Lecture 13:

Machine Learning & Decision Trees

- Supervised vs Unsupervised Learning
- Decision Trees
- Random Forests
- Boosted Decision Trees (AdaBoost)

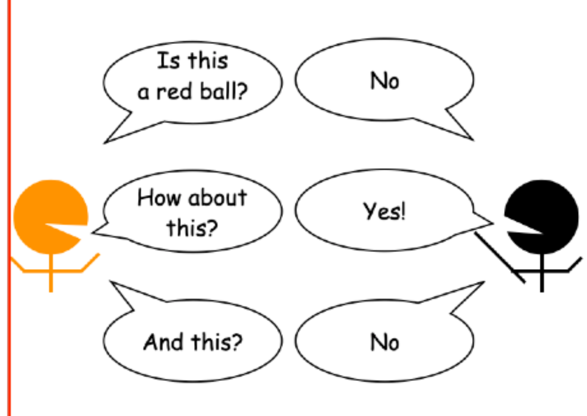
In machine learning, an algorithm to achieve a particular goal is pieced together through a series of iterative guesses on an example of data where the solution is known, guided by a “loss function” that quantifies how well the algorithm is doing.

Generally, there are two basic types of training, known as “supervised” and “unsupervised,” which refers to whether or not an individually ‘labeled’ data set is used. The exact definition is a little fuzzy and the distinction can sometimes blur… but here’s a simple example of the two approaches:

Work out how to find the number of red balls in a bag:

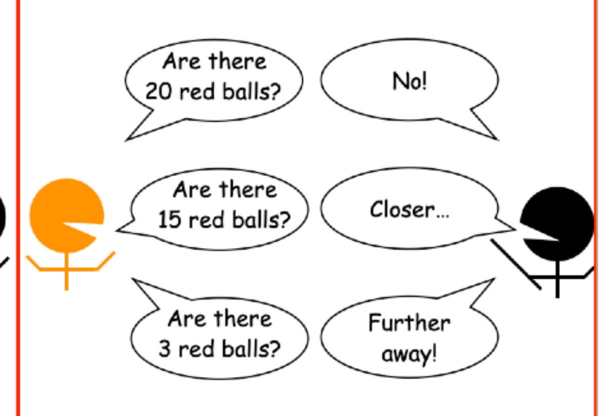
Supervised Training

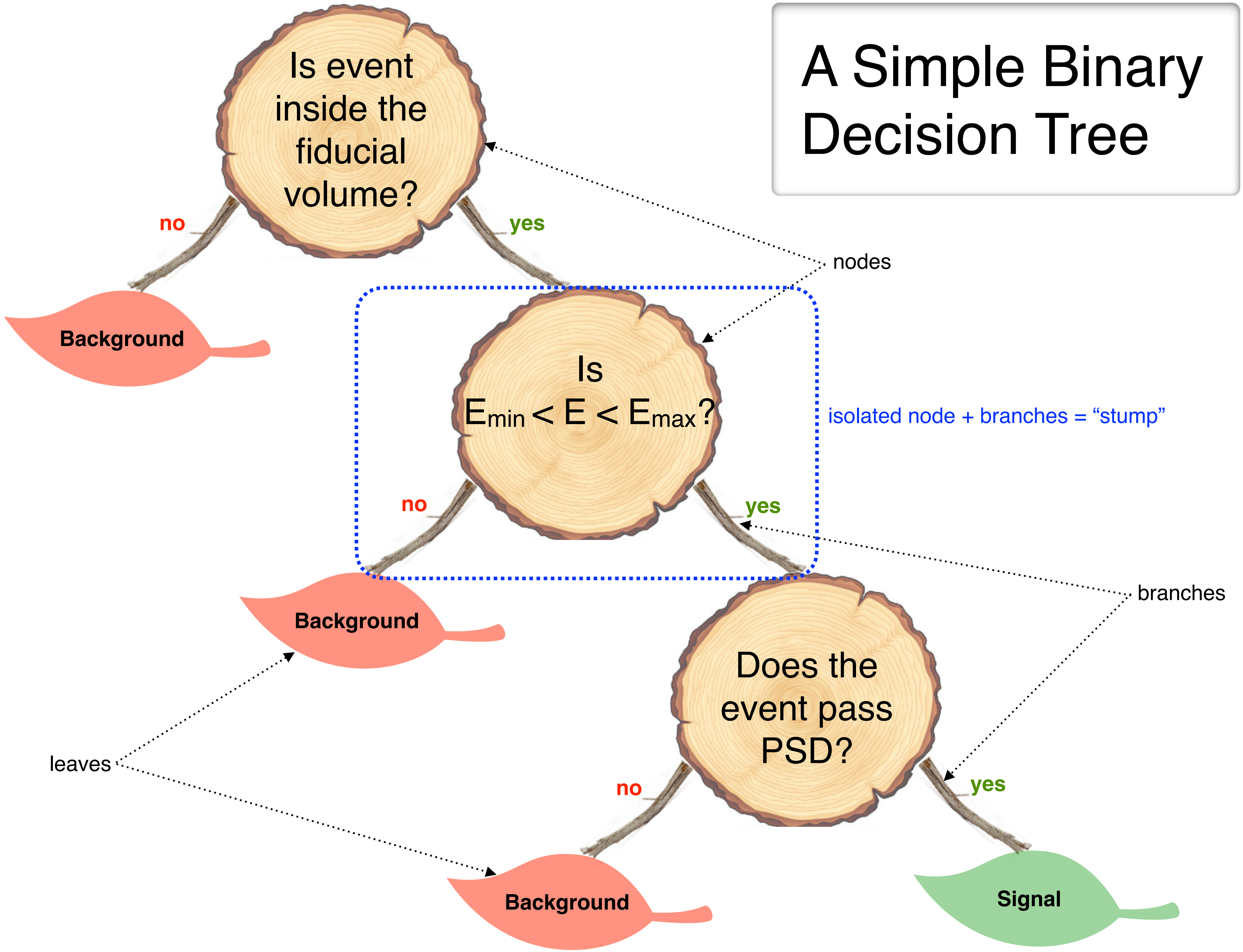
(teach the methodology)



Unsupervised Training

(allow the methodology to be inferred from the training sets)





A Simple Binary Decision Tree
Is event inside the fiducial volume?
no
yes
Background
nodes
Is $E_{min} < E < E_{max}$?
isolated node + branches = “stump”
no
yes
Background
branches
Does the event pass PSD?
leaves
no
yes
Background
Signal

"Goodness of Split"

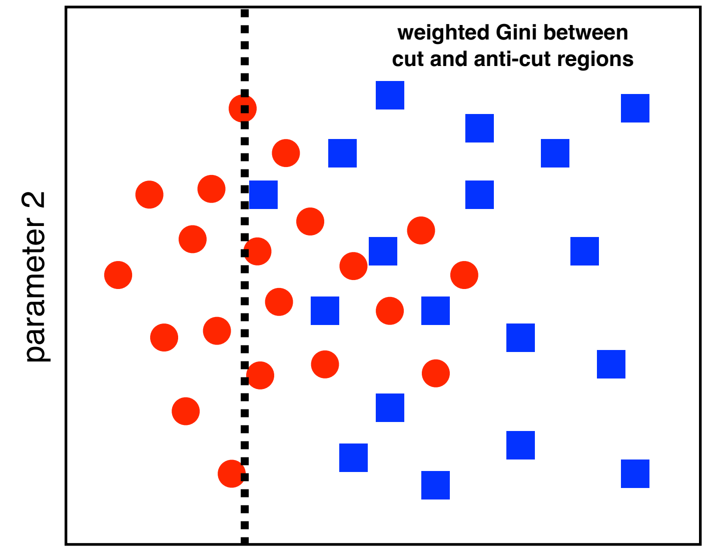
Purity of signal:

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

Purity of background:

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

Where is the best place to cut?



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

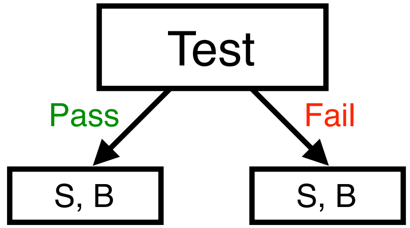
Note: equals zero for p_s or $p_b = 1$ (perfect separation) → Best separation at minimum Gini

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:

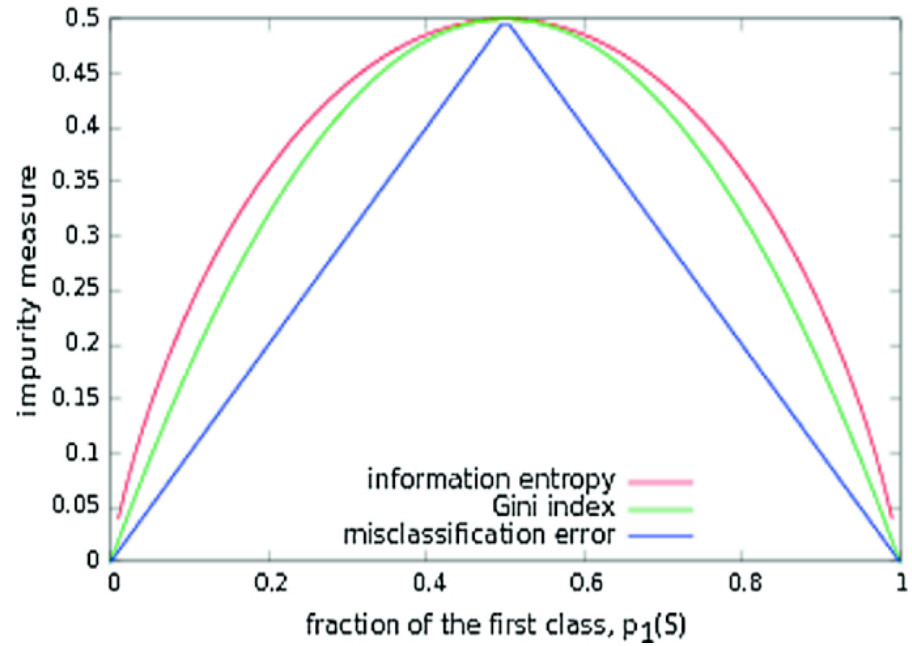
$$I_G(Tot) = f_P I_G(P) + f_F I_G(F)$$

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



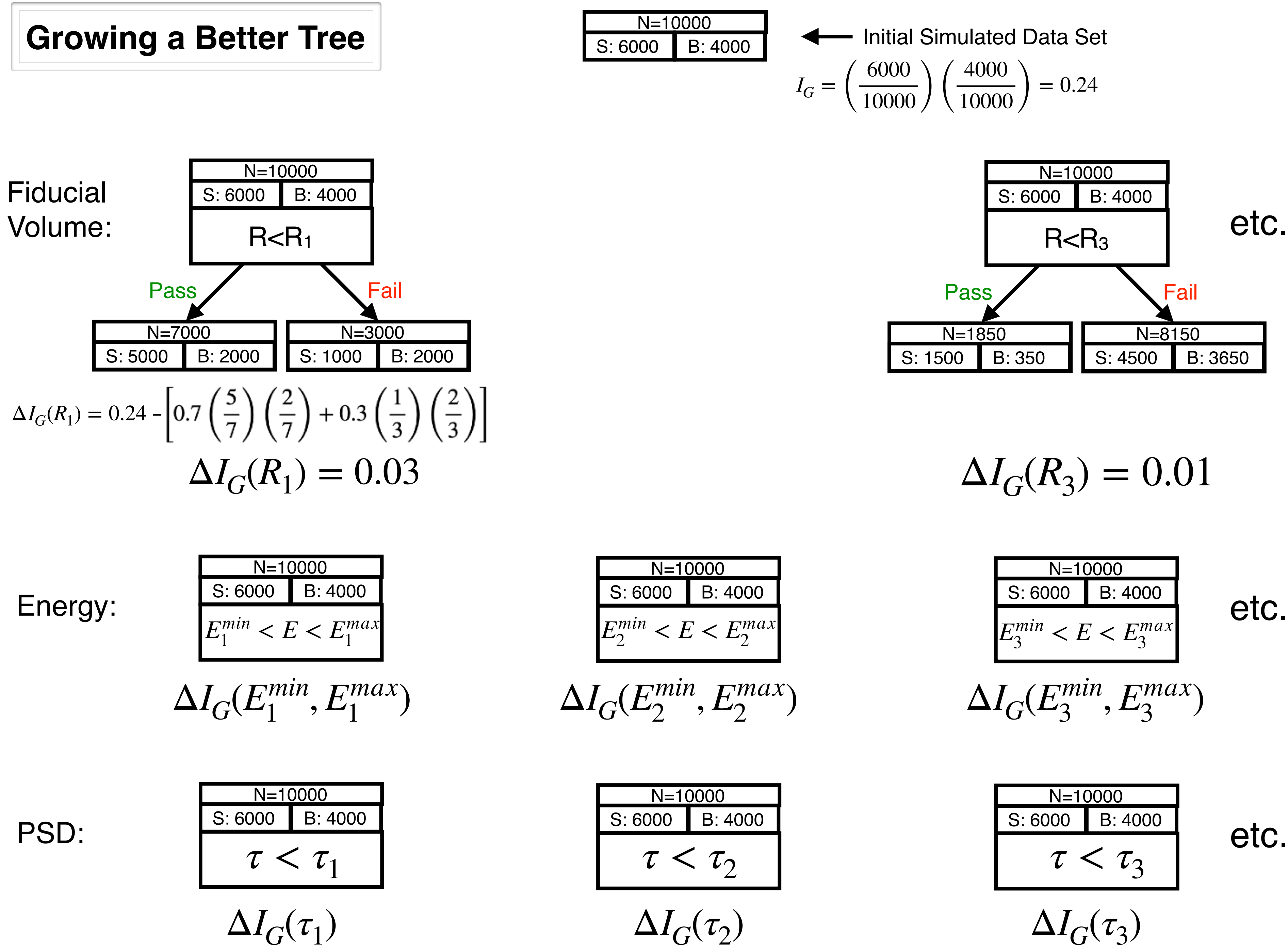
Other Examples of Measures:

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

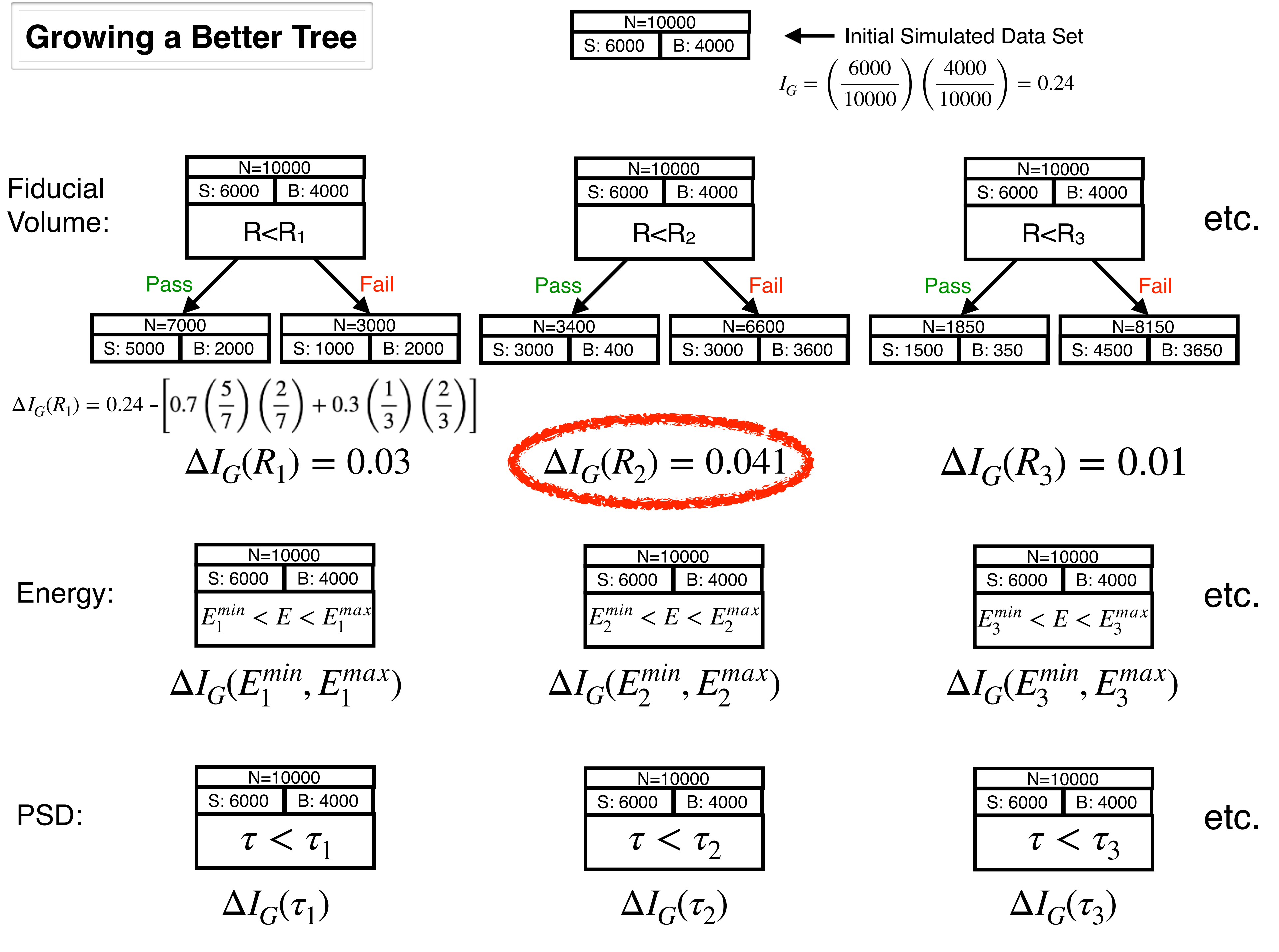
Misclassification Index: $I_M = \sum [1 - max(p_i \, , \, 1 - p_i)]$

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

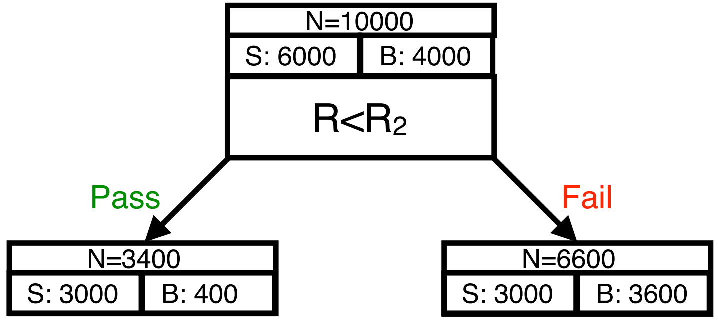
Growing a Better Tree



Growing a Better Tree



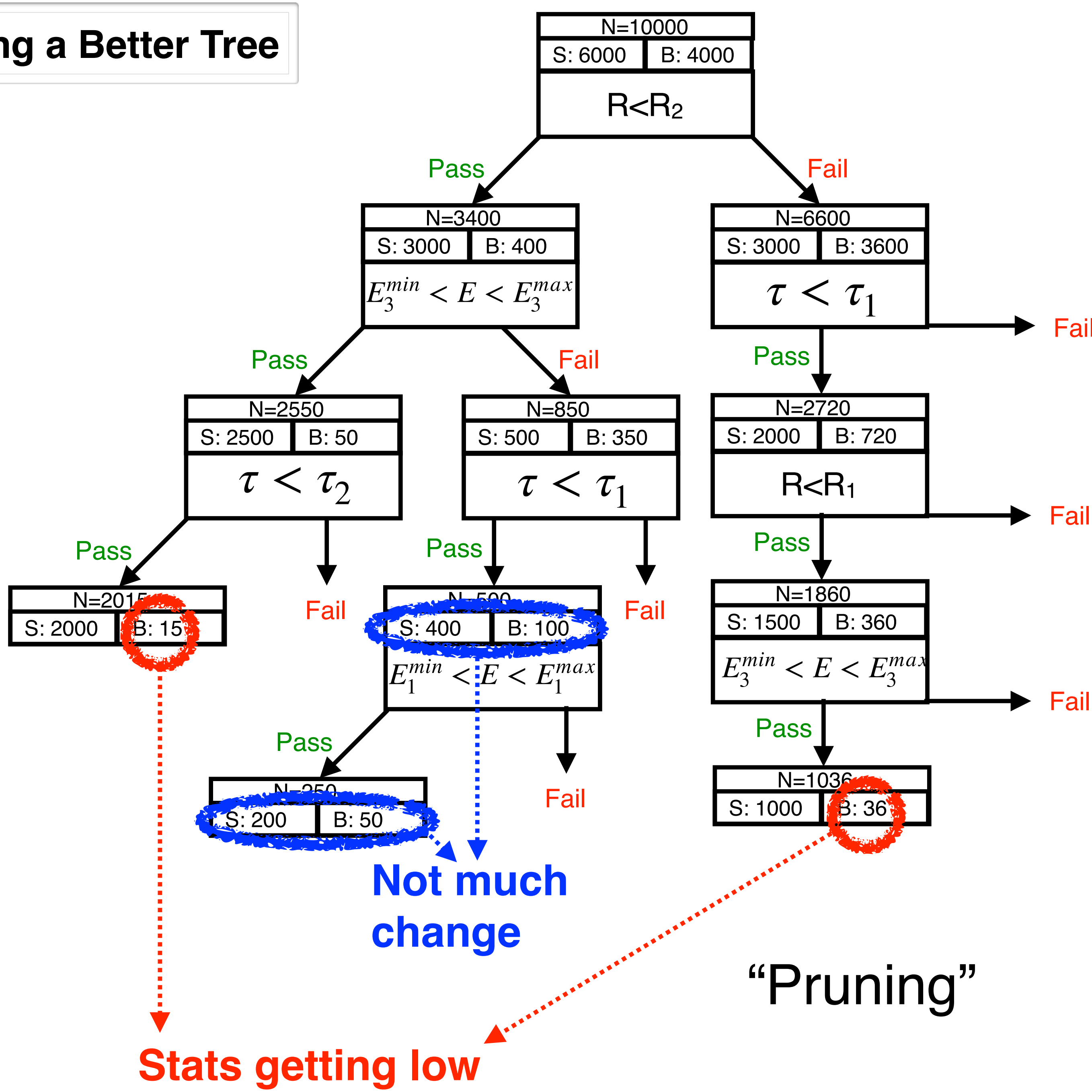
Growing a Better Tree



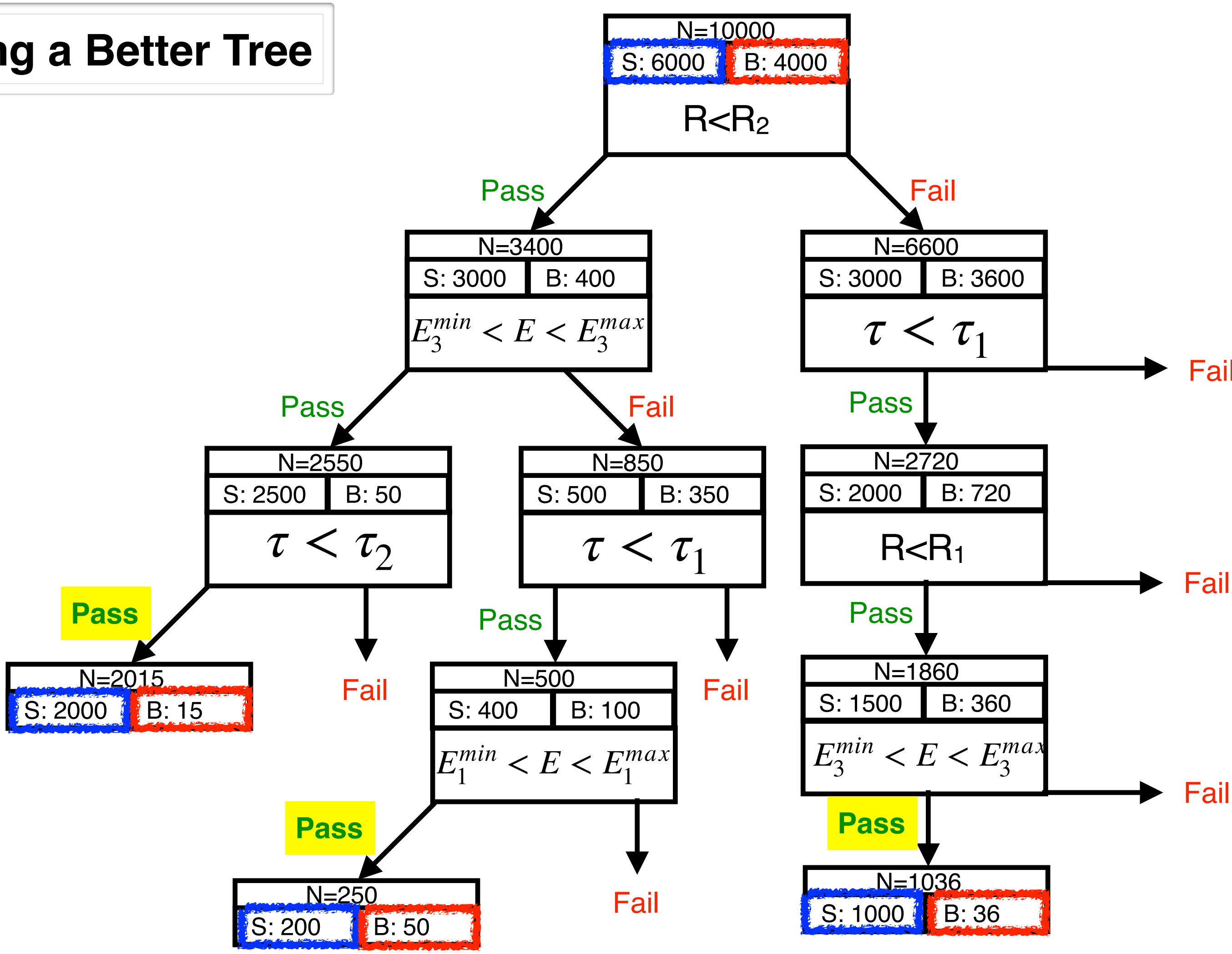
$$\Delta I_G(R_1)$$
$$\Delta I_G(R_2)$$
$$\Delta I_G(R_3)$$
$$\vdots$$
$$\Delta I_G(E_1^{min}, E_1^{max})$$
$$\Delta I_G(E_2^{min}, E_2^{max})$$
$$\Delta I_G(E_3^{min}, E_3^{max})$$
$$\vdots$$
$$\Delta I_G(\tau_1)$$
$$\Delta I_G(\tau_2)$$
$$\Delta I_G(\tau_3)$$
$$\vdots$$

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

Growing a Better Tree



Growing a Better Tree



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

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

Bagging and Random Forests

These are methods to improve the robustness of decision trees and can also help quantify uncertainties.

Bootstrap aggregation, or “**bagging**,” consists of producing many pseudo-independent training sets by randomly sampling events from the main set, allowing events to be sampled more than once. While the produced sets are not truly independent, they yield random variations around the main set without bias and can be used to assess the impact of data variations on decision tree training. Thus, each pseudo-set is used to produce a separate decision tree, and the results from data run through each of these trees is averaged.

“**Random Forest**” takes this a step further to break additional correlations by also randomly sampling a subset of the n features available for discrimination in each generated tree. Typically, each “bagged” tree uses $\sqrt{n}$ randomly selected features, though this should generally be tuned to the particular problem. As before, data is run though all generated trees and the results are averaged.

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":

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

* Freund and Robert E. Schapire, Journal of Computer and System Sciences 55, 119139 (1997)

(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 ?

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

$(w_i^1 \equiv 1)$
relative weights

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

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

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

$$\epsilon_m \equiv \frac{\sum_{q_i \neq \delta_m(x_i)} w_i^m}{\sum_i^N w_i^m}$$

weighted fractional error rate

Generalisation to Multiple Classes: SAMME*

(**S**tage-wise **A**dditive **M**odelling using a **M**ulti-class **E**xponential loss function)

Assume we have K classes, and will more generally ascribe a negative sign to the loss function exponent (*i.e.* lowering the loss) if the correct class is identified, and a positive sign (increasing the loss) if it is not.

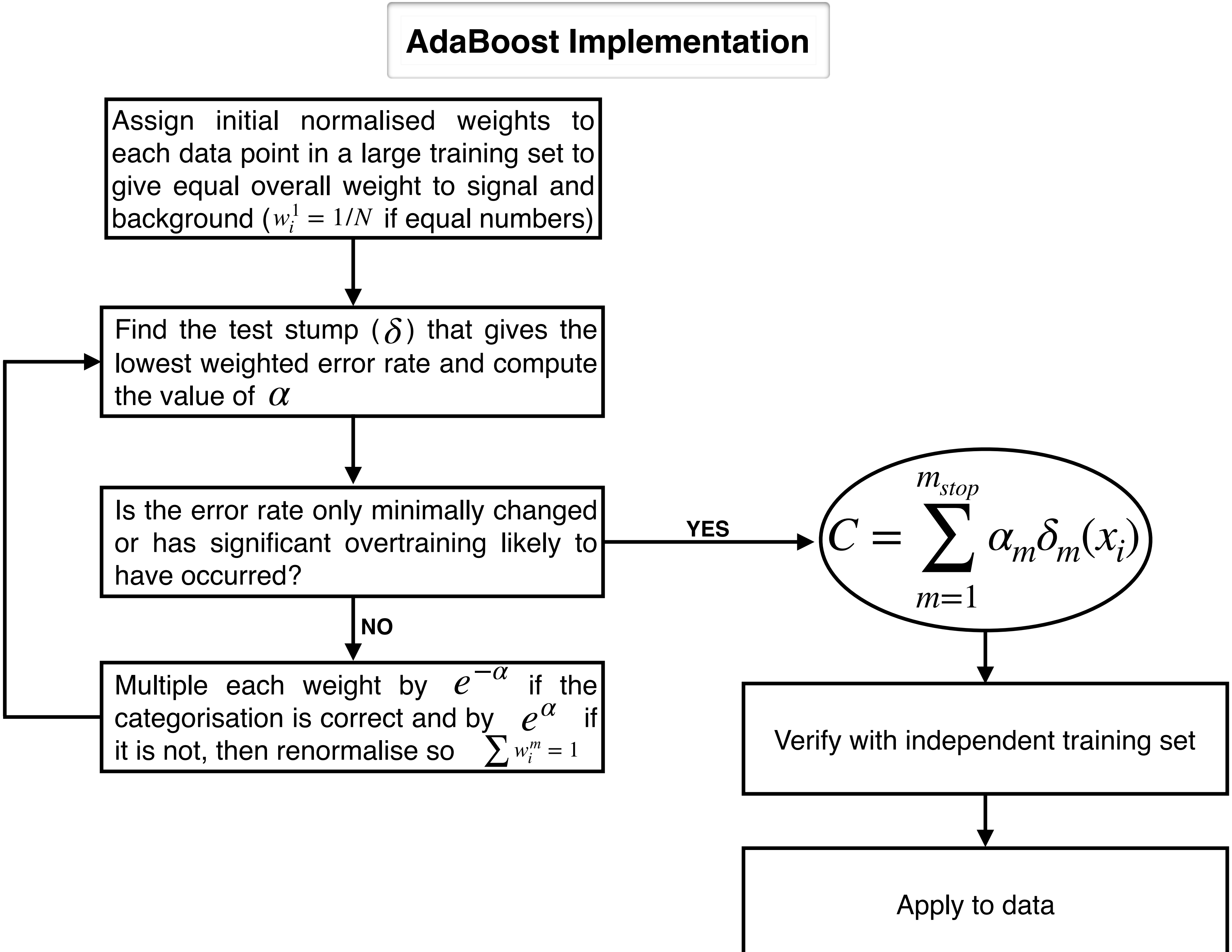
The fraction of correct identifications from random guessing would be 1/K and the fraction of incorrect random identifications would then be 1-1/K. So we modify the error function so that the contributions from each term are weighted relative to the random error rates:

$$E = \left(\frac{1}{1/K}\right) \sum_{correct} w_i^m \, e^{-\alpha_m} + \left(\frac{1}{1-1/K}\right) \sum_{incorrect} w_i^m \, e^{\alpha_m}$$

Carrying through, the value of α_m that minimises the error then becomes:

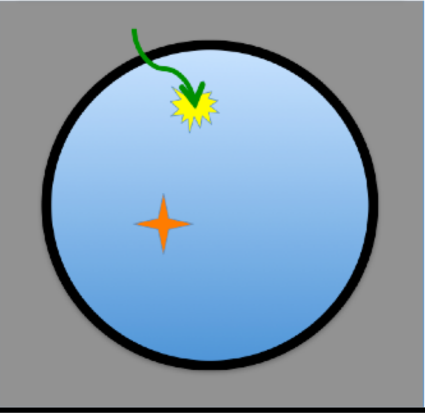
$$\alpha_m = \frac{1}{2}\left[\ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right) + \ln(K-1)\right]$$

*J. Zhu, H. Zou, S. Rosset, T. Hastie, “Multi-class adaboost.” Statistics and its Interface 2.3 (2009): 349-360.



AdaBoost Implementation
Assign initial normalised weights to each data point in a large training set to give equal overall weight to signal and background ($w_i^1 = 1/N$ if equal numbers)
Find the test stump (δ) that gives the lowest weighted error rate and compute the value of α
Is the error rate only minimally changed or has significant overtraining likely to have occurred?
YES
$C = \sum_{m=1}^{m_{stop}} \alpha_m \delta_m(x_i)$
NO
Multiple each weight by $e^{-\alpha}$ if the categorisation is correct and by e^{α} if it is not, then renormalise so $\sum w_i^m = 1$
Verify with independent training set
Apply to data

Let's Play



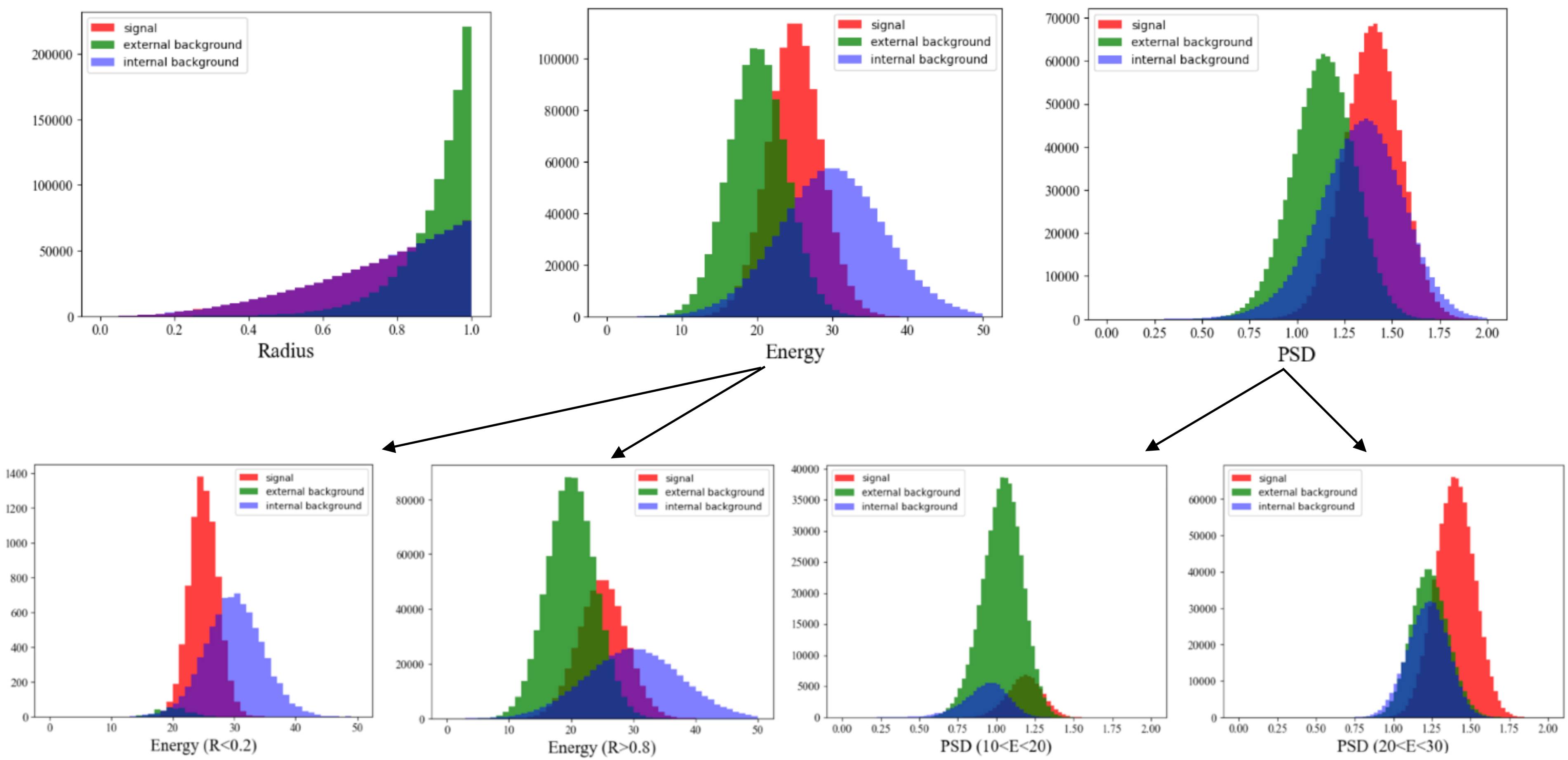
Model Data Set

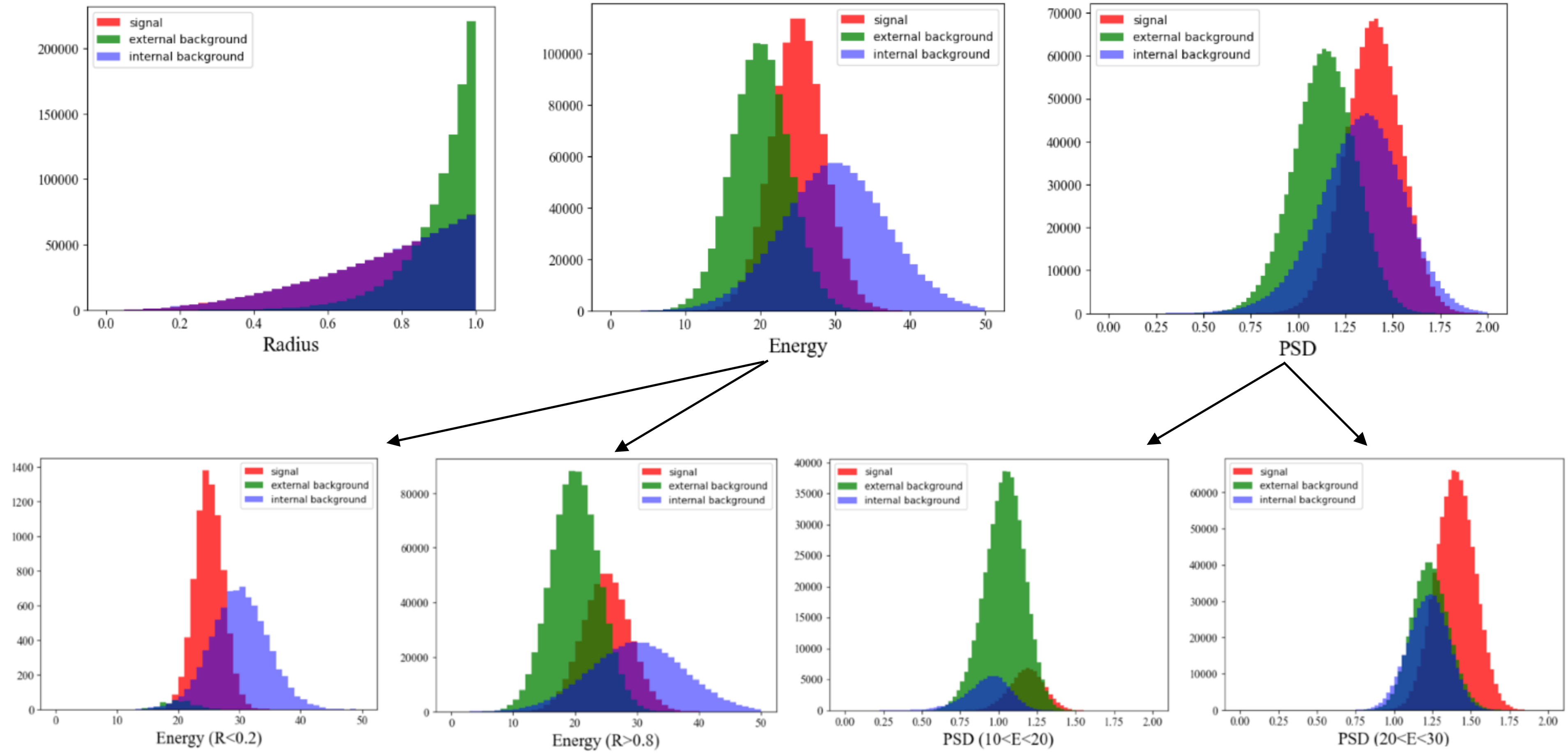
Signal
External background
Internal background

Position (radius)
Energy
Pulse Shape Discrimination

3 categories of events, each with 3 features

- Signal and internal backgrounds are uniform in detector volume [~R³]
- External background falls exponentially from detector edge [~ exp((1-R)/0.1)]
- Energy resolution is twice as bad at the detector edge compared to the centre
- Pulse Shape Discrimination values scale linearly with sqrt of apparent energy





Data frame “Training_Set”

	class	radius	energy	PSD
signal	0	0.576860	21.561235	1.409303
external bkd	1	0.765836	15.071299	0.931453
internal bkd	2	0.611193	27.660483	1.089617
	.			
	.			

examples of entries
(one entry per event)

```
### SET UP AdaBoost ###
from sklearn.ensemble import AdaBoostClassifier
from sklearn import metrics

# Create AdaBoost classifer object (max of 50 weak learners, set initial weights to 1)
bdt = AdaBoostClassifier(n_estimators=50, learning_rate=1)
```

```
### TRAINING ###
print('Training Model...')
X = Train.drop('class',axis=1) # Load list of parameters
y = Train['class']             # Load corresponding classes
y=y.values

model = bdt.fit(X, y)   # Implement multi-class AdaBoost
ym = model.predict(X)   # Resulting model predictions
```

(multiple test sets used to measure average and variance of results)

```
### MODEL TESTING ###
for i in range(NSets):                      # Loop over test data sets stored as data frame list
    X = Test[i].drop('class',axis=1)        # Load list of parameters
    y = Test[i]['class']                    # Load corresponding classes
    y=y.values
    ym = model.predict(X)                   # Model prediction of class
    yp=model.predict_proba(X)               # Class 'probabilities' based on training set
```

Train on a mix of 10000 each of signal, external and internal backgrounds

(2 orders of magnitude larger than individual test sets)

```
Training Accuracy: 0.746
```

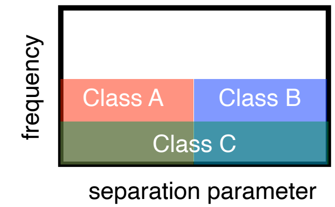
Truth: Sig=Ext=Int=100

```
Summary from 100 Independent Test Sets
-------------------------------------
Overall Test Accuracy: 0.74
Signal Accuracy: 0.7762
External Accuracy: 0.7686
Internal Accuracy: 0.6881

# Signal Selected: 112.4 ± 0.77
# Externals Selected: 104.8 ± 0.72
# Intervals Selected: 82.77 ± 0.68

Total Signal Predicted: 102.0 ± 0.038
Total External Predicted: 95.41 ± 0.067
Total Interval Predicted: 102.6 ± 0.042
```

Some indication of over-training

Efficiencies to select classes in isolation. In general, these are not necessarily the same for each class.

Selection of individual classes can be further biased by the presence of other classes



Probabilities have “baked in” systematic biases that depend on both training sample statistics and algorithm details (*e.g.* number of weak learners etc.)

Uncertainties not reflected by sample-to-sample variances!

Truth: Sig=50, Ext=100, Int=200

```
Summary from 100 Independent Test Sets
-------------------------------------
Overall Test Accuracy: 0.729
Signal Accuracy: 0.7788
External Accuracy: 0.7738
Internal Accuracy: 0.6866

# Signal Selected: 90.94 ± 0.66
# Externals Selected: 111.8 ± 0.65
# Intervals Selected: 147.2 ± 0.72

Total Signal Predicted: 119.8 ± 0.055
Total External Predicted: 107.0 ± 0.1
Total Interval Predicted: 123.2 ± 0.057
```

Subtle change belies big impact!

Essentially unchanged because these are assessed in isolation

Disastrous!

Performance and accuracy depends on the class composition being exactly the same in data as it is in the training set. Otherwise the trained BDT classifier is no longer optimal and results are untrustworthy!

How, then, do you implement things so as to correctly characterise the statistical behaviour, make the performance robust and insure an accurate interpretation of results?

Put a pin in that…
we'll come back to all this again after the next lecture!

(Spoiler: previous discussions of likelihood will not have been wasted!)

Some Other 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 not unique. There is no theorem that says which set of these is the best or produces the most efficient algorithm for a given problem.
- Decision trees can be overly sensitive to noise
- 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** can make things slow and 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 directly get the likelihood and all the benefits that brings.*** But you can always make PDFs of decision tree outputs for different event classes and derive likelihoods and confidence/credibility intervals in the usual way!
- 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).