PHYS 139/239: Machine Learning in Physics

Lecture 4:
(Boosted) Decision Trees

Javier Duarte — January 19, 2023

Updated supervised learning pipeline

- Training dataset: $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ where $x \in \mathbb{R}^D$ and $y \in \mathbb{R}$

  For regression

- Model / hypothesis class: $f(x \mid w) = w^\intercal x$ (linear models)

  or $\phi(x)$ instead of x

- Loss function: $L(y, y') = (y - y')^2$ (squared loss)
- Optimization algorithm: SGD with regularization (L^1 or L^2)
- Cross validation and model selection:

- Testing and deployment

Select λ

Recap: Probabilistic approach

Parametrized by w

- Idea: Model a probability distribution $p(y \mid x; w)$ of labels y given inputs x
- Choose a form for $p(y \mid x; w)$ (different for regression and classification)
- Write the likelihood of w, i.e. the probability of observing the labels y_i of the training dataset S given the inputs x_i:

$$p(S \mid w) = \prod_{i=1}^{N} p(y_i \mid x_i)$$

Assuming training examples are independent

- Maximum likelihood estimation (MLE): find w that maximizes the (log) likelihood:

$$\log p(S \mid w) = \sum_{i=1}^{N} \log p(y_i \mid x_i; w) = -\, l(w)$$

Equivalently, minimize the loss function!

Binary classification revisited

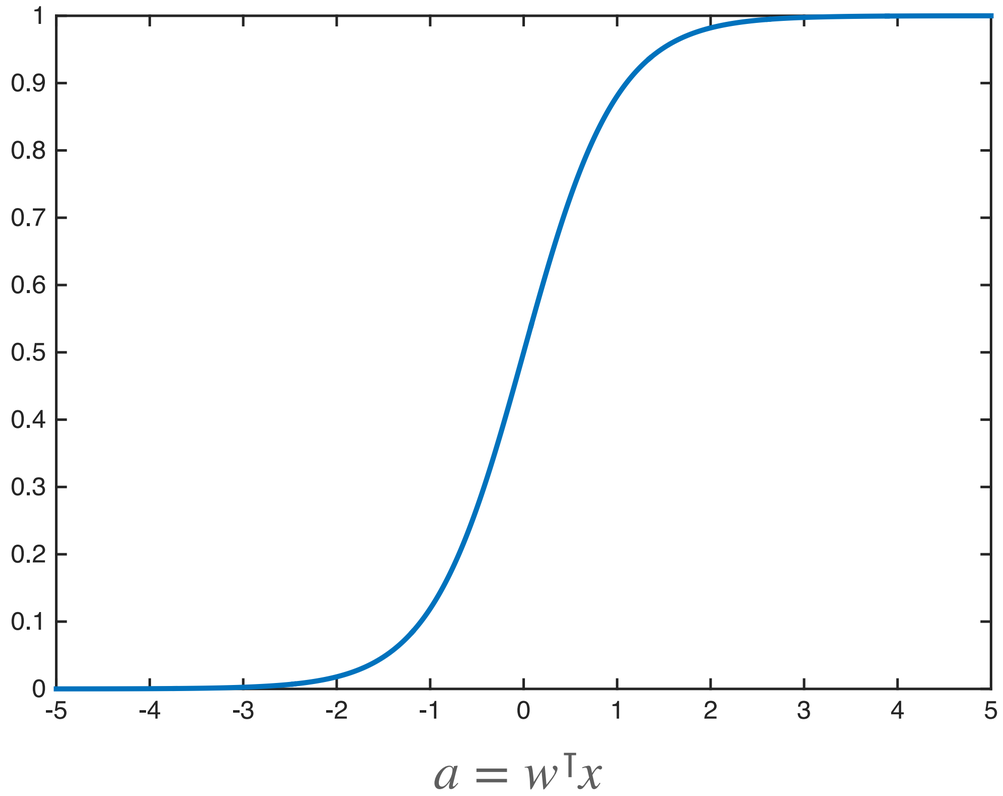
- Linear model for binary classification: $f(x \mid w) = \text{sign}(\underbrace{w^\intercal x}_{\text{Raw score}}) \in \{\overset{+}{+1}, \overset{\triangle}{-1}\}$

- Idea: raw score to model the probability of each class

$$\sigma(w^\intercal x) \approx \text{probability that } y = +1$$

Logistic/sigmoid function $\sigma : \mathbb{R} \to (0,1)$

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$



What is the right loss function?

- Assume that the true probability that $y = +1$ given x is $\sigma(\bar{w}^\intercal x)$ and that $p(y \mid \sigma(\bar{w}^\intercal x))$ is a Bernoulli distribution

True value of w

- Likelihood of w:

Only one of these two terms appears depending on y_i

$$p(S \mid w) = \prod_{i=1}^{N} \sigma(w^\intercal x_i)^{\delta_{\{y_i=+1\}}} (1 - \sigma(w^\intercal x_i))^{\delta_{\{y_i=-1\}}}$$

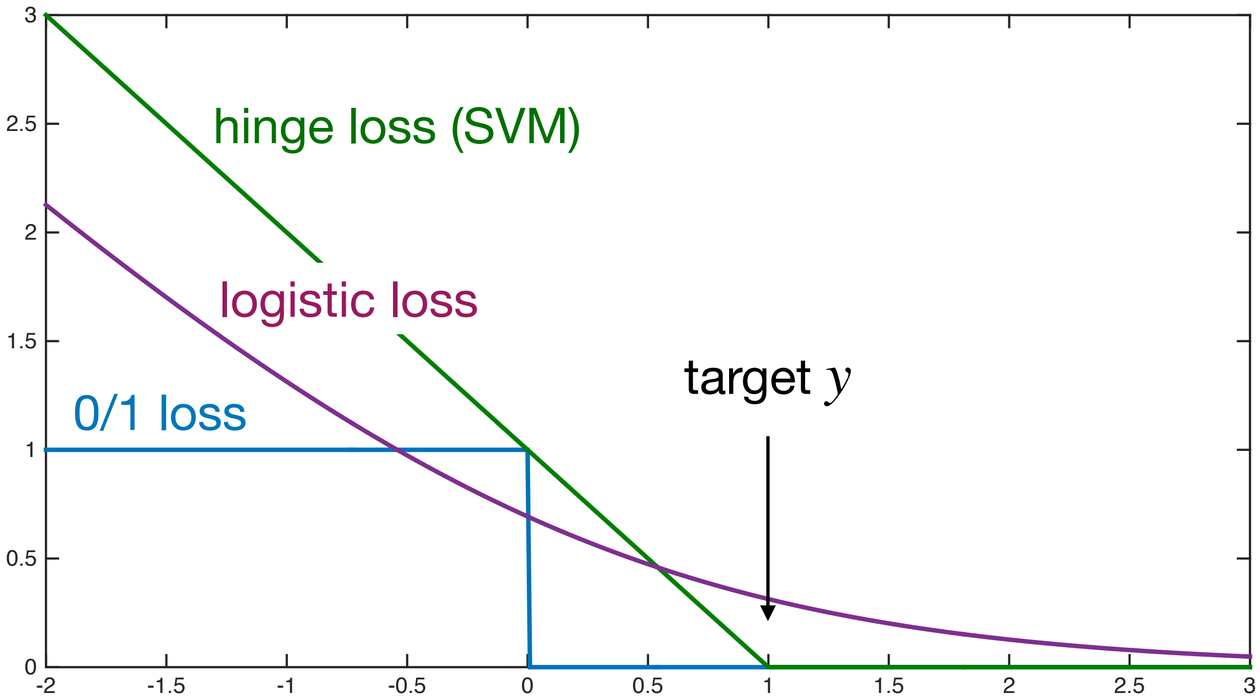
- Negative log likelihood of w a.k.a. logistic / log / binary cross-entropy loss:

$$-\log p(S \mid w) = -\sum_{i=1}^{N} \delta_{\{y_i=+1\}} \log \sigma(w^\intercal x_i) + \delta_{\{y_i=-1\}} \log(1 - \sigma(w^\intercal x_i))$$

Logistic loss

- Logistic / log / binary cross-entropy loss:

$$L(y, y') = -\,\delta_{\{y=+1\}} \log y' - \delta_{\{y=-1\}} \log(1 - y')$$



Logistic regression update

- Logistic loss: $-\left(\delta_{\{y_i=+1\}} \log \sigma(w^\intercal x_i) + \delta_{\{y_i=-1\}} \log(1-\sigma(w^\intercal x_i))\right)$

- Gradient: $-\left(\delta_{\{y_i=+1\}} - \sigma(w^\intercal x_i)\right) x_i$

  Using: $\sigma'(a) = \sigma(a)(1-\sigma(a))$

  1 if $y_i = +1$
  0 otherwise

  Model's prediction

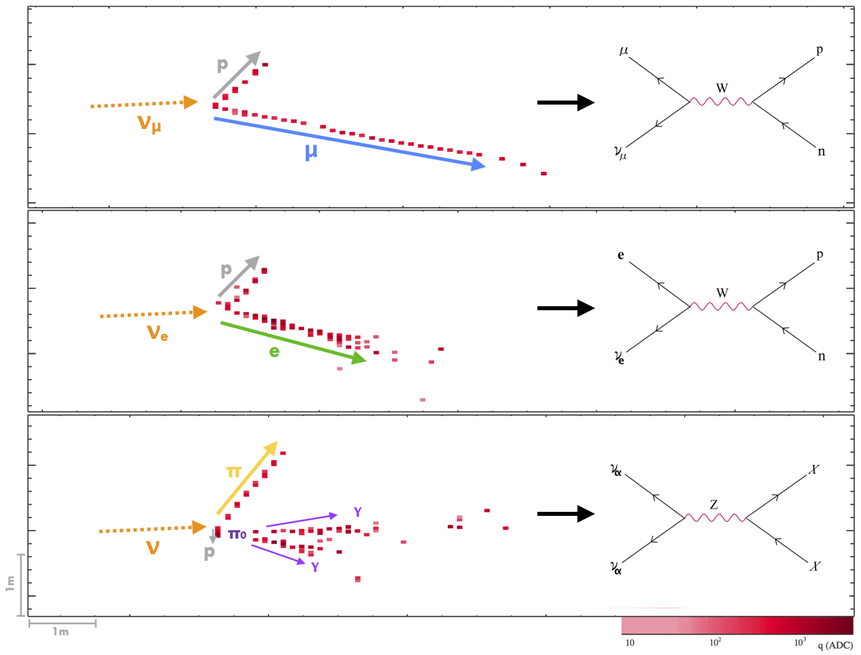
- SGD update: $w(t+1) = w(t) + \eta\left(\delta_{\{y=+1\}} - \sigma(w^\intercal x)\right) x$ for $(x, y) \in S$
  (logistic regression)

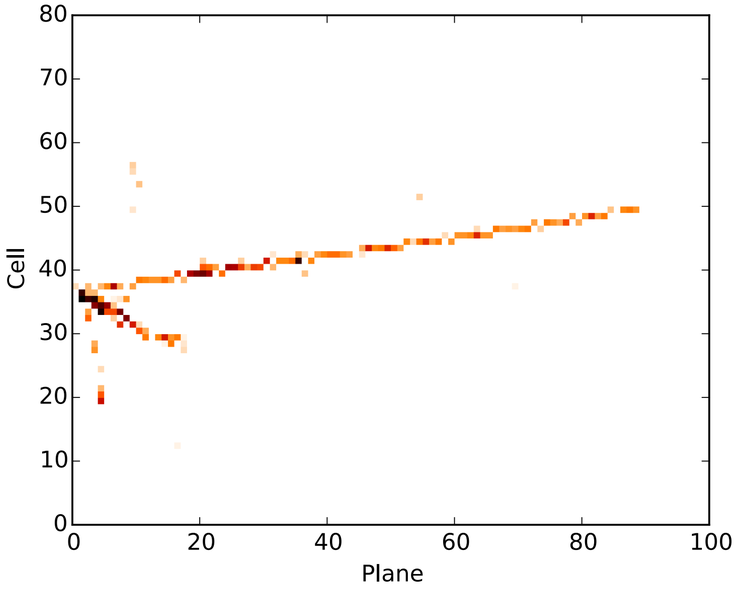
- SGD update: $w(t+1) = w(t) + 2\eta(y - w^\intercal x)x$ for $(x, y) \in S$
  (linear regression)

Multiclass logistic regression

arXiv:1604.01444

- Predict a raw score for each of K classes
- Example: $K = 3, Y = \{\nu_\mu \text{ CC}, \nu_e \text{ CC}, \text{NC}\}$





$x \in \mathbb{R}^D$

$$w^\intercal x = \begin{bmatrix} w_1^\intercal x \\ w_2^\intercal x \\ w_3^\intercal x \end{bmatrix} \begin{array}{l} \longleftarrow \nu_\mu \text{ CC score} \\ \longleftarrow \nu_e \text{ CC score} \\ \longleftarrow \text{NC score} \end{array}$$

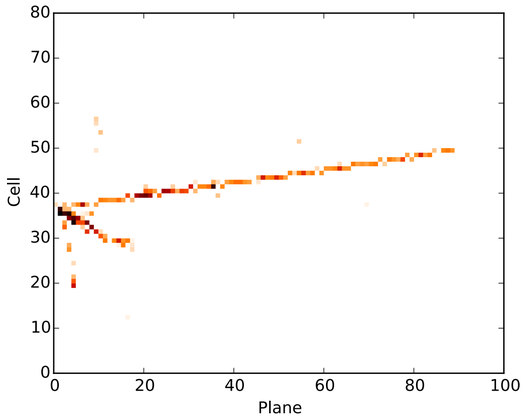
Model parameters: $w \in \mathbb{R}^{K \times D}$

Multiclass logistic regression

- Sigmoid is replaced by softmax:

$$\text{softmax}\left(\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_K \end{bmatrix}\right) = \frac{1}{\sum_{k=1}^{K} \exp(a_k)} \begin{bmatrix} \exp(a_1) \\ \exp(a_2) \\ \vdots \\ \exp(a_K) \end{bmatrix}$$

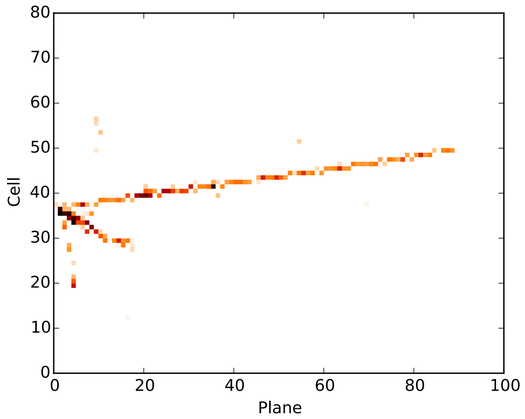
K numbers between 0 and 1 that sum to 1



$$\longrightarrow \quad w^\intercal x = \begin{bmatrix} w_1^\intercal x \\ w_2^\intercal x \\ w_3^\intercal x \end{bmatrix} \quad \longrightarrow \quad \text{softmax}(w^\intercal x) = \begin{bmatrix} f_1(x \mid w) \\ f_2(x \mid w) \\ f_3(x \mid w) \end{bmatrix}$$

$x \in \mathbb{R}^D$

Multiclass logistic regression example



$x \in \mathbb{R}^D$

$$w^\intercal x = \begin{bmatrix} 3.1 \\ 0.5 \\ -1.2 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} 0.919 \\ 0.068 \\ 0.013 \end{bmatrix}$$

ν_μ CC: 93%

ν_e CC: 7%

NC: 1%

softmax outputs sum to 1

Categorical cross-entropy loss

- Negative log likelihood of w a.k.a. categorical cross-entropy loss:

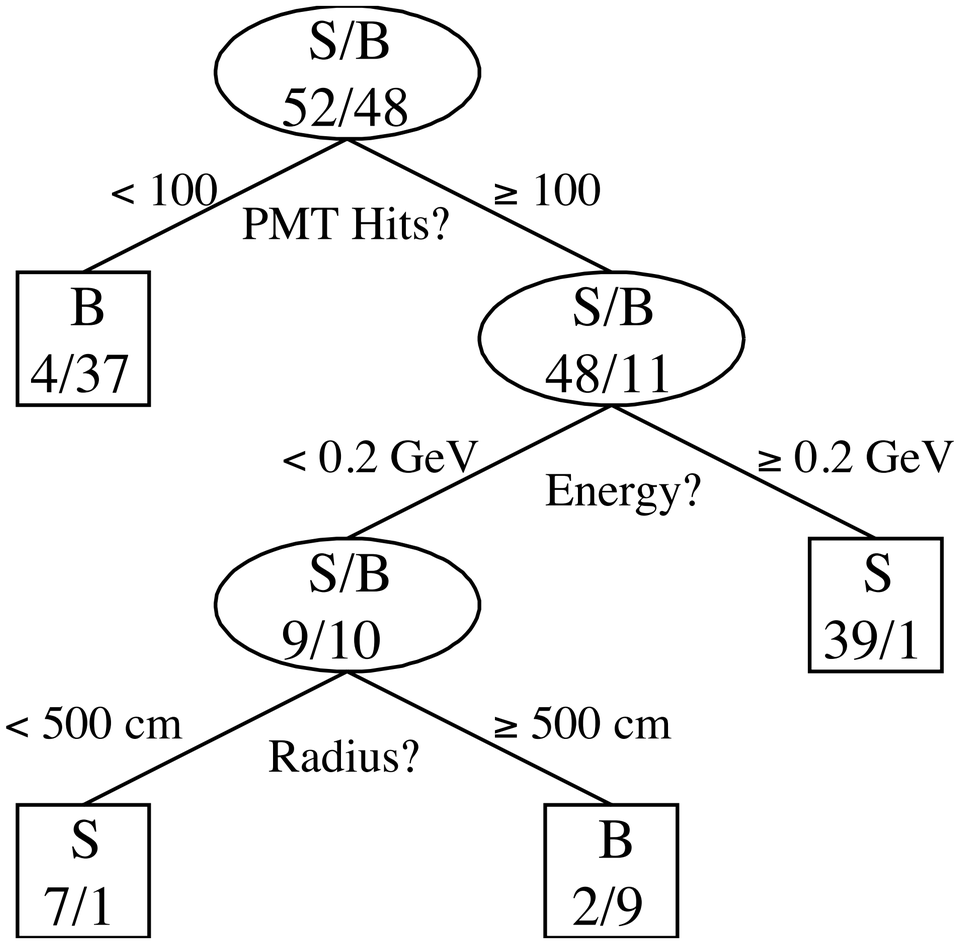
$$-\log p(S \mid w) = -\sum_{i=1}^{N} \sum_{k=1}^{K} \delta_{\{y_i = k\}} \log f_k(x \mid w)$$

- Generalizes the binary cross-entropy loss (and equivalent when $k = 2$)

Recap: Activations and loss functions

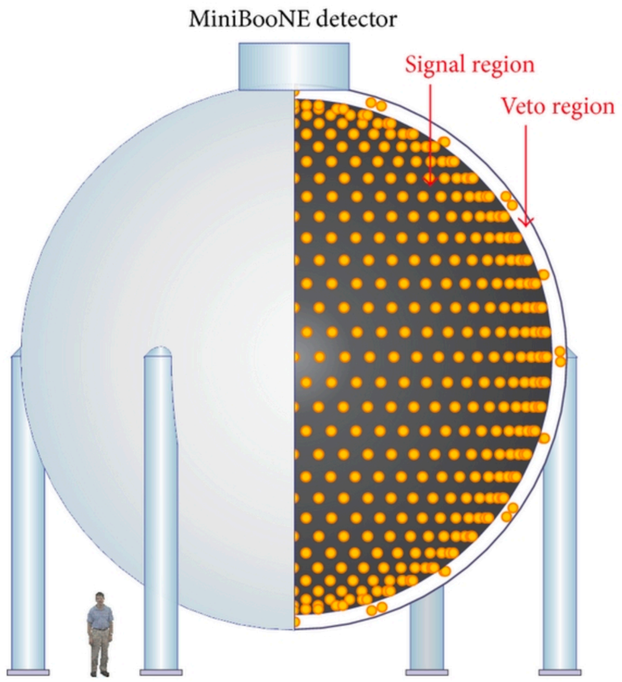
- Linear regression:
  - Activation: linear; loss: mean-squared error
- Binary classification:
  - Activation: linear; loss: perceptron (PLA)
  - Activation: linear; loss: hinge (SVM)
  - Activation: sigmoid; loss: binary cross-entropy
- Multiclass classification:
  - Activation: softmax; loss: categorical cross-entropy

Decision trees



- Leaf nodes classify events as either signal (ν_e) or background (ν_μ)

arXiv:physics/0408124

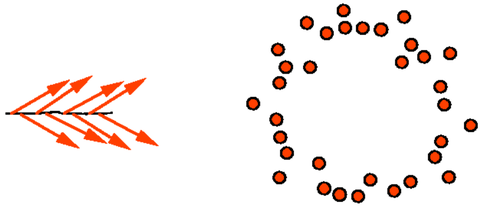


MiniBooNE: 1520 photomultiplier signals

Goal: separate ν_e and ν_μ events

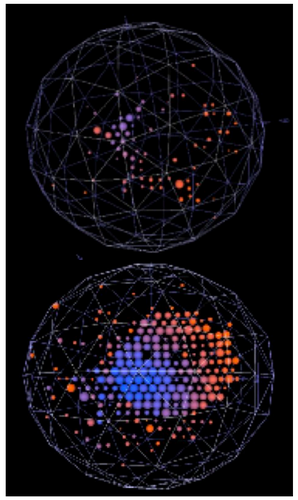
ν_e CCQE

$\nu_e n \rightarrow p e^-$

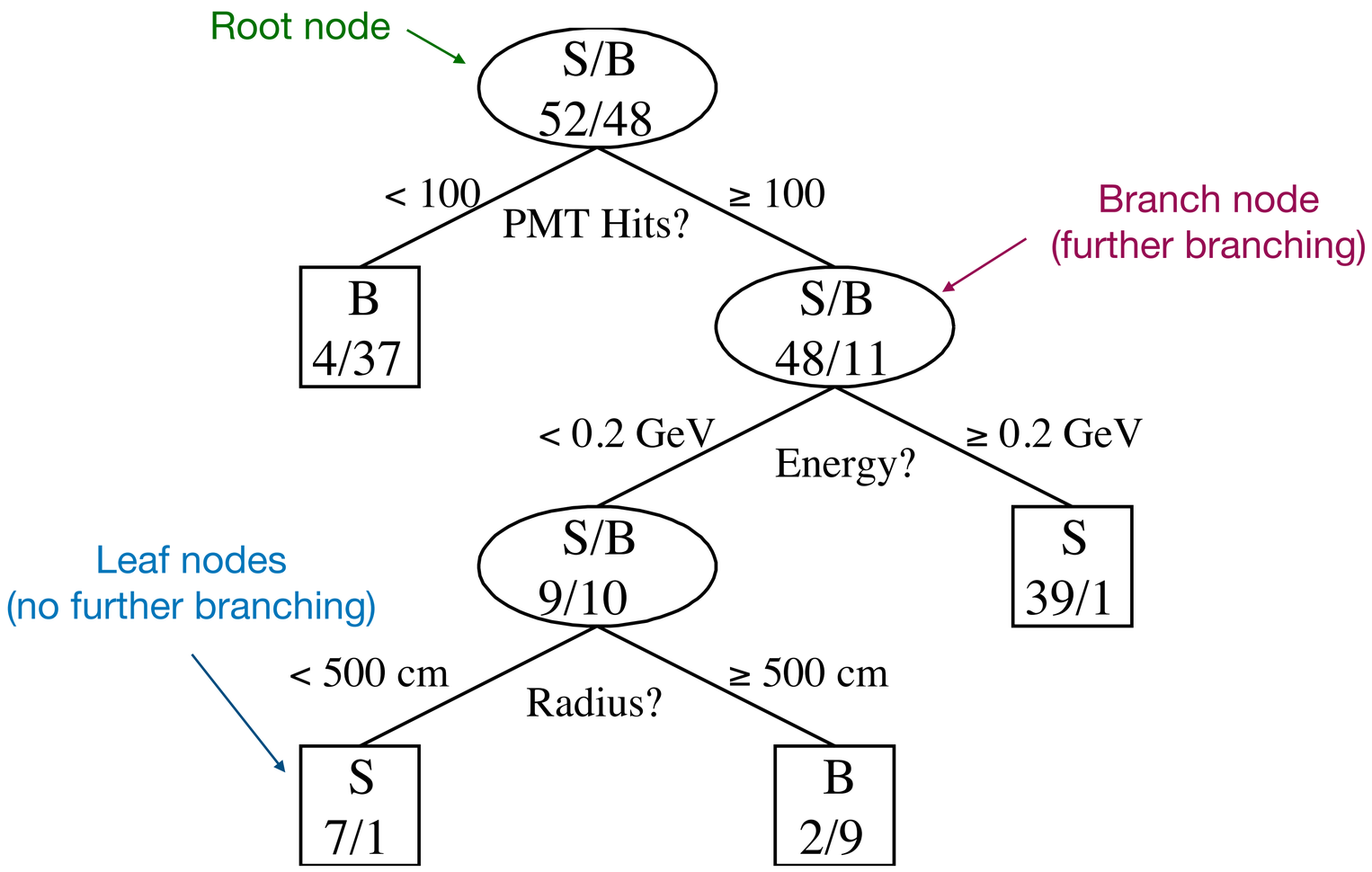


ν_μ CCQE

$\nu_\mu n \rightarrow p \mu^-$



Decision trees



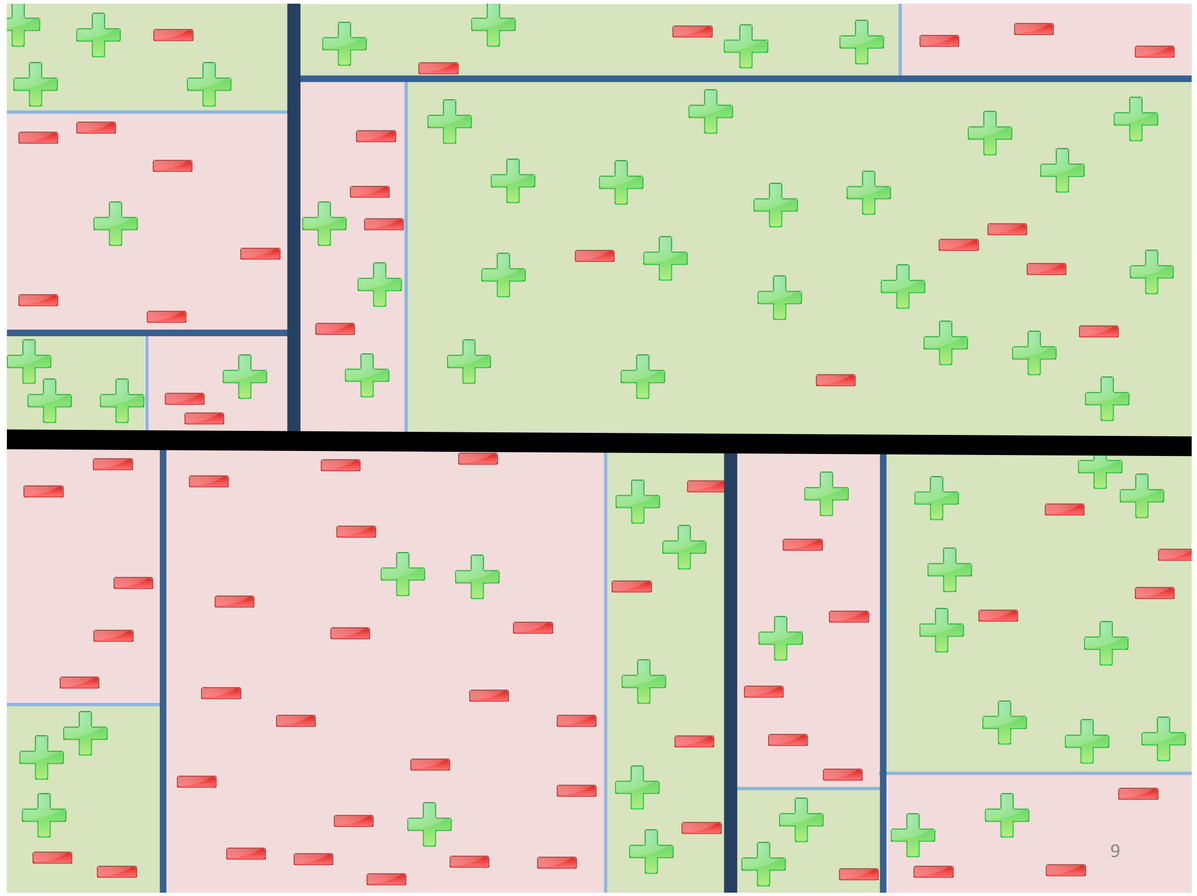
- Every internal/branch node has a binary query function $q(x)$
- Every leaf node has a prediction (0 or 1)
- Prediction starts at root node
  - Recursively calls query function
  - Positive response → left child
  - Negative response → right child
  - Repeat until leaf node

Queries

- Decision tree defined by tree of queries
- Binary query $q(x)$ maps features to 0 or 1
- Basic form is a “cut”: $q(x) = \delta[x^{(d)} > c]$
  - $q(x) = \delta[x^{(3)} > 5]$
  - $q(x) = \delta[x^{(1)} > 0]$
  - $q(x) = \delta[x^{(55)} > 1.2]$
  - …

Decision tree function class

- “Piecewise static” function class
  - All possible partitioning over feature space
  - Each partition has a static prediction
- Partitions are **axis-aligned**
  - i.e. no diagonals!

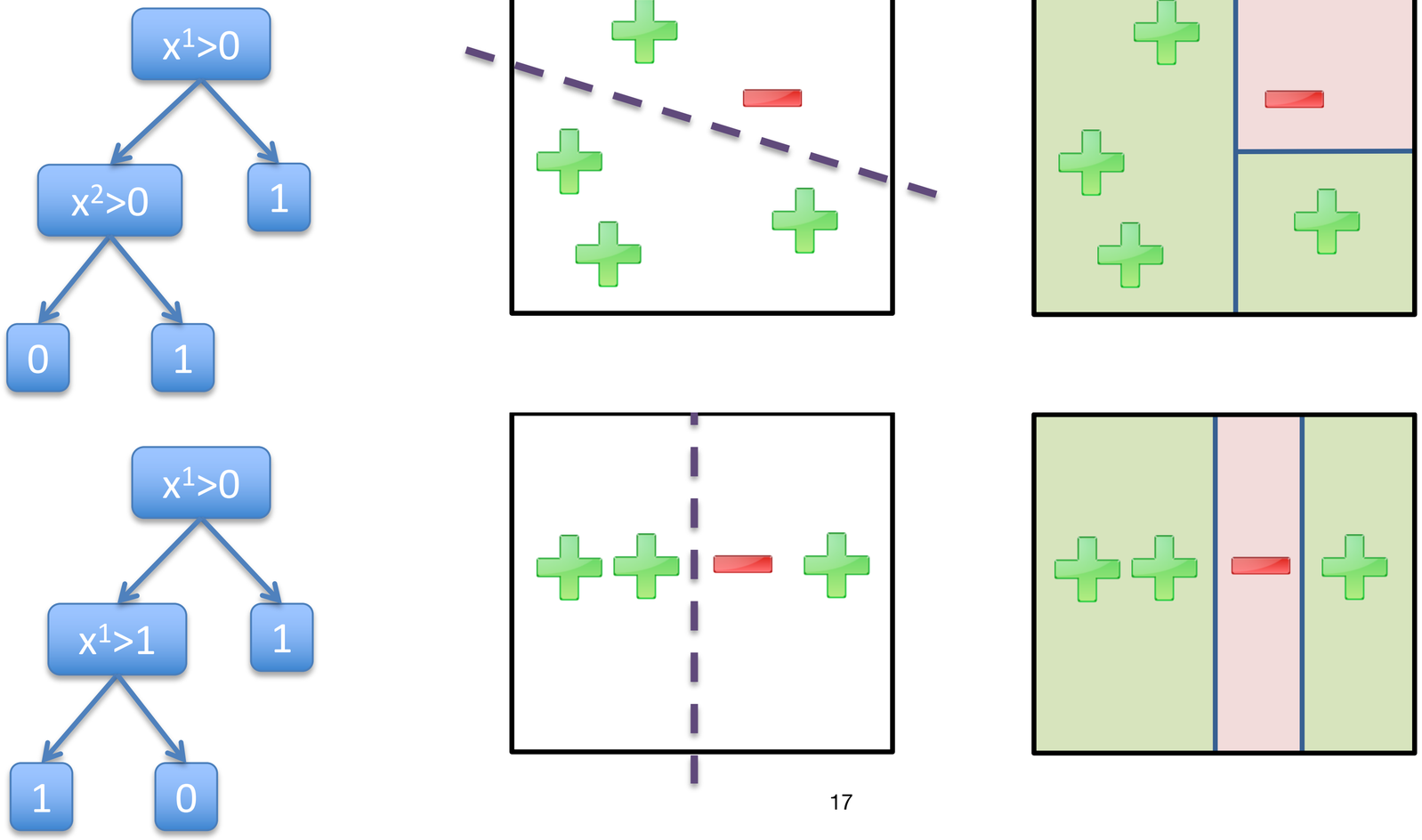


Decision trees vs. linear models

- Decision trees are nonlinear models!
- Examples:

No linear model can achieve 0 error

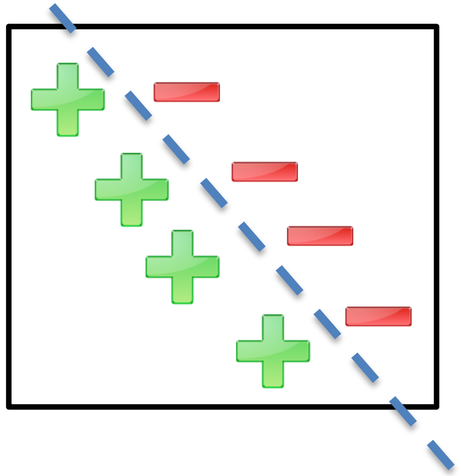
Simple decision tree can achieve 0 error



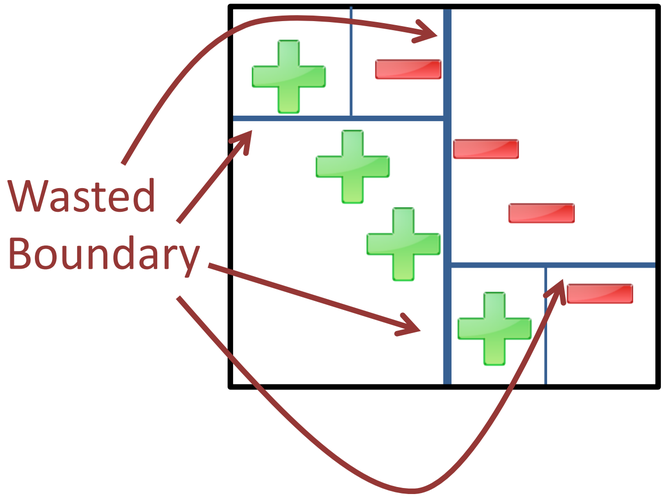
Decision trees vs. linear models

- Decision trees are axis-aligned!
- Example:

Simple linear SVM can easily find max margin



Decision trees require complex axis-aligned partitioning

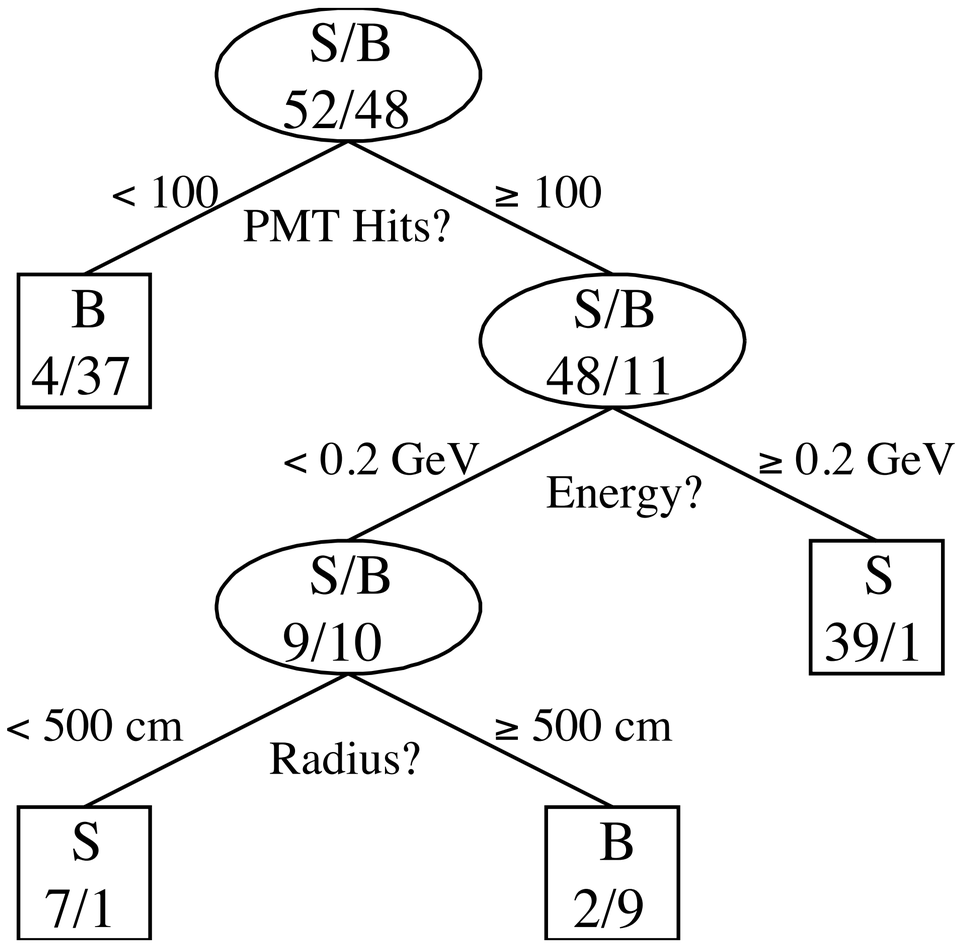


Decision trees vs. linear models

- Decision Trees are often more accurate!
- Non-linearity is often more important
  - Just use many axis-aligned boundaries to approximate diagonal boundaries
  - It’s OK to waste model capacity
- **Catch**: requires sufficient training data

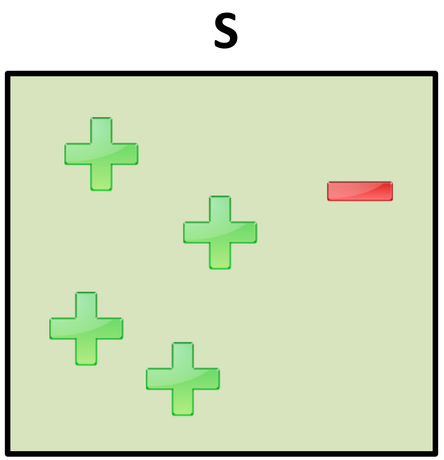
Decision tree training

- Every node = partition/subset of dataset S
- Every layer = complete partitioning of S
- Children = complete partitioning of parent



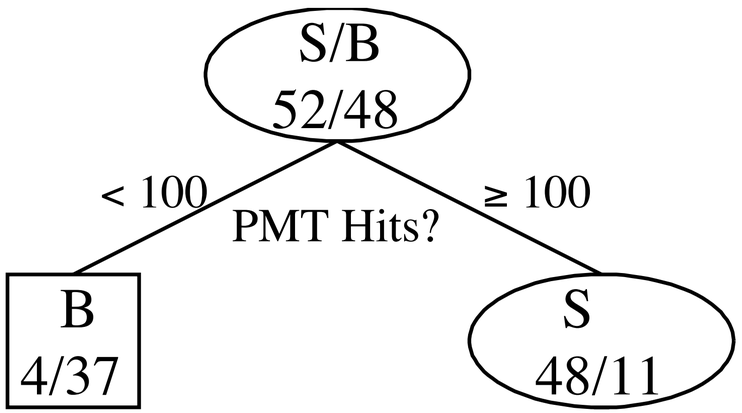
Decision tree training

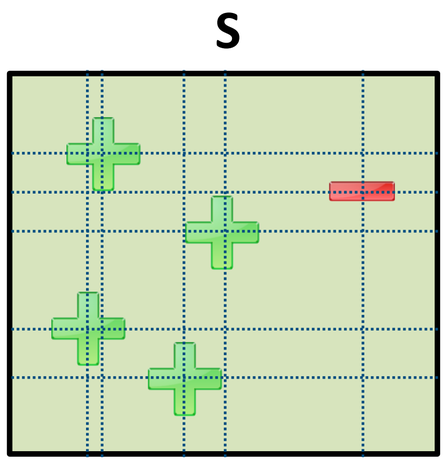
- What if just one node?
  - i.e. just root node
  - No queries
  - Single prediction for all data
- Make a single prediction: majority class in training set (i.e. signal here)



Decision tree training

- What if just two levels?
  - i.e. root node and 2 children
  - Single query (Which one?)
  - How many possible queries?
    - Number of possible queries = Number of possible splits = DN
    - D = Number of features
    - N = Number of training samples
  - How do we choose the “best” query?



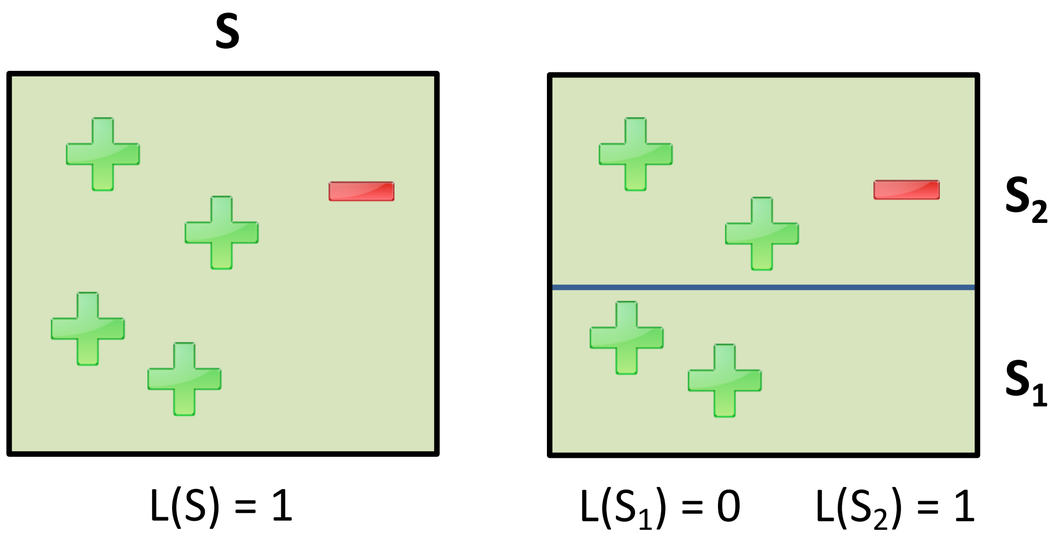


Impurity

- Define impurity function, e.g. 0/1 loss:

$$L(S') = \min_{f(x)\in\{0,1\}} \sum_{(x,y)\in S} \delta[f(x) \neq y]$$

Classification error of best single prediction in each partition



Impurity Reduction = 0

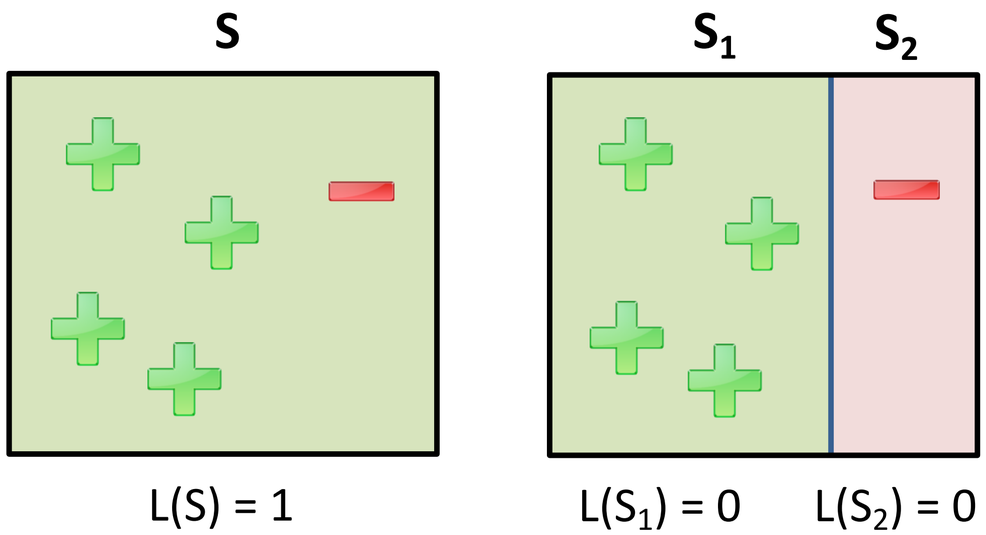
No Benefit From This Split!

Impurity

- Define impurity function, e.g. 0/1 loss:

$$L(S') = \min_{f(x)\in\{0,1\}} \sum_{(x,y)\in S} \delta[f(x) \neq y]$$

Classification error of best single prediction in each partition



Impurity Reduction = 1

Choose Split with largest impurity reduction!

Impurity as loss function

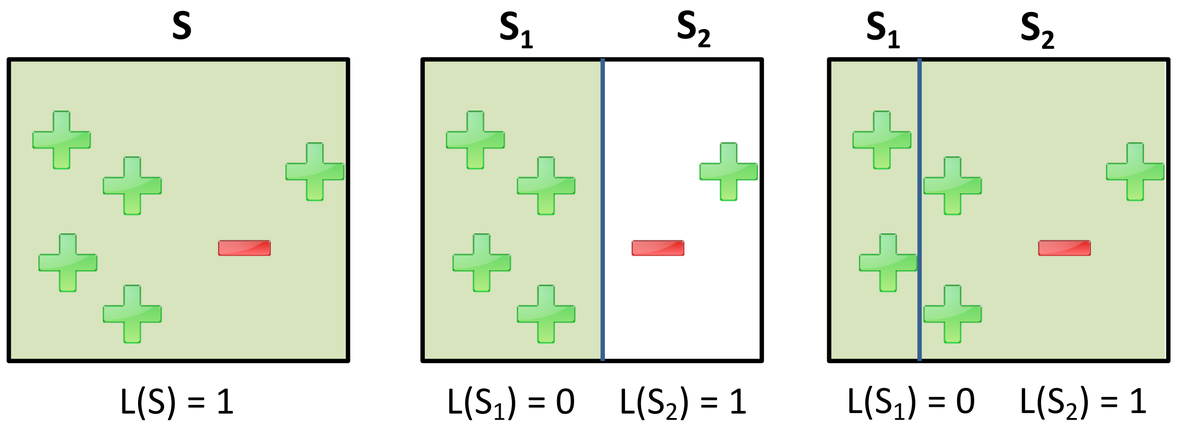
- Training goal: find decision tree with low impurity
- Impurity over leaf nodes = training loss

$$L(S) = \sum_{S' \subset S} L(S') \text{ where } L(S') = \min_{f(x) \in \{0,1\}} \sum_{(x,y) \in S} \delta[f(x) \neq y]$$

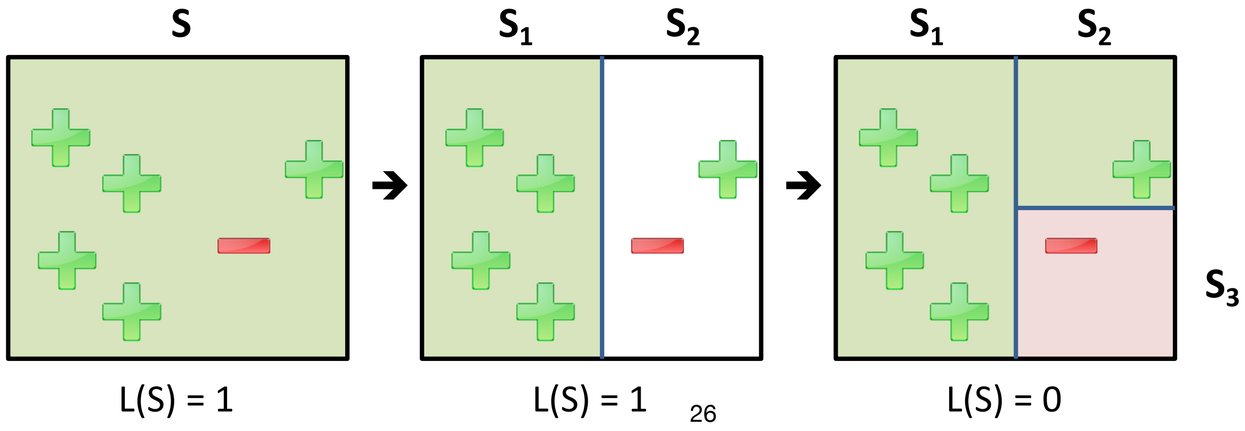
Problems with 0/1 loss

- 0/1 loss is discontinuous

- Degeneracies: all partitioning give same impurity reduction



- A good partitioning choice may not improve 0/1 loss, e.g., leads to an accurate model with a subsequent split

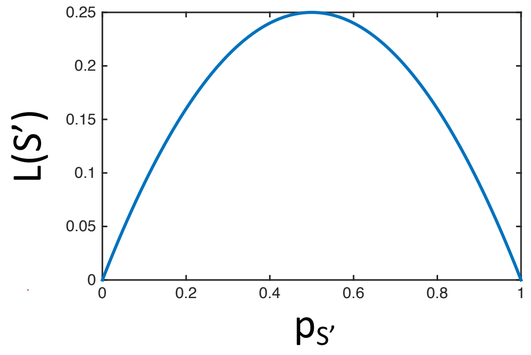


Continuous impurity measures

- Bernoulli variance:

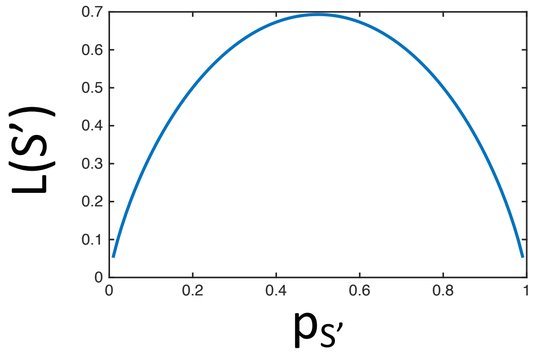
$$L(S') = |S'| p_{S'}(1 - p_{S'})$$

where $p_{S'}$ = fraction of S' that are $y = +1$



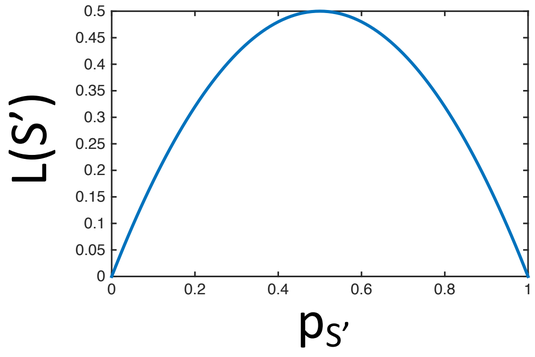
- Entropy / information gain:

$$L(S') = -|S'| \left(p_{S'} \log p_{S'} + (1 - p_{S'}) \log(1 - p_{S'}) \right)$$



- Gini index / impurity:

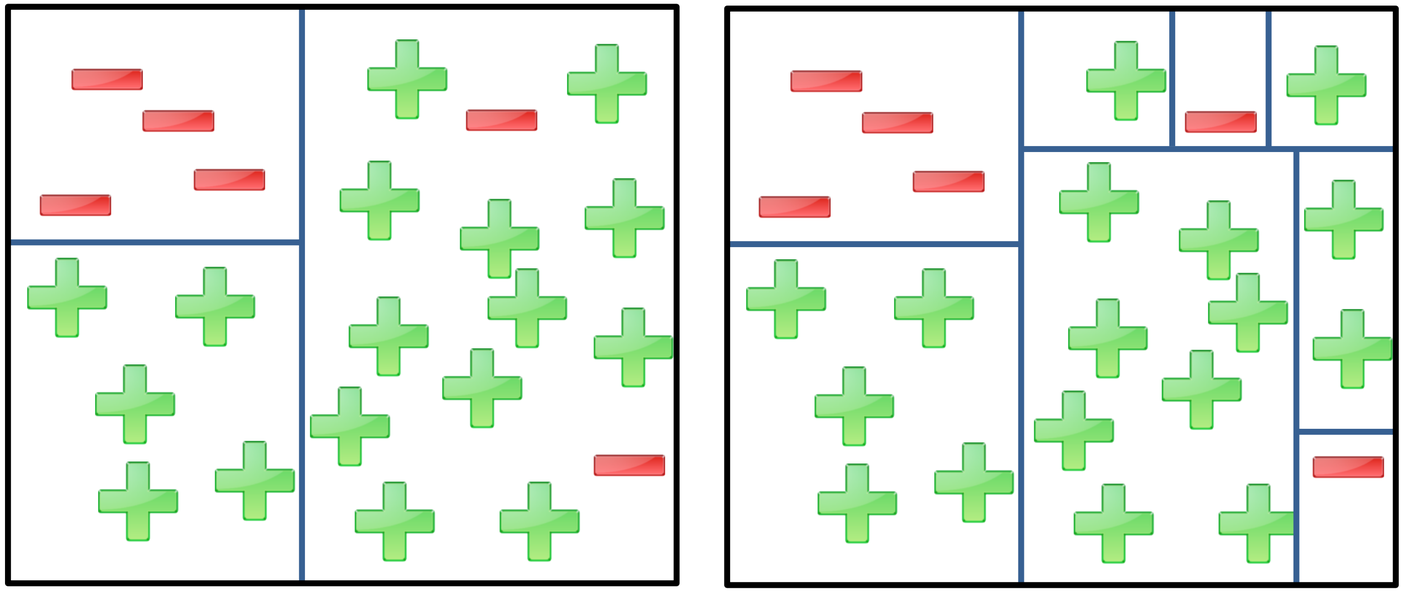
$$L(S') = |S'| \left(1 - p_{S'}^2 - (1 - p_{S'})^2 \right)$$



- All behave similarly

When/how to stop splitting?

- When do we stop growing a tree?
  - When all the nodes are pure? No, that's overfitting!
- How do we regularize?



Stopping conditions (regularizers)

- Minimum size: do not split if resulting children are smaller than a minimum size
- Maximum depth: do not split if the resulting children are beyond some maximum tree depth
- Maximum number of nodes: do not split if tree already has maximum number of allowable nodes
- Minimum reduction in impurity: do not split if resulting children do not reduce impurity by at least $\delta\,\%$

Single decision trees

- Pros:
  - Requires little data preparation (unlike neural networks)
  - Can use continuous and categorical inputs
- Cons:
  - Danger of overfitting training data
  - Sensitive to fluctuations in training data
  - Hard to find global optimum
  - When to stop slitting?

Ensemble methods: combine weak learners

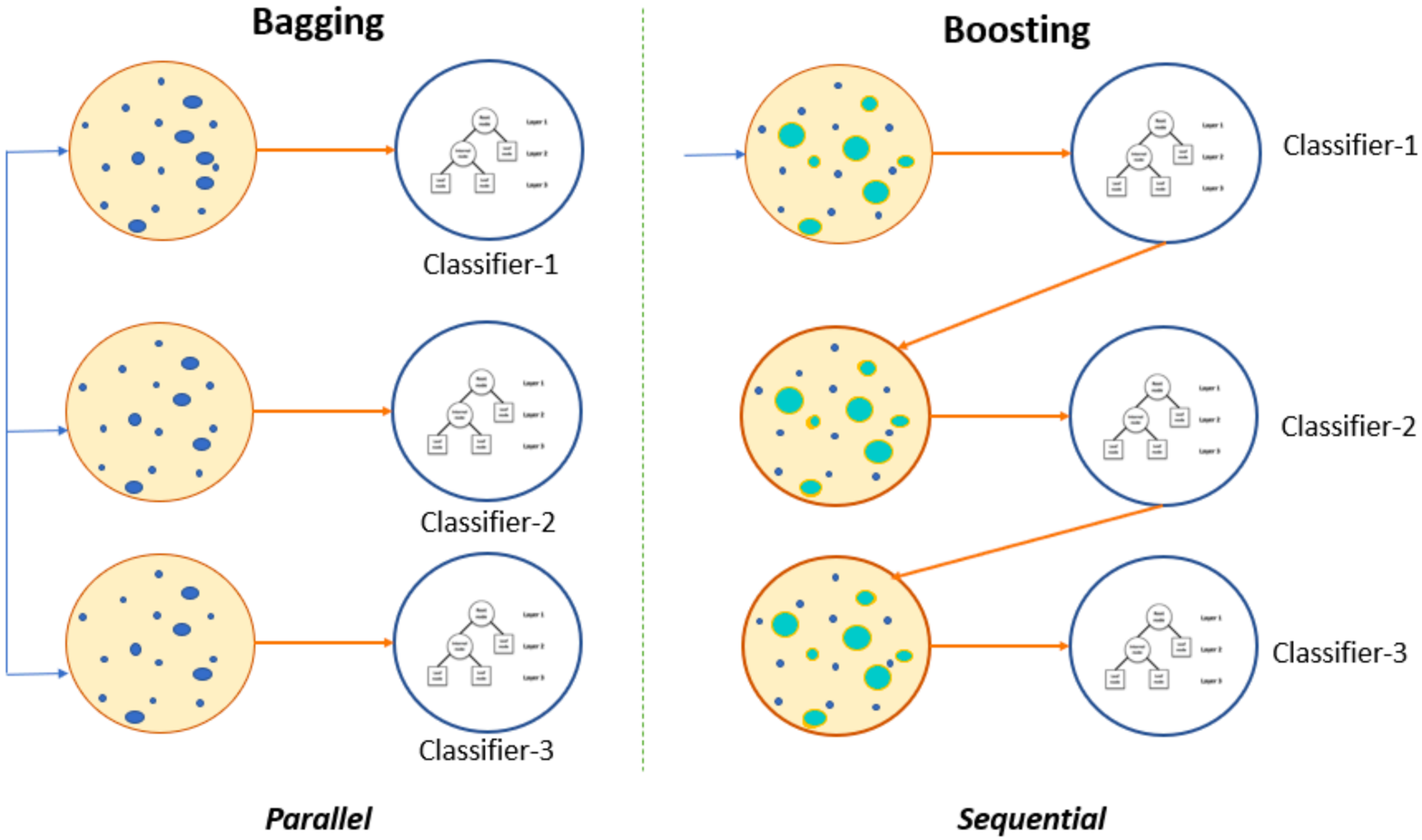
- Bootstrap aggregation (bagging)
  - Sample training data (with replacement) and train a (minimally regularized) tree on each of the derived training sets ⇒ high variance, low bias
  - Classify example with majority vote or compute average output from each tree as model output
  - Reduce **variance** of **low-bias models**

$$f(x) = \frac{1}{N_{\text{trees}}} \sum_{i=1}^{N_{\text{trees}}} f_i(x)$$

- Boosting
  - Train (highly regularized) models in sequence, giving more weight to examples not correctly classified by previous model ⇒ high bias, low variance
  - Take weighted average to classify examples
  - Reduce **bias** of **low-variance models**

$$f(x) = \sum_{i=1}^{N_{\text{trees}}} \alpha_i f_i(x)$$

Bagging vs. boosting



Tree boosting

- Each tree is created iteratively
- Tree's output $f_t(x)$ is given a weight w_i relative to its accuracy
- The ensemble output is the weighted sum

$$f(x) = \sum_{t=1}^{N_{\text{trees}}} \alpha_t f_t(x)$$

- After each iteration, each data sample is given a weight based on how often it's misclassification
- Goal is to minimize the objective function

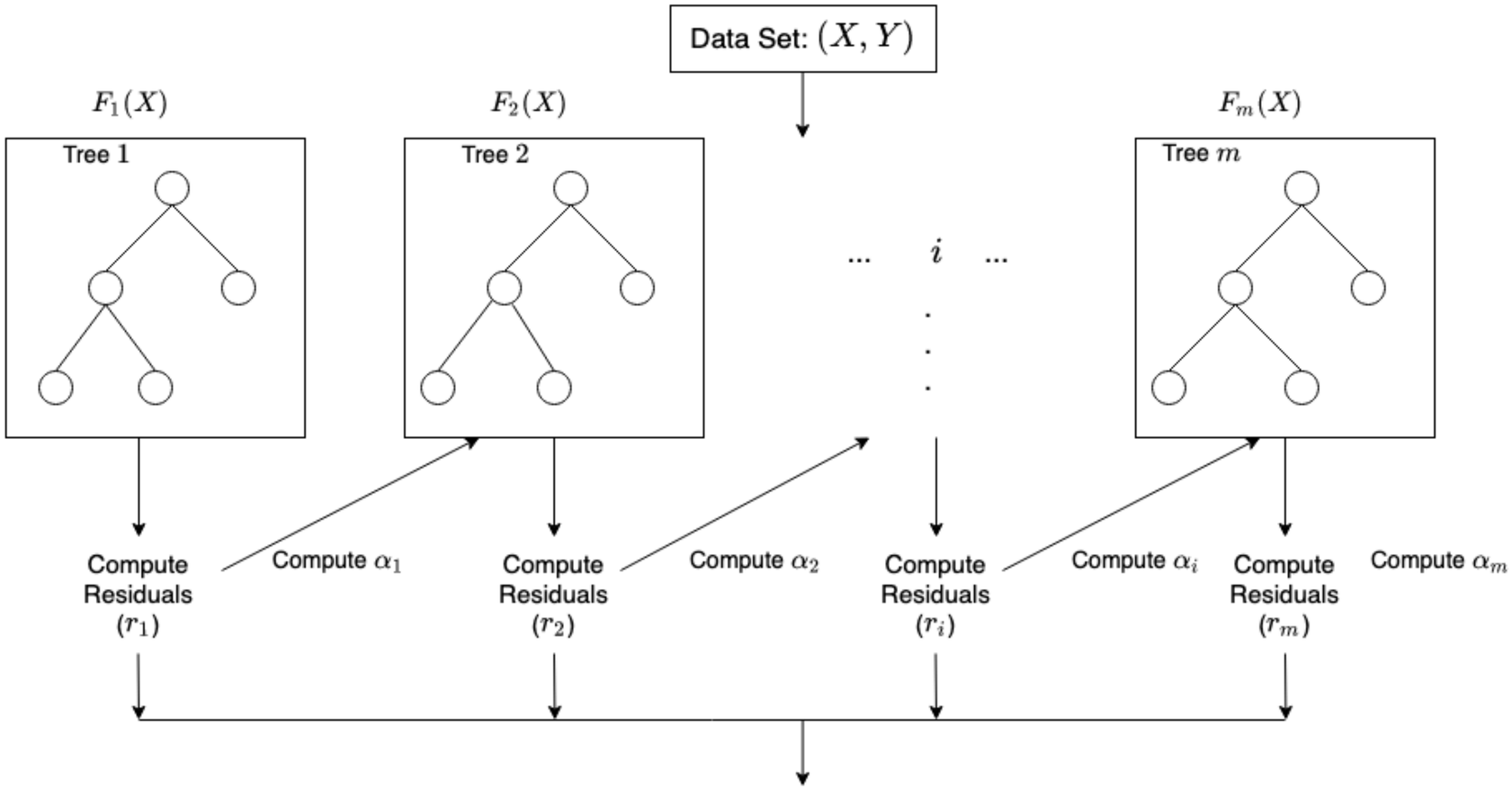
$$l(S) = \sum_{i=1}^{N} L(f(x_i), x_i) + \sum_{t=1}^{N_{\text{trees}}} \Omega(f_t)$$ where L is the loss function and Ω is a regularization term

Types of boosting

- Adaptive boosting (AdaBoost)
  - One of the originals
  - Freund and Schapire (1997): 10.1006/jcss.1997.1504
- Gradient boosting
  - Uses gradient descent to create new learners
  - The loss function is differentiable
  - Friedman (2001): 10.1214/aos/1013203451
- **Extreme gradient boosting (XGBoost)**
  - Very popular in data science (Kaggle) competitions
  - Chen and Guestrin (2016): 10.1145/2939672.2939785

XGBoost

How it works: https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html



$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

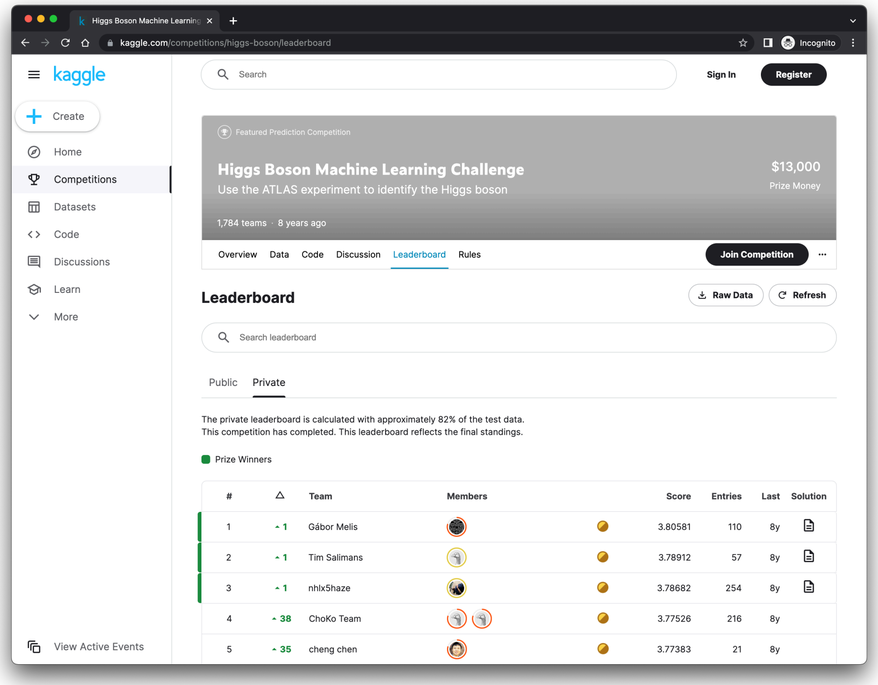
where α_i, and r_i are the regularization parameters and residuals computed with the ith tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the ith tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

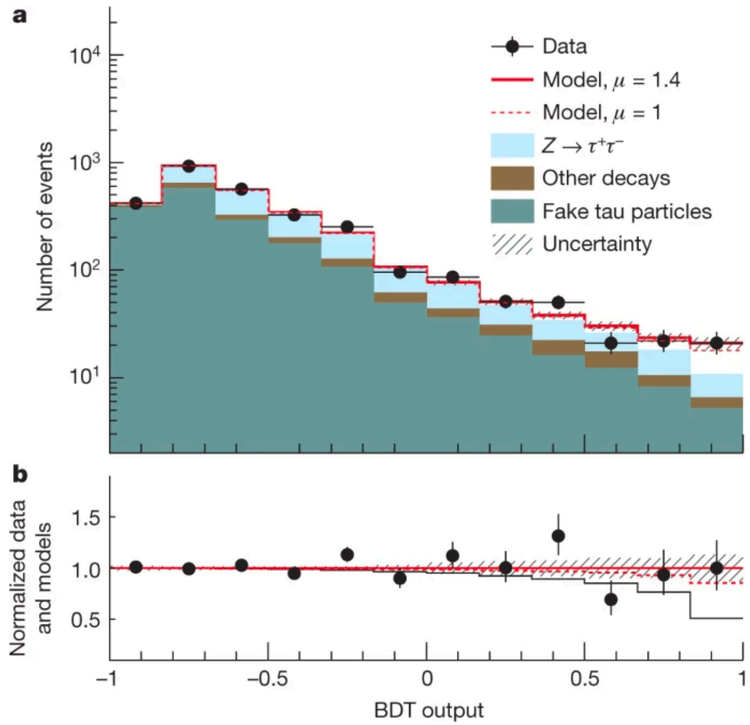
Tunable parameters

- Loss function: How to define the distance between the truth and the prediction
  - Use binary logistic when you have two classes
- Learning rate: how much to adjust data weights after each iteration
  - Smaller is better but slower
- Subsample size: How many samples to train each new tree
  - Data samples are randomly selected each iteration
- Number of trees: How many total trees to create
  - This is the same as the number of iterations
  - Usually more is better, but could lead to overfitting

BDTs in the wild

- 1st place in Kaggle Higgs Boson Machine Learning Challenge [kaggle.com/competitions/higgs-boson]
  - And many other uses at LHC, e.g. in Higgs boson discovery [10.1038/s41586-018-0361-2]
- Predicting critical temperature of a superconductor [10.1016/j.commatsci.2018.07.052]
- MiniBooNE neutrino event classification [10.1016/j.nima.2004.12.018]
- Observation of single top quark production at D0 [10.1103/PhysRevLett.103.092001]





Next time

- Neural networks