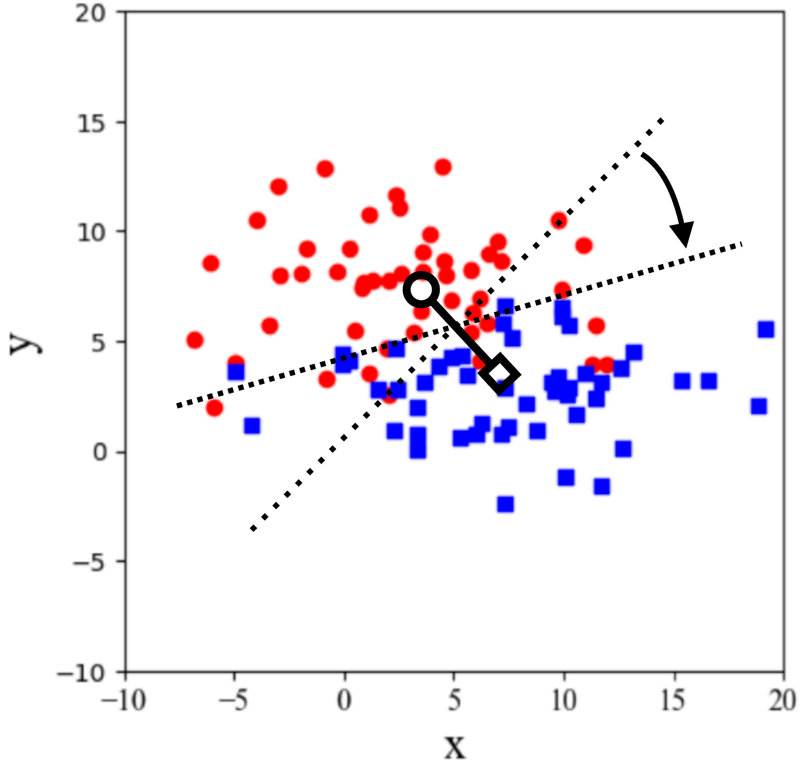
Lecture 12:

- Fisher Linear Discriminant
- Decision Trees
- Boosted Decision Trees (AdaBoost)

Fisher Linear Discriminant



Let's pick a new variable to act as a discriminant between the two classes that is some linear combination of x and y:

$$u \equiv ax + by$$

We want to choose values for a and b to maximise the mean distance (or variance) between the classes, while minimising the variances within each class so as to give the cleanest separation.

Fisher thus proposed maximising:

$$J = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \simeq \frac{(\overline{u_1} - \overline{u_2})^2}{s_1^2 + s_2^2}$$

$$J = \frac{[(a\overline{x_1} + b\overline{y_1}) - (a\overline{x_2} + b\overline{y_2})]^2}{[(as_{x_1})^2 + (bs_{y_1})^2] + [(as_{x_2})^2 + (bs_{y_2})^2]}$$

$$J = \frac{[(a(\overline{x_1} - \overline{x_2}) + b(\overline{y_1} - \overline{y_2})]^2}{[a^2(s_{x_1}^2 + s_{x_2}^2) + b^2(s_{y_1}^2 + s_{y_2}^2)]}$$

$$J = \frac{[(a(\overline{x_1} - \overline{x_2}) + b(\overline{y_1} - \overline{y_2})]^2}{[a^2(s_{x_1}^2 + s_{x_2}^2) + b^2(s_{y_1}^2 + s_{y_2}^2)]}$$

$$\frac{dJ}{da} = \frac{2[\ldots](\overline{x_1} - \overline{x_2})}{[--]} - \frac{[\ldots]^2}{[--]^2} 2a(s_{x_1}^2 + s_{x_2}^2) = 0$$

$$\frac{dJ}{db} = \frac{2[\ldots](\overline{x_1} - \overline{x_2})}{[--]} - \frac{[\ldots]^2}{[--]^2} 2b(s_{y_1}^2 + s_{y_2}^2) = 0$$

More Generally: $u = \vec{w}^T \vec{p}$

transpose vector of weights

vector of parameters

$$\vec{w} = (\Sigma_1 + \Sigma_2)^{-1}(\vec{\mu_1} - \vec{\mu_2})$$

class covariance matrices

vectors of class means

There is also a form that can be used for more than 2 classes, but the optimisation of this can be trickier

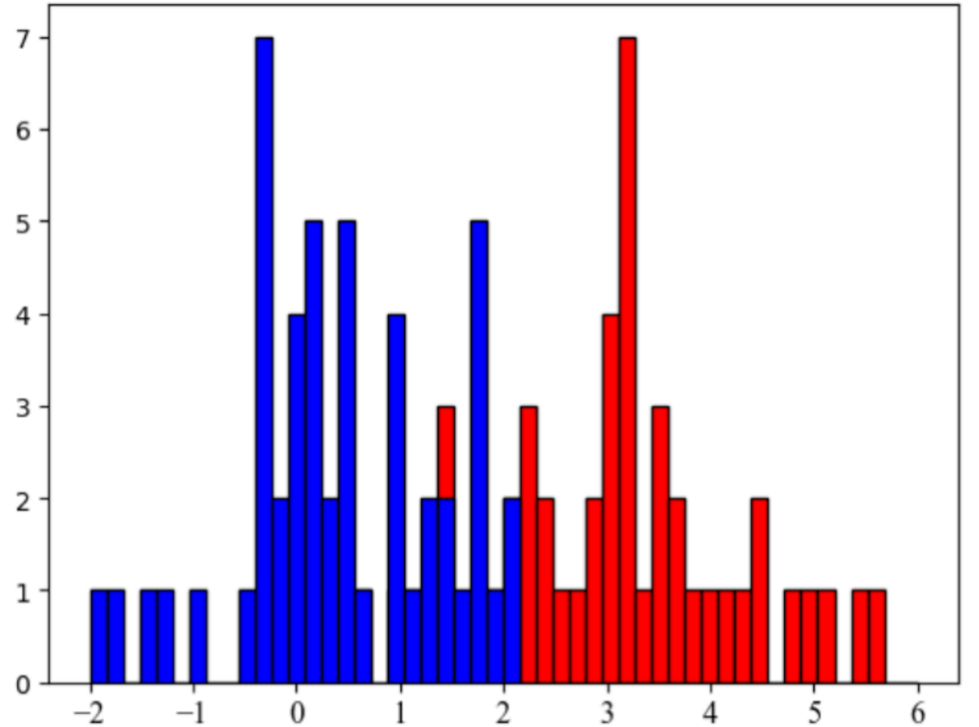
take ratios:

$$\frac{a}{b} = \frac{(\overline{x_1} - \overline{x_2})(s_{y_1}^2 + s_{y_2}^2)}{(\overline{y_1} - \overline{y_2})(s_{x_1}^2 + s_{x_2}^2)}$$

so choose

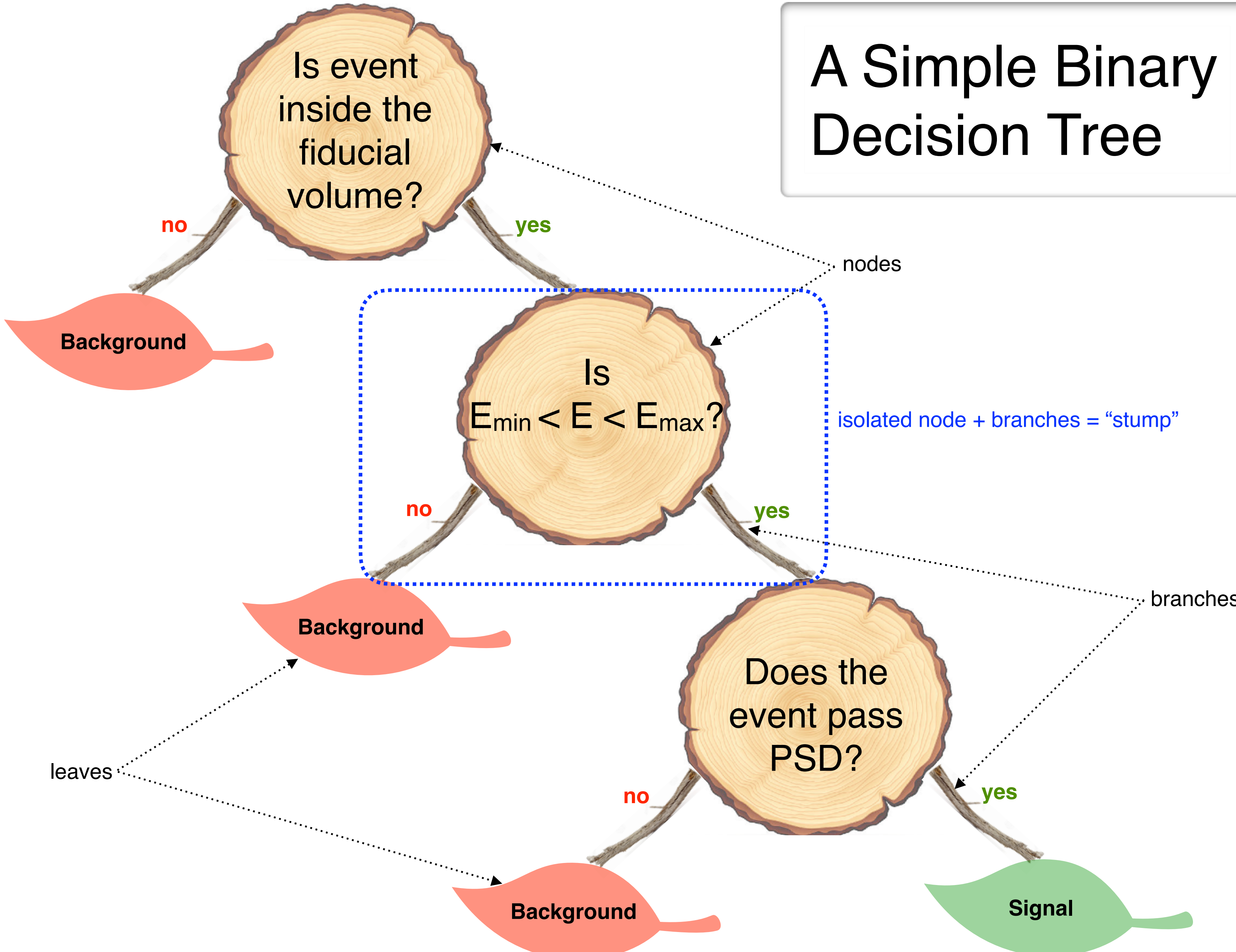
$$a = \frac{\overline{x_1} - \overline{x_2}}{s_{x_1}^2 + s_{x_2}^2} \qquad b = \frac{\overline{y_1} - \overline{y_2}}{s_{y_1}^2 + s_{y_2}^2}$$

"train" on data sets or simulation



$u = ax + by$

A Simple Binary Decision Tree



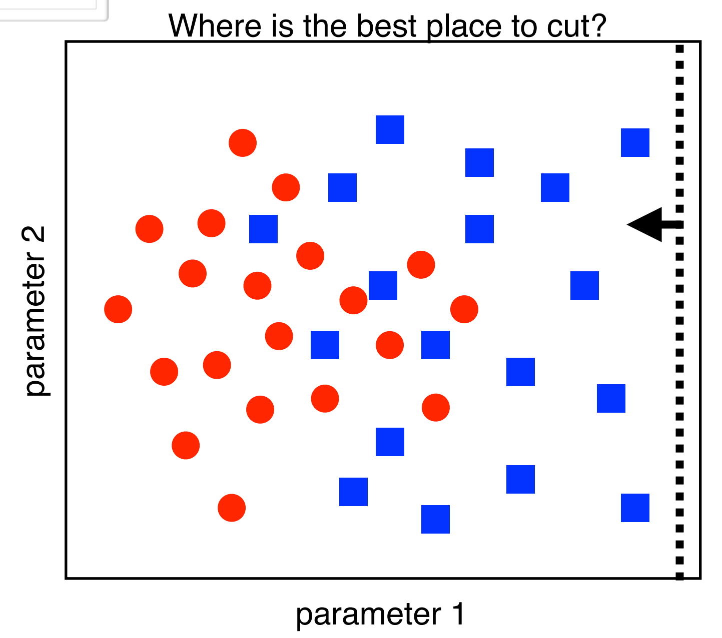
"Goodness of Split"

Purity of signal in the cut region:

$$p_s = \frac{n_s}{n_b + n_s}$$

Purity of background in the cut region:

$$p_b = \frac{n_b}{n_b + n_s} = 1 - p_s$$



Gini index (Corrado Gini): $I_G = p_s p_b = p_s(1 - p_s)$

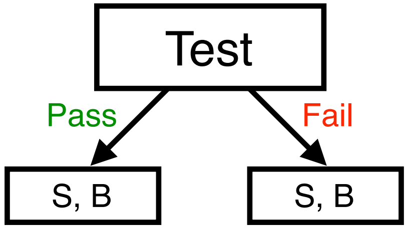
Note: equals zero for p_s or $p_b = 1$ (perfect separation) → Best separation at minimum Gini

More generally, for n classes, where p_i is the purity of the i^{th} target class:

$$I_G = \sum_{i=1}^{n} p_i(1 - p_i) = \sum (p_i - p_i^2)$$

$$= \sum p_i - \sum p_i^2 = 1 - \sum p_i^2$$

"Goodness of Split"



Weighted Gini index for test:

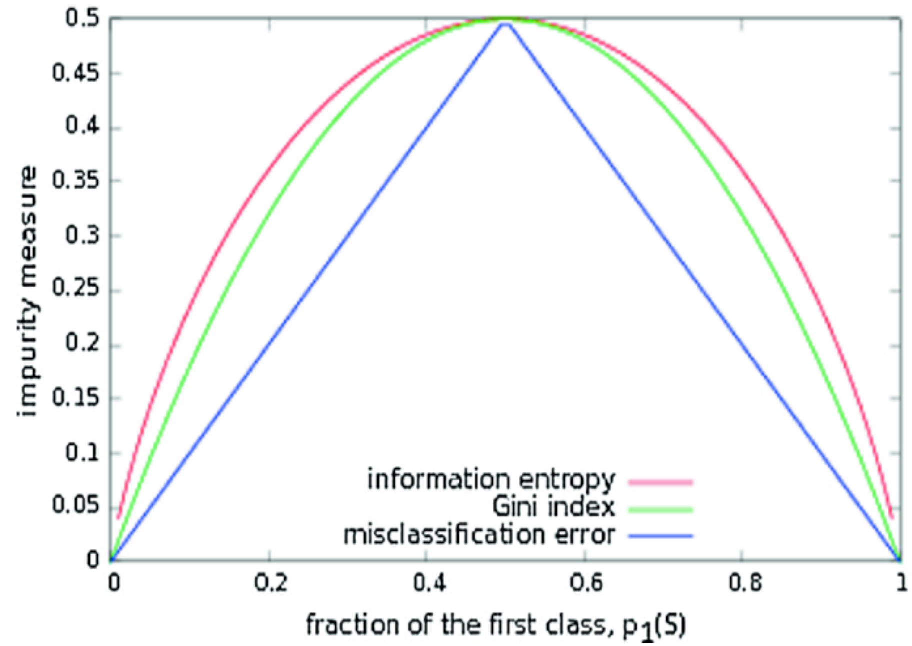
$$I_G(Tot) = f_P I_G(P) + f_F I_G(F)$$

What if the starting population is already unevenly split?

Use difference in Gini index (want to maximise):

$$\Delta I_G = I_G(0) - [f_P I_G(P) + f_F I_G(F)]$$

initial pre-split value for node



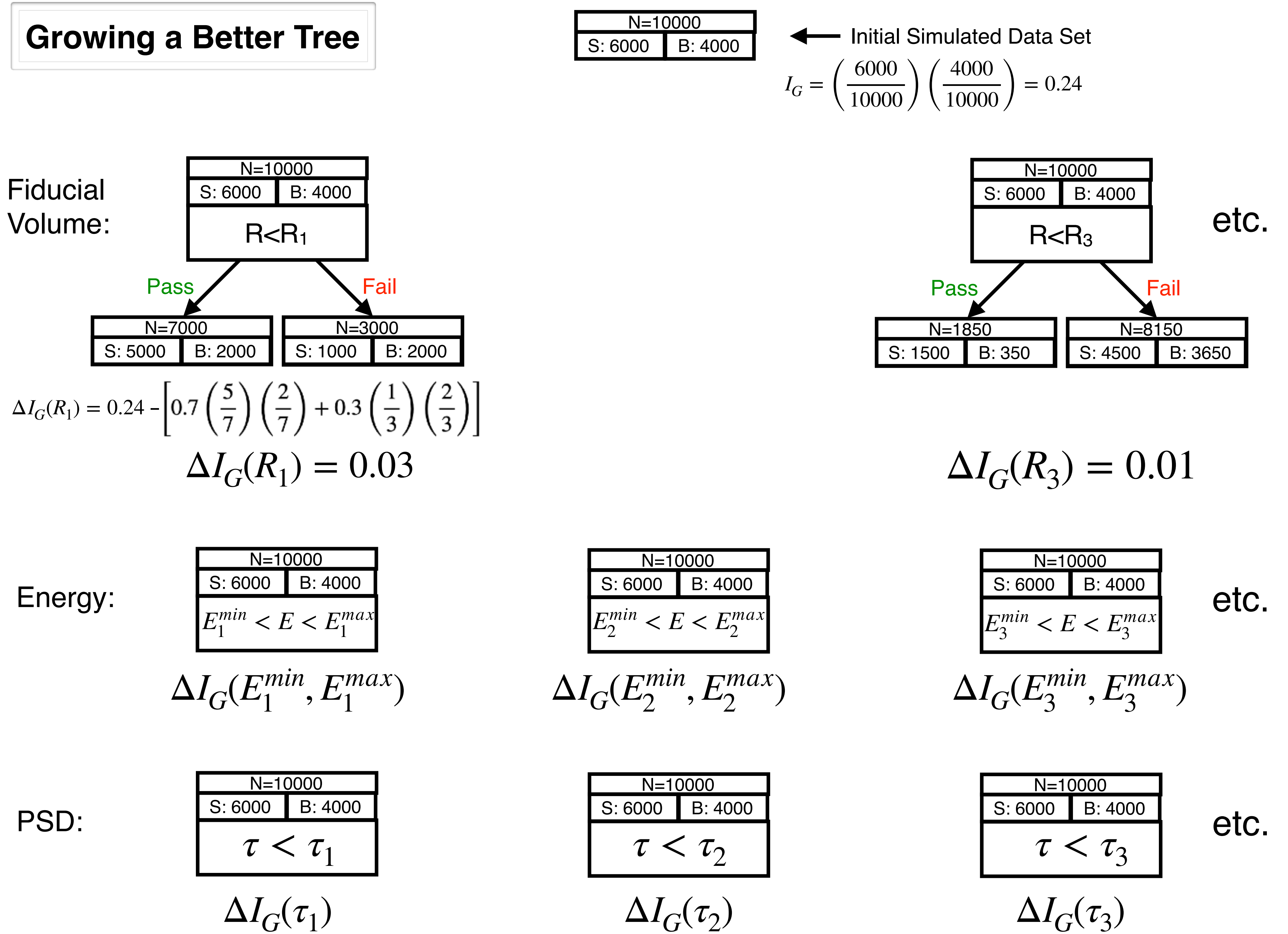
Other Examples of Measures:

Entropy: $I_E = -\sum p_i \log p_i$

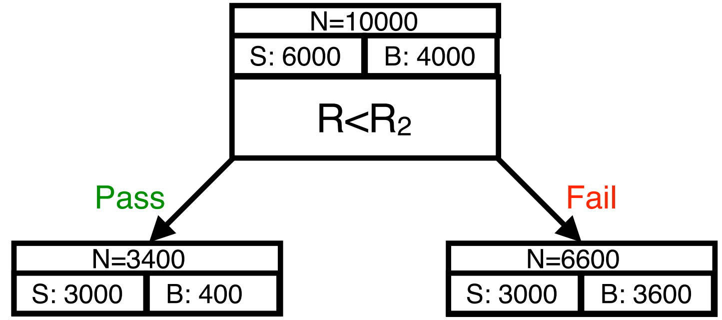
Misclassification Index: $I_M = \sum [1 - max(p_i\ ,\ 1 - p_i)]$

Significance: $I_S = \sum \frac{s_i^2}{b_i}$
(maximise)

Growing a Better Tree



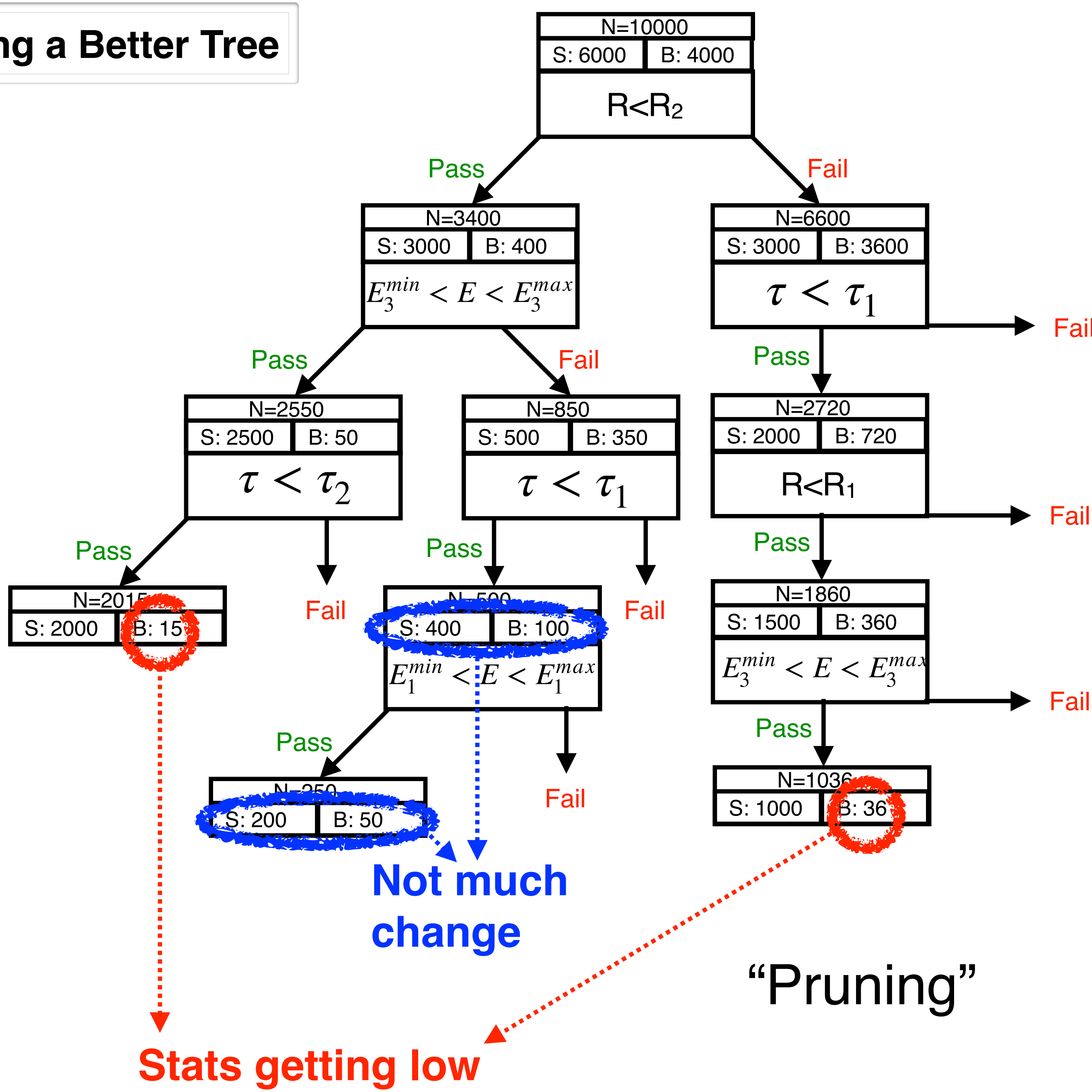
Growing a Better Tree



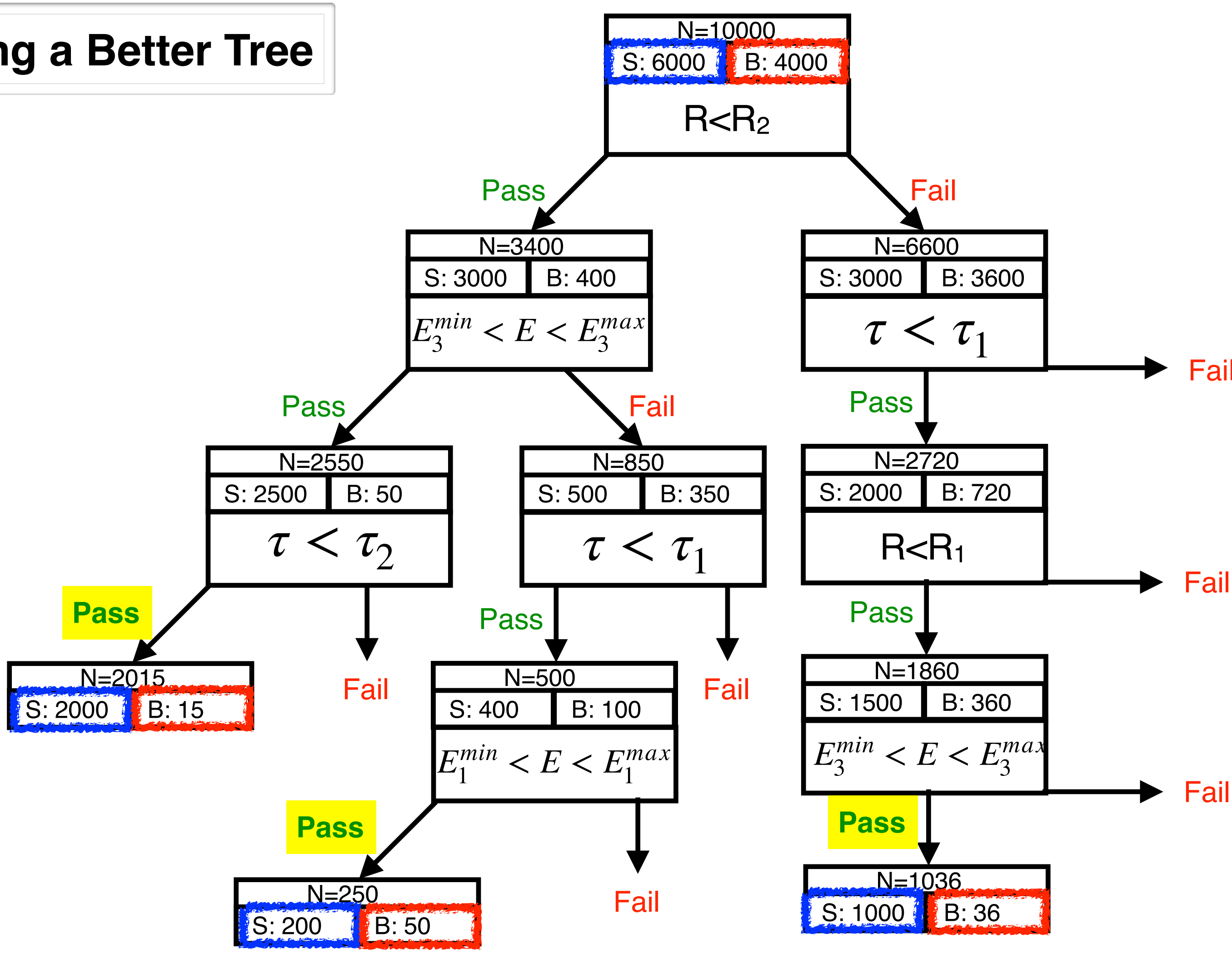
$$\Delta I_G(R_1)$$
$$\Delta I_G(R_2)$$
$$\Delta I_G(R_3)$$
$$\vdots$$
$$\Delta I_G(E_1^{min}, E_1^{max})$$
$$\Delta I_G(E_2^{min}, E_2^{max})$$
$$\Delta I_G(E_3^{min}, E_3^{max})$$
$$\vdots$$
$$\Delta I_G(\tau_1)$$
$$\Delta I_G(\tau_2)$$
$$\Delta I_G(\tau_3)$$
$$\vdots$$

$$\Delta I_G(R_1)$$
$$\Delta I_G(R_2)$$
$$\Delta I_G(R_3)$$
$$\vdots$$
$$\Delta I_G(E_1^{min}, E_1^{max})$$
$$\Delta I_G(E_2^{min}, E_2^{max})$$
$$\Delta I_G(E_3^{min}, E_3^{max})$$
$$\vdots$$
$$\Delta I_G(\tau_1)$$
$$\Delta I_G(\tau_2)$$
$$\Delta I_G(\tau_3)$$
$$\vdots$$

Growing a Better Tree



Growing a Better Tree



Signal efficiency : $$\frac{3200}{6000} = 53\ \%$$

Background efficiency : $$\frac{101}{4000} = 2.5\ \%$$

Boosted Trees (AdaBoost)

Assume we have a data set with relevant parameter values for a given test: $x_1, x_2, x_3 \ldots x_N$

each of which corresponds to a given class: $q_1, q_2, q_3 \ldots q_N$

where, for example, $q_i = 1$ if it's signal & $q_i = -1$ if background.

Test

$$\delta(x) \begin{cases} \text{for example} \\ = 1 & \text{if Pass} \\ = -1 & \text{if Fail} \end{cases}$$

Assume we have some arbitrary number of test results from a series of "weak learners":

$$\delta_1(x_i), \delta_2(x_i), \delta_3(x_i) \ldots \delta_L$$

and that we wish to find a strong classifier that is a linear combination of these:

$$C_L(x_i) = \sum_{j=1}^{L} \alpha_j \delta_j(x_i)$$

where the sign of C_L indicates the preferred class and the magnitude is related to the strength of the classification.

Further assume an exponential "loss function" to penalise incorrect classifications within an "error function":

$$E = \sum_{i=1}^{N} e^{-q_i C(x_i)}$$

*Other boost algorithms, loss functions and classifier combinations are available at specially selected stores!

$$E = \sum_{i=1}^{N} e^{-q_i C(x_i)} \qquad C_L(x_i) = \sum_{j=1}^{L} \alpha_j \delta_j(x_i)$$

Assume we have a classifier composed of m-1 weak learners and we wish to add another: $C_m(x_i) = C_{m-1}(x_i) + \alpha_m \delta_m(x_i)$

What choice of α_m will minimise E ?

$$E = \sum_{i=1}^{N} e^{-q_i C_{m-1}(x_i)} \; e^{-q_i \alpha_m \delta_m(x_i)} = \sum_{i=1}^{N} w_i^m \; e^{-q_i \alpha_m \delta_m(x_i)} \qquad (w_i^1 \equiv 1)$$

relative weights

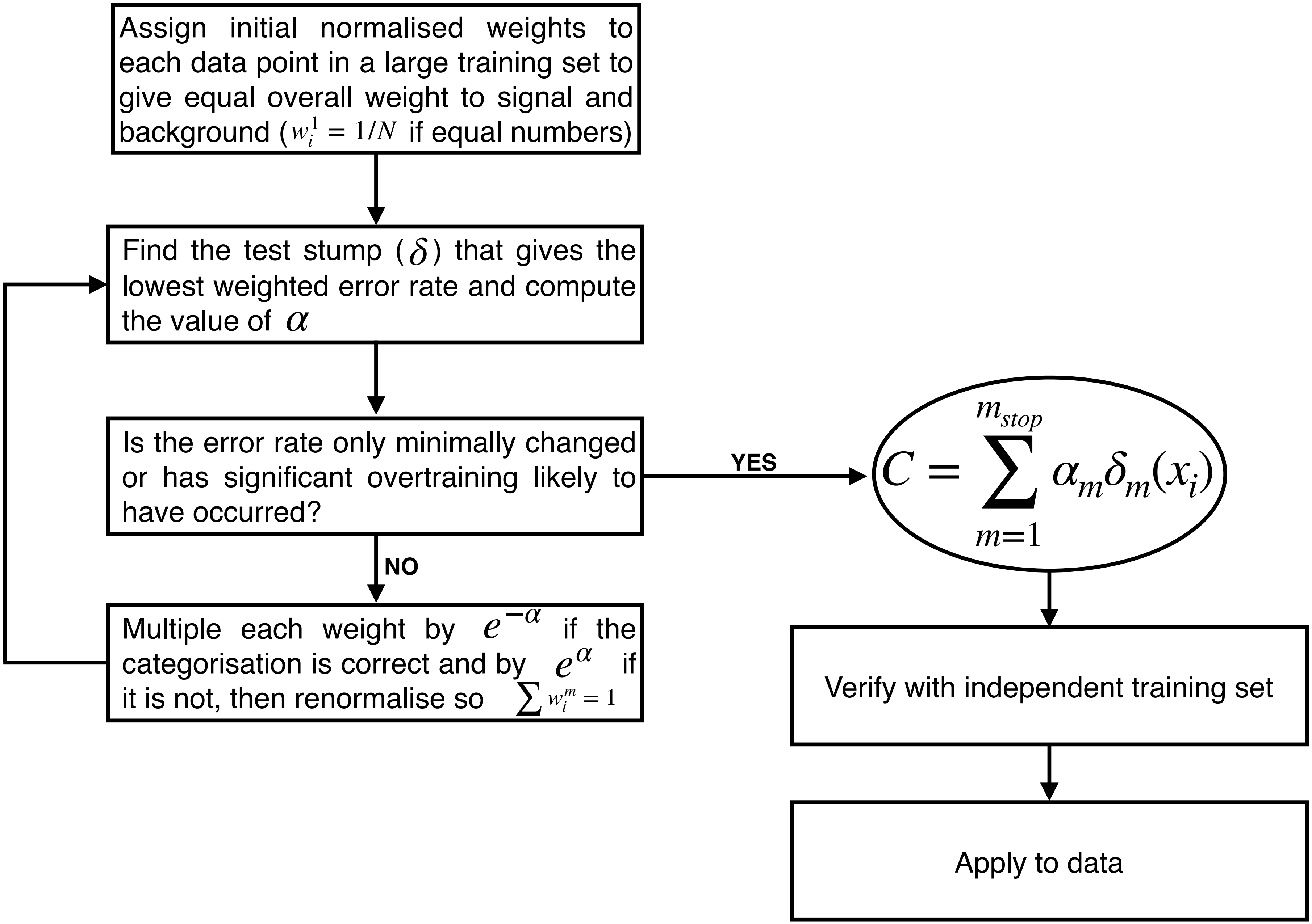
$$= \sum_{q_i = \delta_m(x_i)} w_i^m \; e^{-\alpha_m} + \sum_{q_i \neq \delta_m(x_i)} w_i^m \; e^{\alpha_m}$$

$$\frac{dE}{d\alpha_m} = -\,\alpha_m e^{-\alpha_m} \sum_{q_i = \delta_m(x_i)} w_i^m \; + \; \alpha_m e^{-\alpha_m} \sum_{q_i \neq \delta_m(x_i)} w_i^m = 0$$

$$\boxed{\alpha_m = -\frac{1}{2} \ln \left(\frac{\sum_{q_i \neq \delta_m(x_i)} w_i^m}{\sum_{q_i = \delta_m(x_i)} w_i^m} \right) = \frac{1}{2} \ln \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)} \qquad \epsilon_m \equiv \frac{\sum_{q_i \neq \delta_m(x_i)} w_i^m}{\sum_i^N w_i^m}$$

weighted fractional error rate

AdaBoost Implementation



Some Observations:

- If the problem can be completely specified by PDFs that capture the relevant information, then you cannot do better than likelihood!
- The boost algorithm, loss function and classifier combination is arbitrary and not unique. There is no theorem that says which set of these is the best or produces the best possible discrimination.
- BDTs **will** overtrain! It is therefore important to pay attention to convergence criteria and verify the final efficiency with independent training sets.
- The use of **too many extraneous or redundant parameters** will make it more likely for BDTs to get distracted by fluctuations in multiple dimensions, resulting in a failure to converge on the relevant region and leading to a loss in efficiency. It's worth putting thought into the parameter choices and building elements one by one.
- *You don't get the likelihood and all the benefits that brings.*
- BDTs and other ML approaches are particularly useful if computational speed is an issue or it is difficult to couch the problem in terms of PDFs (i.e. simple hypotheses).