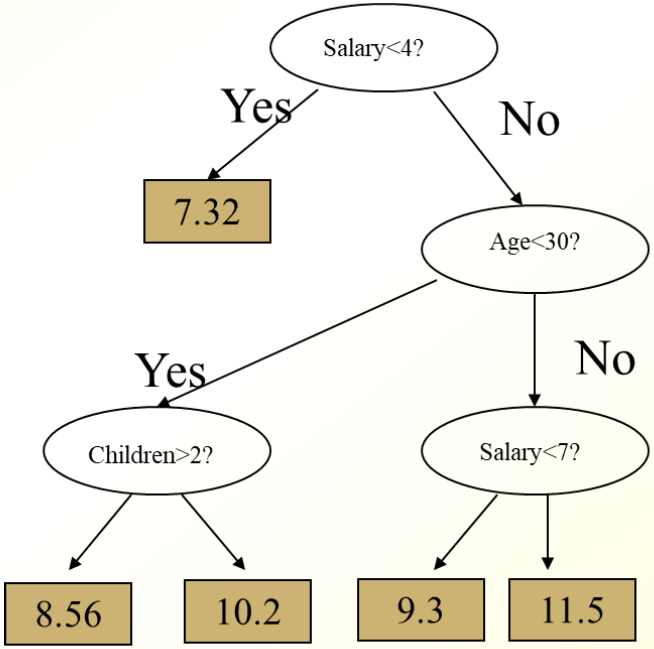
Train error train (black)

Test error (green)

Number of models in the ensemble

GRADIENT BOOSTING

- Adaptation of boosting for regression
- Typically, regression trees are used

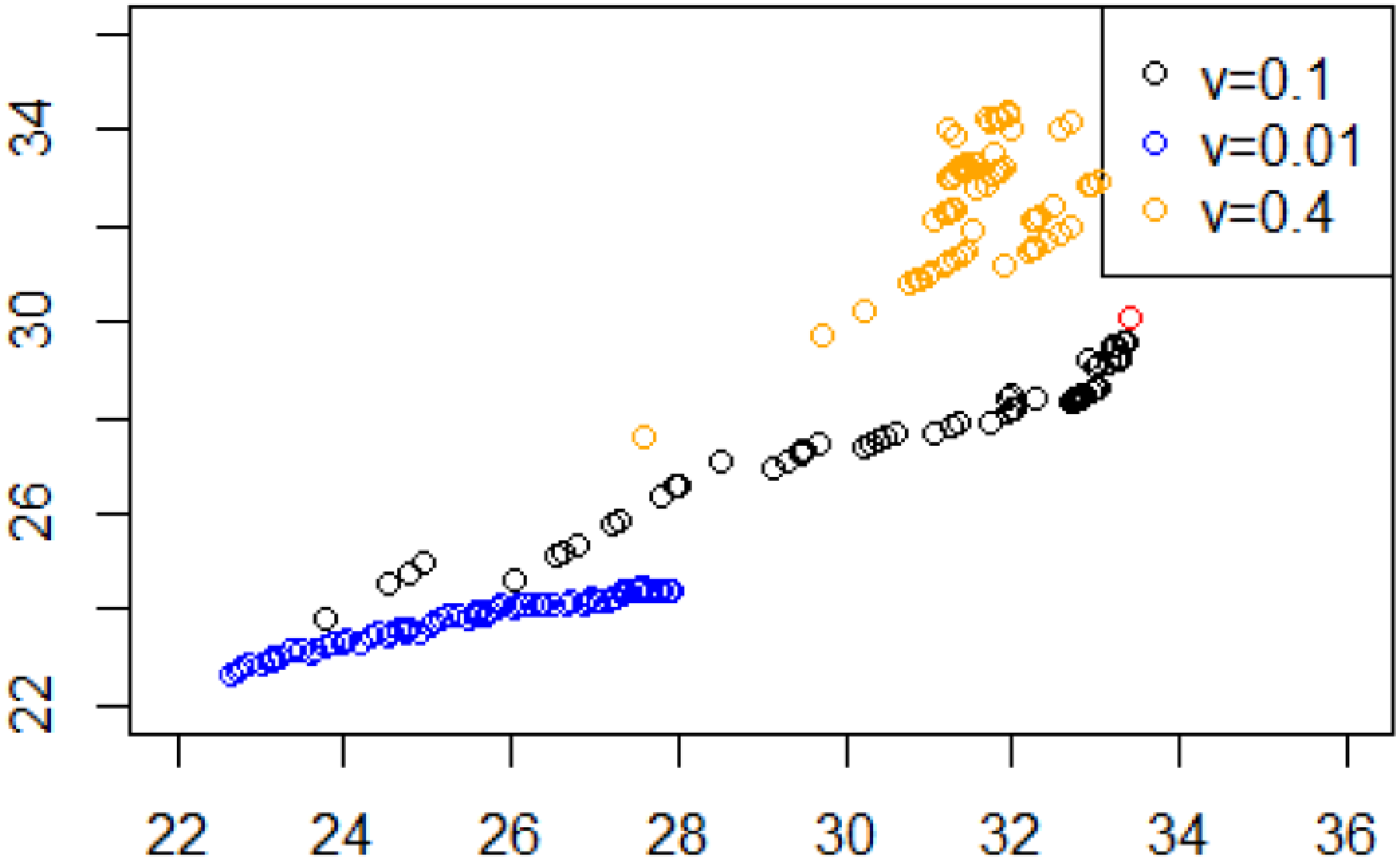


Boosting and overfitting

- Hyper-parameter T (number of trees): the larger, the more complex the model. Too complex models may result in overfitting.
  - **n_estimators** (in sklearn)
- **learning_rate** (0< α <1) (): $f_t(x) = f_{t-1}(x)+c_t\, \alpha\, h_{t-1}(x)$
  - Less overfitting because each model *h* has less influence on the final result
  - small α require more trees in the ensemble (larger T)
  - Sometimes the learning rate α is also called *v*

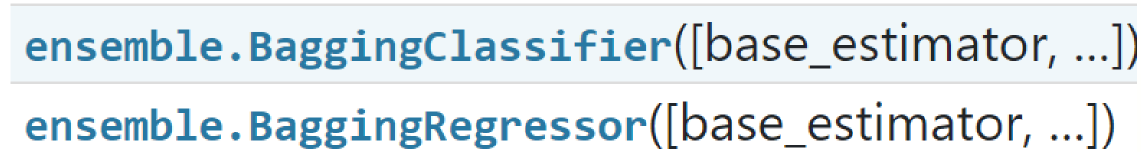
Different learning rate values

$v = \alpha =$ learning rate



ENSEMBLE IMPLEMENTATIONS.

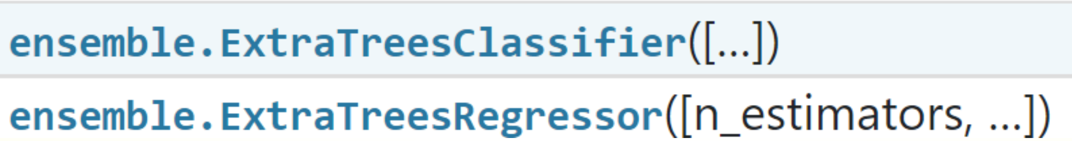
- In sklearn:
  - Bagging:



ensemble.BaggingClassifier([base_estimator, ...])

ensemble.BaggingRegressor([base_estimator, ...])

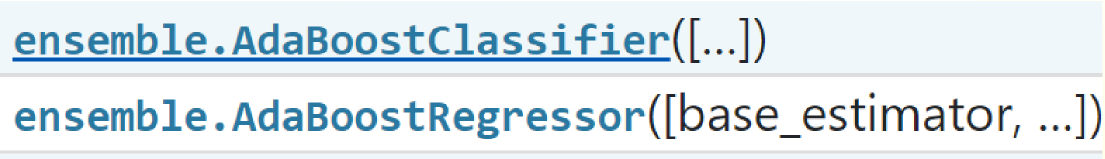
Faster:



ensemble.ExtraTreesClassifier([...])

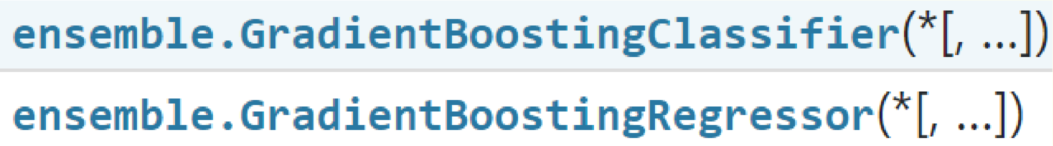
ensemble.ExtraTreesRegressor([n_estimators, ...])

  - Boosting:



ensemble.AdaBoostClassifier([...])

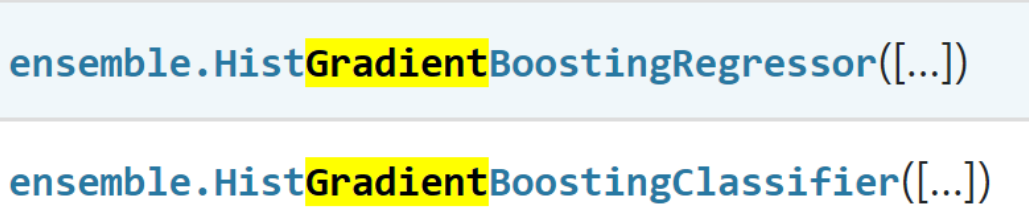
ensemble.AdaBoostRegressor([base_estimator, ...])



ensemble.GradientBoostingClassifier(*[, ...])

ensemble.GradientBoostingRegressor(*[, ...])

Faster:



ensemble.HistGradientBoostingRegressor([...])

ensemble.HistGradientBoostingClassifier([...])

ENSEMBLE IMPLEMENTATIONS.

- External to sklearn (but they can be used from within sklearn)
  - Boosting:
    - xgboost:
      - https://www.kaggle.com/stuarthallows/using-xgboost-with-scikit-learn
      - https://xgboost.readthedocs.io/en/latest/
    - lightgbm:
      - https://lightgbm.readthedocs.io/en/latest/
    - catboost:
      - https://catboost.ai/