Machine Learning

General Framework
Near-Neighbor Learning
Decision Trees

Jeffrey D. Ullman
Stanford University/Infolab

The ML Framework

Model Selection and Optimization
Testing and Validation
Overfitting

Framework

- We are given a set of *training examples*, consisting of *input-output* pairs (x,y), where:
  1. x is an item of the type we want to evaluate.
  2. y is the value of some function f(x).
- Example: x is an email, and f(x) is +1 if x is spam and -1 if not.
  - Binary classification.
- Example: x is a vector giving characteristics of a voter, and y is their preferred candidate.
  - More general classification.

Framework – (2)

- In general, input x can be of any type.
- Often, output y is binary.
  - Usually represented as +1 = true, or "in the class" and -1 = false, or "not in the class."
  - Called *binary classification*.
- y can be one of a finite set of categories.
  - Called *classification*.
- y can be a real number.
  - Called *regression*.

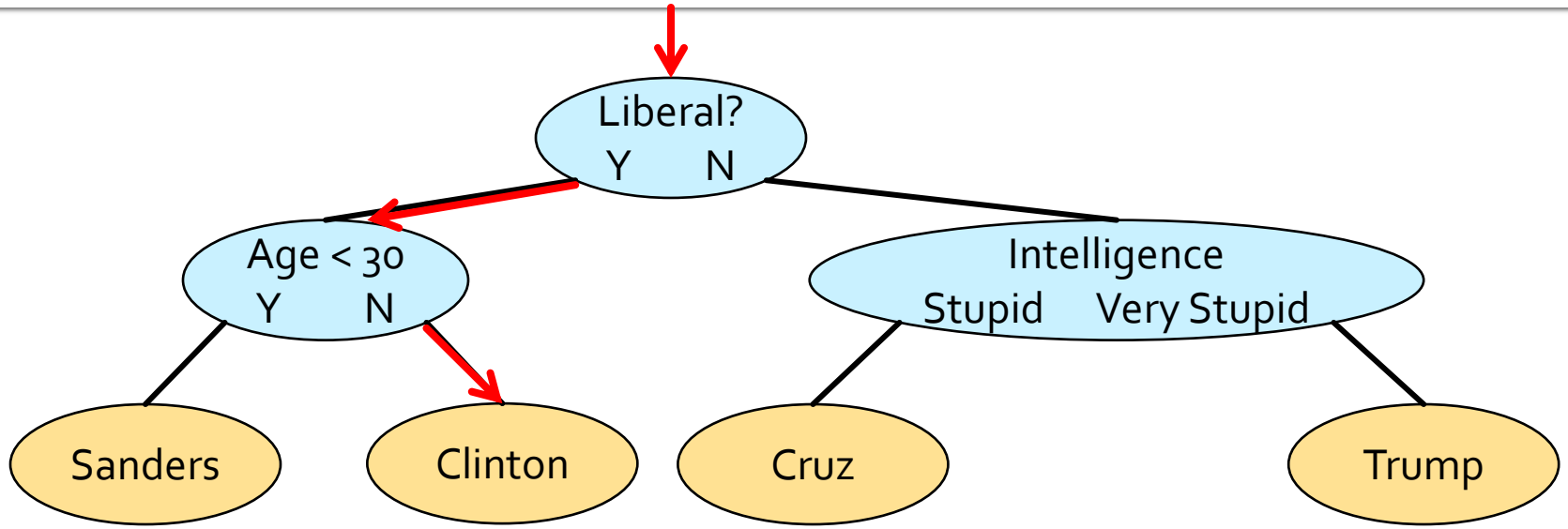
Supervised Learning

- *Supervised learning* is building, from the training data, a model that closely represents the function y = f(x).
- Example: If x and y are real numbers, the model of f might be a straight line.
- Example: if x is a email and y is +1 or -1, the model might be weights on words together with a threshold such that the answer is +1 (spam) if the weighted sum of the words in the email exceeds the threshold.

Example: Decision Trees

- A decision tree is a model in the form of a tree where:
  - Interior nodes have tests about the value of x, with one child for each possible outcome of the test.
  - Leaves have a value for f(x).
- Given an x to be tested, start at the root, and perform the tests, moving to the proper child.
- When you reach a leaf, declare the value of f(x) to be whatever is found at that leaf.

Example: A Decision Tree



The Loss Function

- We need to choose a *loss function* that measures how well or badly a given model represents the function y = f(x).
- Common choice: the fraction of x's for which the model gives a value different from y.
- Example: if we use a model of email spam that is a weight for each word and a threshold, then the loss for given weights + threshold could be the fraction of misclassified emails.

Loss Function – (2)

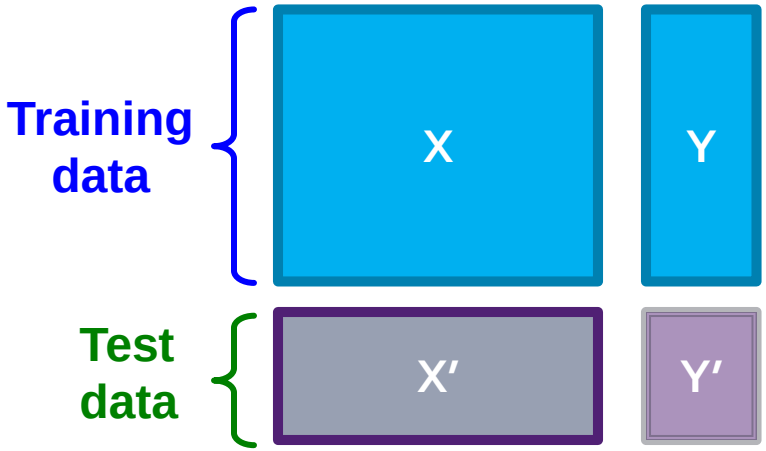
- But are all errors equally bad?
- Question for thought: do you think that the loss should be the same for a good email classified as spam and a spam email passed to the user's inbox?
- If y is a numerical value, cost could be the average magnitude of the difference between f(x) as computed by the model, and the true y.
  - Or square each error (like RMSE).
    - Subtle point: squaring errors makes the loss function much more tolerant of small errors, but not big ones.

The Optimization Problem

- Given training data and a form of model you wish to develop, find the instance of the model that minimizes the loss on the training data.
- Example: For the email-spam problem, incrementally adjust weights on words until small changes cannot decrease the probability of misclassification.
- Example: design a decision tree top-down, picking at each node the test that makes the branches most homogeneous.

Validation

- Divide your data randomly into training and *test* data.
- Build your best model based on the training data only.
- Apply your model to the test data.
- Does your model predict y' for the test data as well as it predicted y for the training data?



Overfitting

- Sometimes, your model will show much greater loss on the test data than on the training data.
  - Called *overfitting*.
- The problem is that the modeling process has picked up on details of the training data that are so fine-grained that they do not apply to the population from which the data is drawn.
- Example: a decision tree with so many levels that the typical leaf is reached by only one member of the training set.

The Real Test

- The test data helps you measure overfitting.
- But you want your model to work not only on the test data, but on all the unseen data that it will eventually be called upon to process.
  - The *validation* set.
- If the training and test sets are truly drawn at random from the population of all data, and the test set shows little overfitting, then the model should not exhibit overfitting on real data.
  - A big "if," e.g., with email spam, where the population is always changing.

Near-Neighbor Learning

Training Sets as Models
Some Options

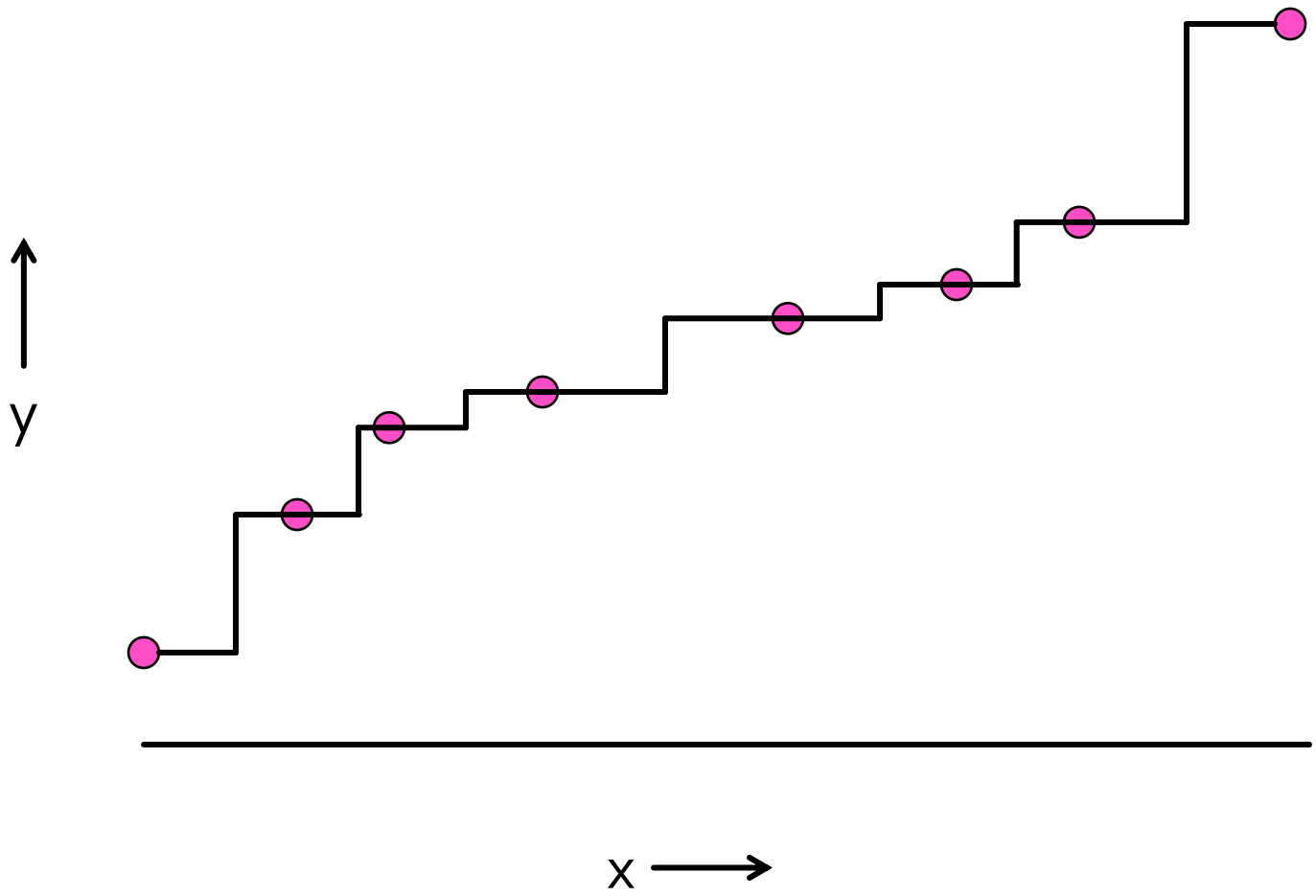
The Big Idea: Training Set = Model

- Given a query q (a data point), find the nearest inputs (values of x) in the training set.
- Combine the outputs y associated with these values of x, in some way.
- Result is the estimated output for the query q.

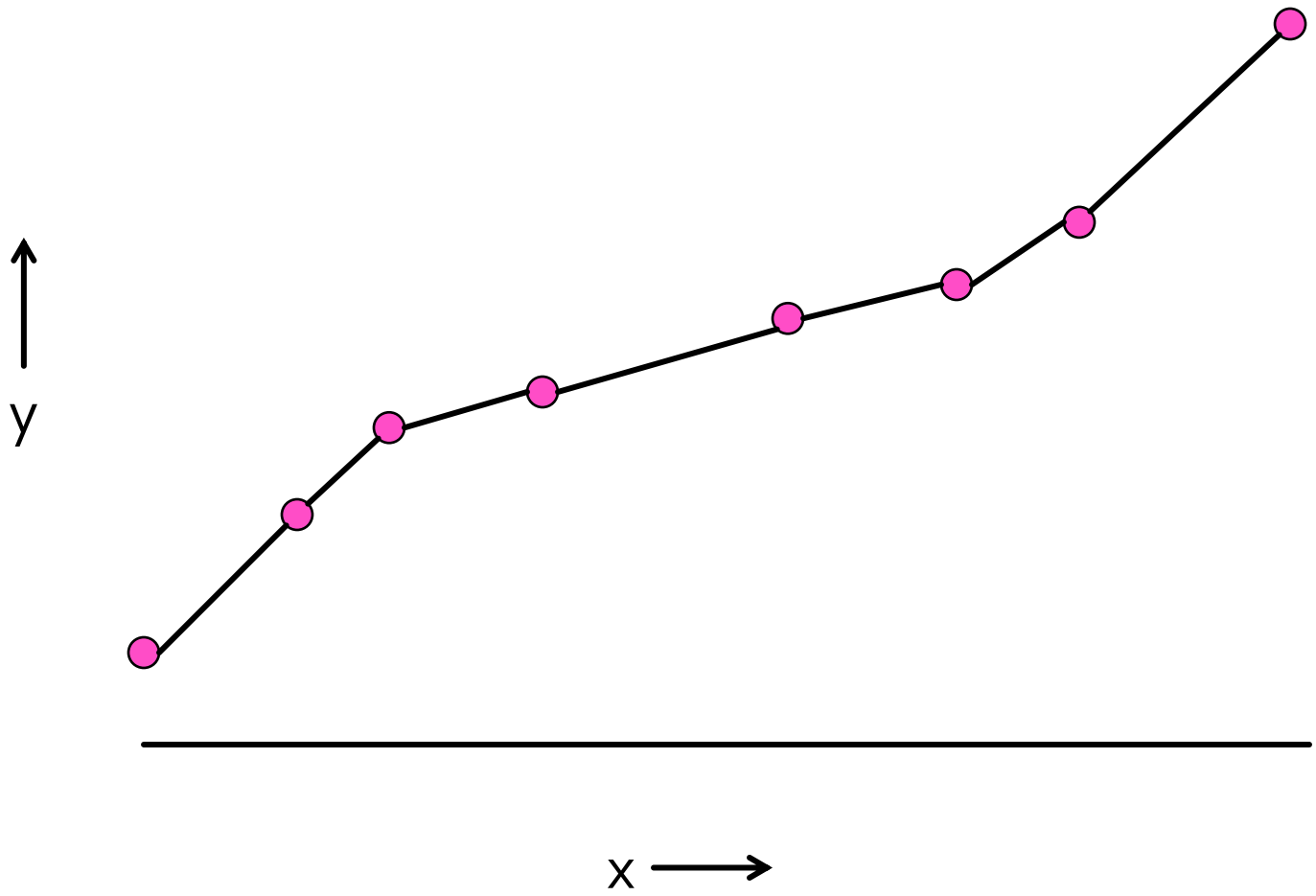
Example: Interpolation

- Input for training set is a number.
- Output is the value of some function.
- For any query point, take the two nearest points in the training set and
  a. (Option 1): Average their outputs.
  b. (Option 2): Take the average of their outputs, weighted by the distance to the other point.
     - I.e., give more influence to the closer point.

Example: Nearest Point



Example: Weighted Average



Things to Worry About

1. How do we find nearest neighbors, especially in very high-dimensional spaces?
2. How many neighbors do we consider?
3. How do we weight the influence of each near neighbor?

K-Nearest Neighbor Approach

- For some chosen k, find the k training points nearest to the query point q, according to some distance measure d.
- If y is numerical, blend the y's for the k nearest x's.
  - Weight (inversely) by distance or just take the average.
- If y is not numerical, combine in some appropriate way.
  - Example: If y is a category (e.g., a presidential candidate), take the value appearing most often.

Kernel Regression

- All training points contribute to the estimate of y for a query point q.
- Assumes some distance function d.
- Weight of training point x is a function g (the *kernel function*) of d(x,q).
- Predict f(q) = average of y's for all training points, weighted by g(d(x,q)).
- Common example: g(d) is a scaled Gaussian distribution $c*\exp(-d^2/\sigma^2)$.

Finding Nearest Neighbors

- Many data structures that have been designed to find near neighbors of a query point in a high-dimensional space (somewhat) efficiently: kd-trees, Quad trees, R-trees, Grid files.
  - Not covered in CS246; may be worth Googling.

Nearest Neighbors by LSH

- Locality-sensitive hashing may be an option.
- Previously used, in LSH discussion, for many-many similarity search.
- Rather, bucketize the training set as if you were looking for similar training points.
- Given query point q, hash it to buckets using the same process as for the training points.
- Compare q with members of all the buckets into which it falls, but do not add it to the buckets.
- As usual, false negatives are possible.

Large-Scale Decision Trees

Quality Measures
Efficient Construction of Nodes
Dealing With Overfitting

Some material borrowed from Hendrik Blockeel

The Setting

- We are given a training set of input/output pairs (**x**,y).
- Inputs **x** are vectors whose components correspond to attributes of entities.
- Our goal is to build a decision tree, where each node looks at the value of one attribute and sends us to one of its two children (left or right), depending on that value.
- Leaves declare a value for the output y.
  - Hopefully, all the inputs that get us to that leaf have the same y; else there are incorrect classifications.

Designing a Node

- At each step of the decision-tree construction, we are at one node of the tree, and we have a set S of training examples getting us to that node.
- Our goal is to find a test that partitions S into sets S1 and S2 that are as close to *pure* (all examples in the set have the same output) as possible.
- Tests are normally simple: compare one attribute to a value, not complex logical expressions.
  - Not allowed: "IF Age < 30 AND (hair != "blond" OR eyes = "blue") AND …"

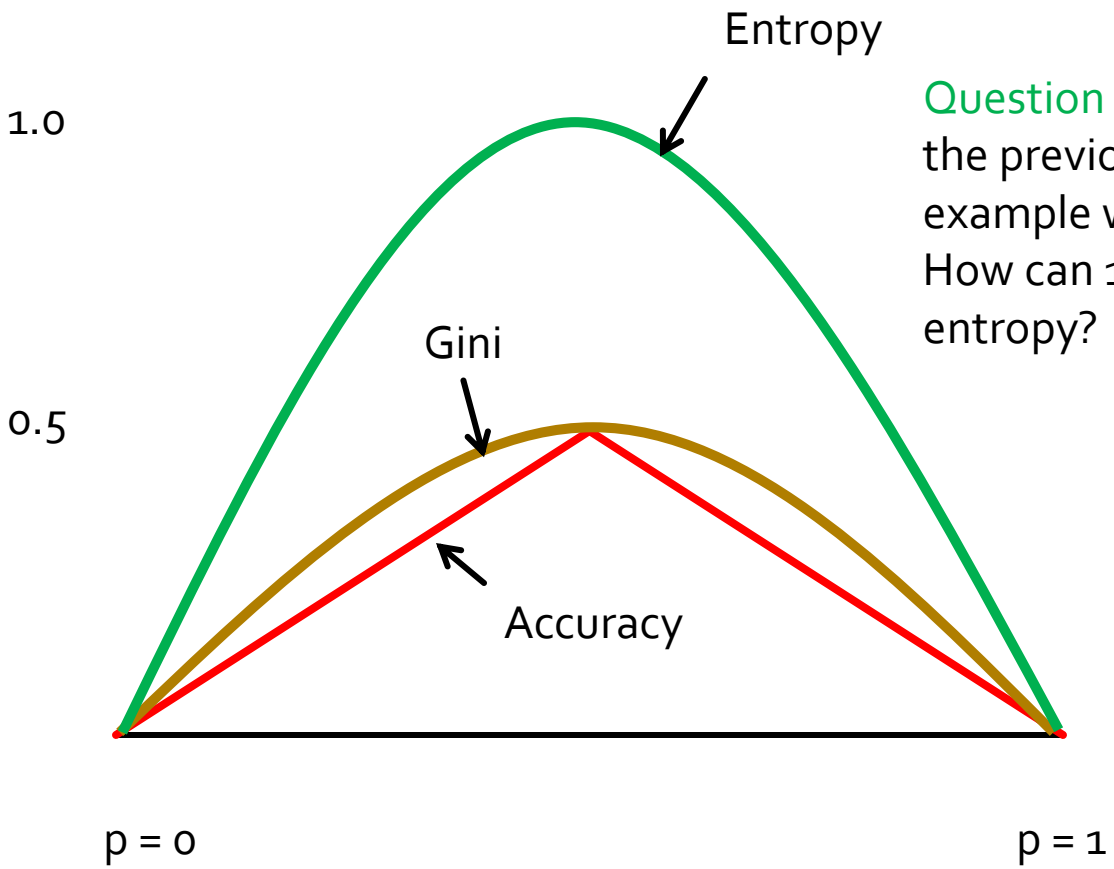
Measuring Impurity of a Set S

- Let $p_1,\ldots,p_k$ be the fractions of members of S with the k possible values of y.
  1. Accuracy: if output is the most common value of y, what fraction of inputs in the set S are not given their correct output; i.e., $1 - \max_i p_i$.
  2. GINI Impurity: $1 - \sum_i (p_i)^2$.
  3. Entropy: $\sum_i -p_i \log_2 p_i$ or equivalently $\sum_i p_i \log_2(1/p_i)$.

Example: Impurity

- Suppose S consists of examples with three possible outputs, with probabilities 2/3, 1/4, and 1/12.
  - Note: We are measuring impurity, so high is bad.
- Accuracy-based impurity = 1/4 + 1/12 = 1/3.
- GINI = $1 - (2/3)^2 - (1/4)^2 - (1/12)^2 = 35/72$.
- Entropy = (2/3) log(3/2) + 1/4 log(4) + 1/12 log(12) = 1.19.

Impurity When 2 Outcomes With Probabilities p and 1-p



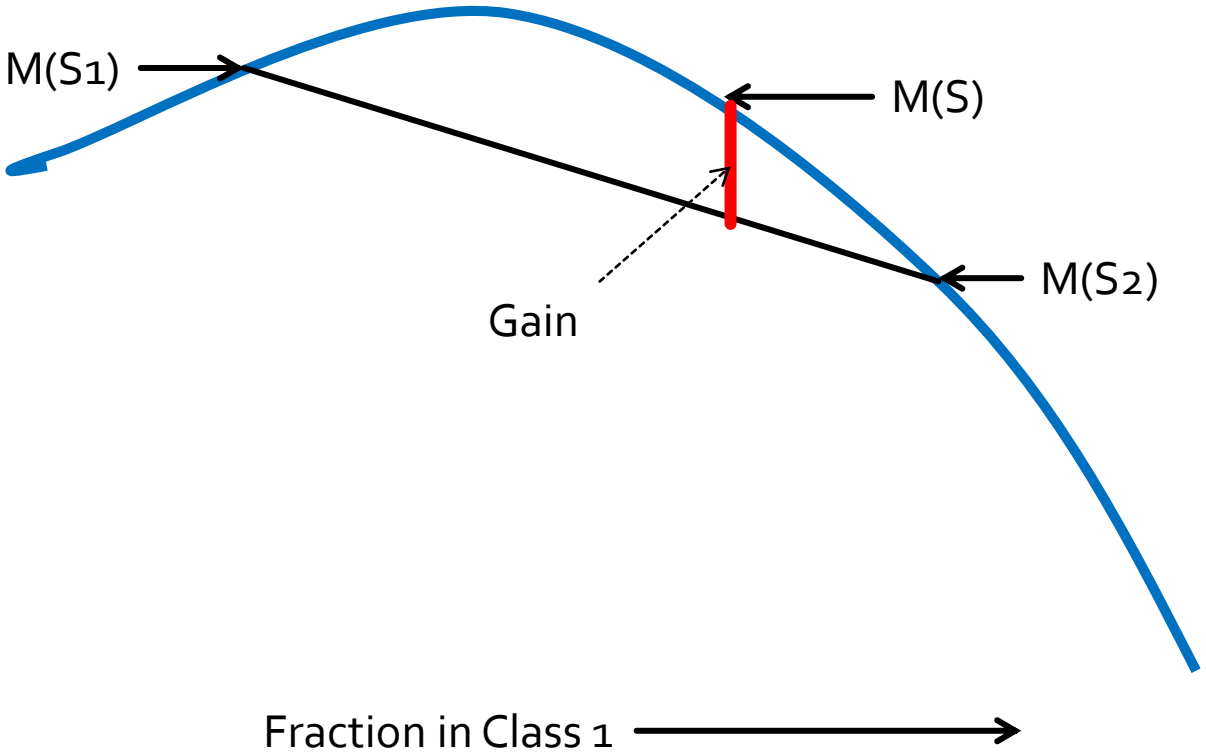
Question for thought: Didn't the previous slide have an example with Entropy = 1.19? How can 1.0 be the maximum entropy?

Good Impurity Measures

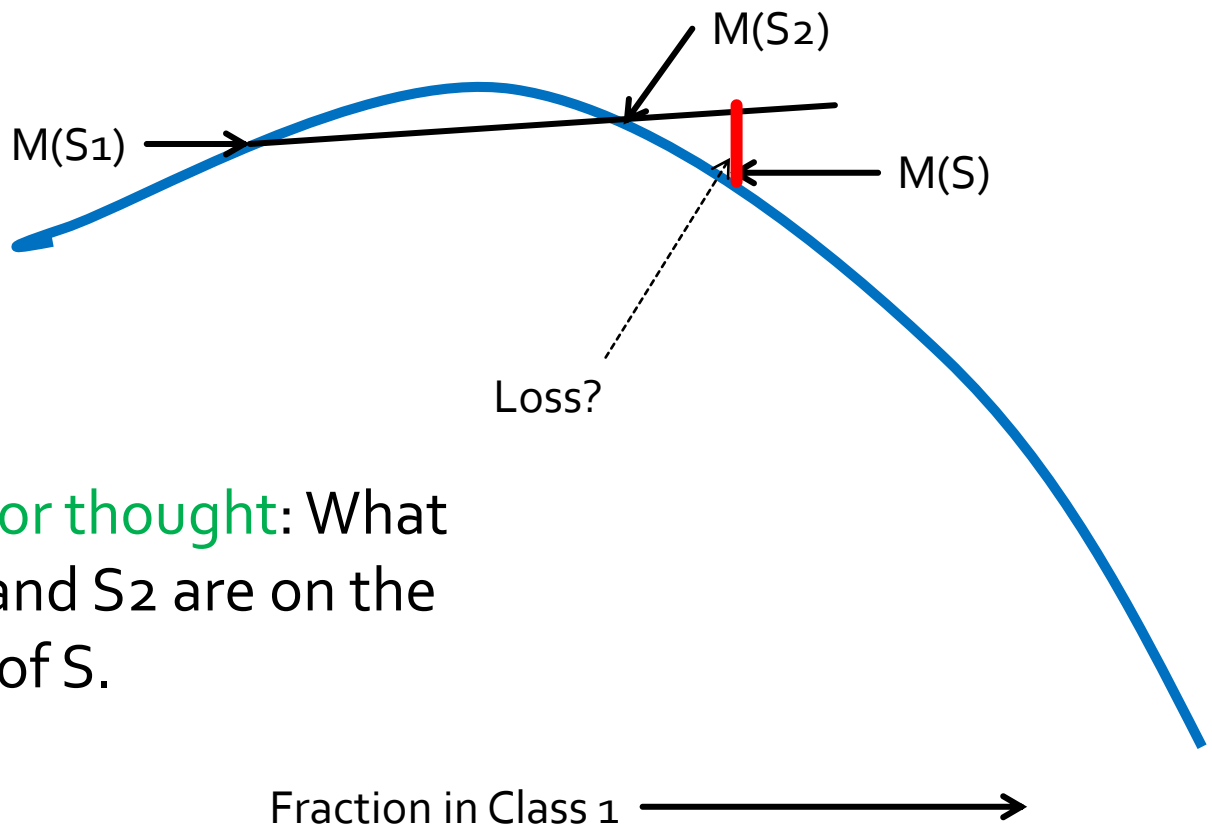
1. If a set S is pure, then its impurity measure is 0.
2. Impurity is concave downward.

- GINI and Entropy have these properties; Accuracy lacks (2).
- When we partition S into S1 and S2, we *gain* according to impurity measure M if $|S1|M(S1) + |S2|M(S2) < |S|M(S)$.
  - I.e., the weighted-average impurity drops.

Concave => Always Gain

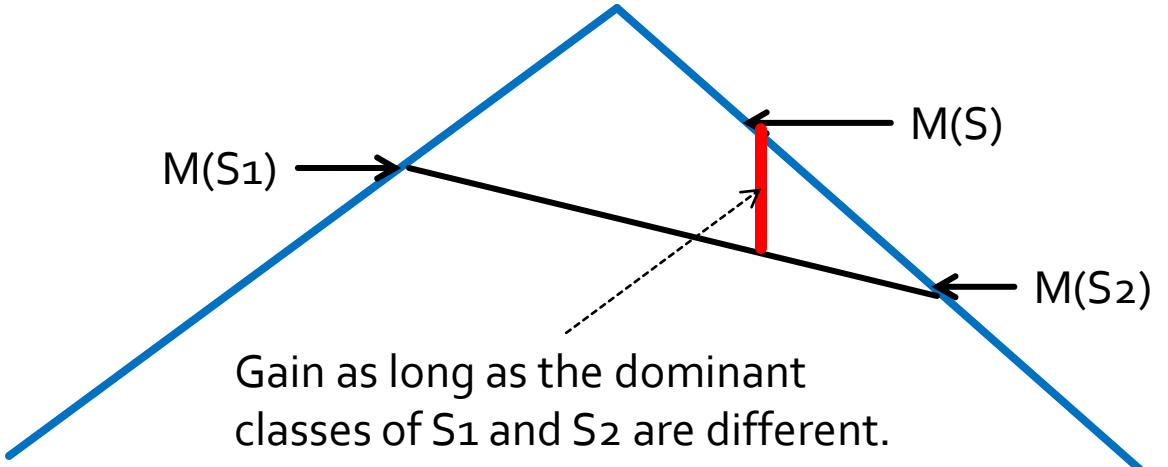


Concave => Always Gain



Question for thought: What if both S1 and S2 are on the same side of S.

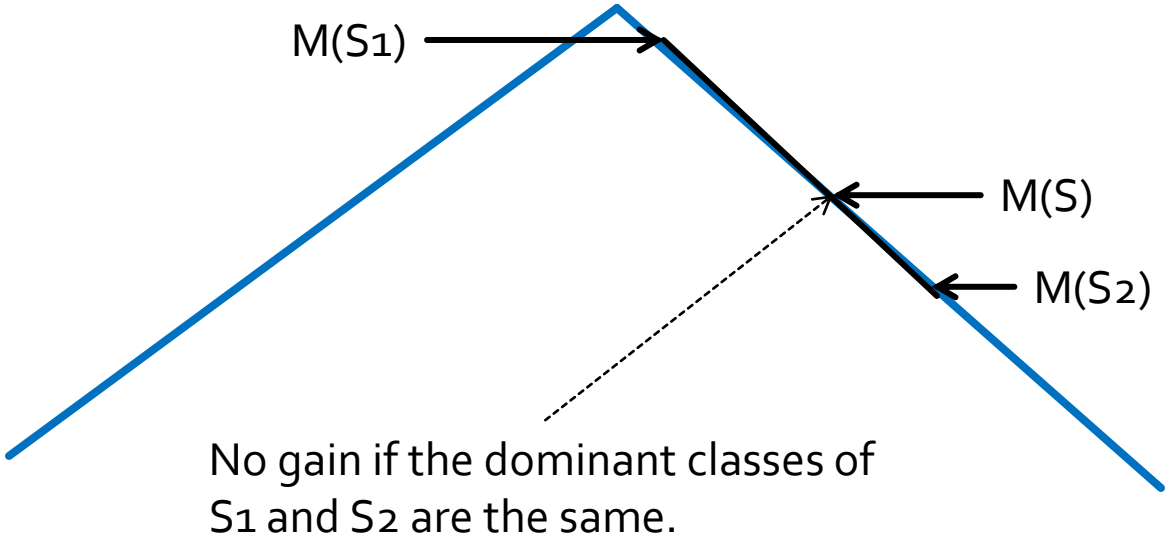
Problem With the Accuracy Measure



Gain as long as the dominant classes of S1 and S2 are different. Otherwise, M(S) is the weighted average of M(S1) and M(S2), so no gain.

Fraction in Class 1 ⟶

Problem With the Accuracy Measure



Partitioning a Set

- Given set S of examples, we want to find the simple test that partitions S into S1 and S2 in a way that maximizes the gain.
- Consider each component of the input vector to see which simple comparison breaks S into the sets with the lowest weighted-average impurity.
- Case 1: numerical attributes.
  - Comparisons are of the form attribute < constant.
- Case 2: discrete-value attributes.
  - Comparisons are of the form attribute in {set of values}

Partitioning Using Numerical Attributes

- Consider an attribute (component of **x**) A with a numerical value.
- Sort the set of examples S according to A.
- Visit each example (**x**,y) in sorted order.
- Keep a running count of the number of examples in each class.
- As we visit each example, compute the measure M (e.g., GINI or Entropy) assuming the split is just after that example.
- Remember the point at which the minimum M occurs; that is the best split using A.

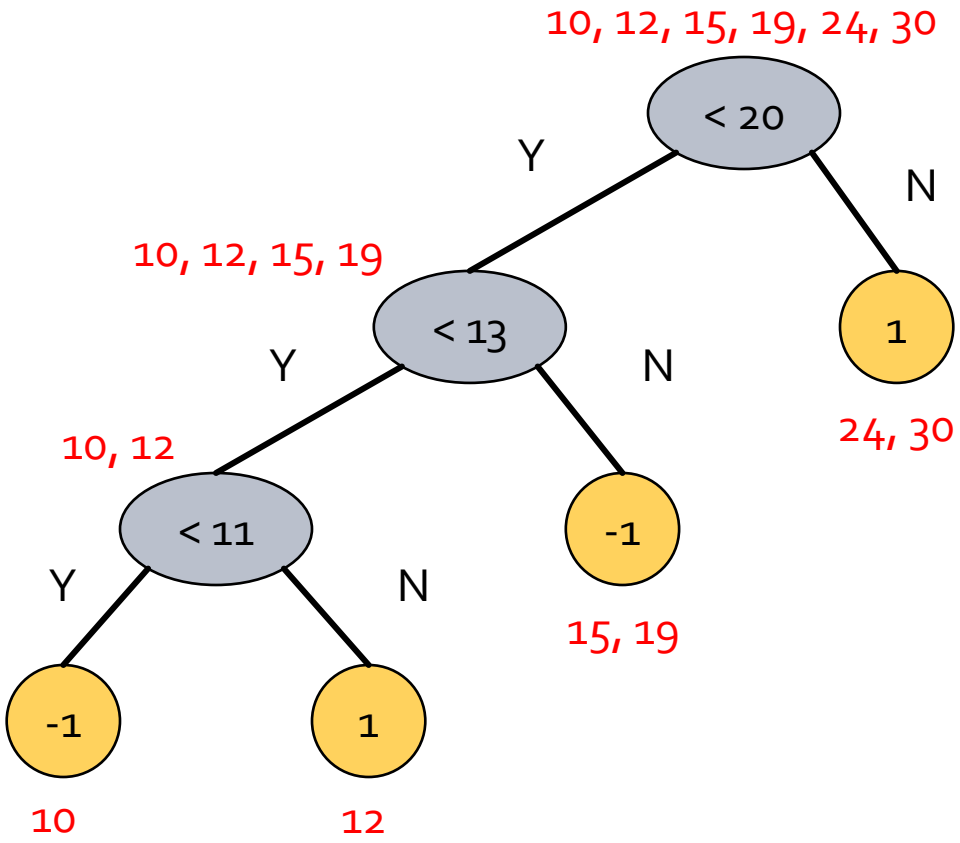
Example: Uses GINI

Example, assumes x = A	M(S1) if S1 ends here	M(S2) if S2 begins below	Weighted average
(10, -1)	0	12/25	2/5
(12, +1)	1/2	1/2	1/2
(15, -1)	4/9	4/9	4/9
(19, -1)	3/8	0	1/4
(24, +1)	12/25	0	2/5
(30, +1)			

Example: for the test A < 11, M(S1) = 0;
M(S2) = $1 - (2/5)^2 - (3/5)^2 = 12/25$.
Weighted average = (1/6)*0 + (5/6)*12/25 = 2/5.

Best choice: test = A < 20 (or maybe A < 21.5 to split the difference between 19 and 24).

Complete Design



Note: decisions need not be based on this one attribute.

Parallelization – Sorting

- Sorting can be done with a lot of parallelism.
  - "Batcher Sort uses $O(\log^2 n)$ rounds, and there are $O(\log n)$ expected-time randomized algorithms.
- More relevant: two-pass *external* (disk-based) parallel mergesort can be done on petabytes today.

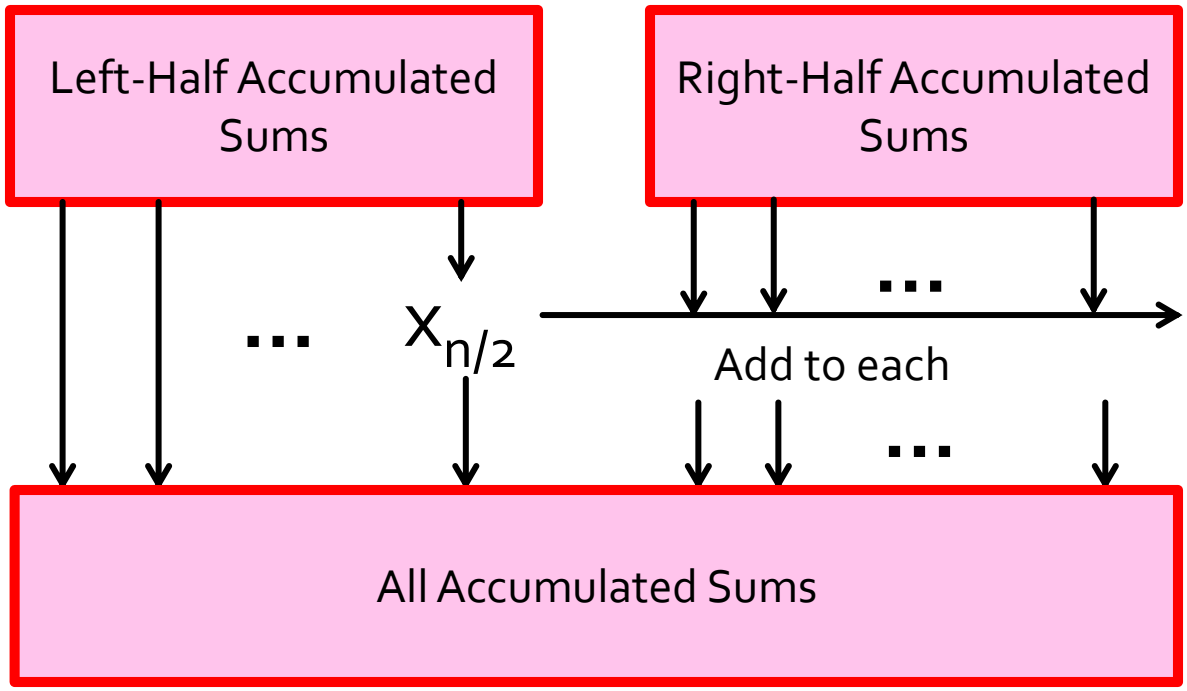
Parallelization – Finding Splits

- We can compute the best split for each numerical attribute in parallel.
  - Remember, what we showed in the example was only for one of possibly many attributes, and we want only the split with the biggest gain.
- Computing the running sum of occurrences of each output value is also parallelizable in O(log n) rounds.
  - Parallel algorithm on next slide.
  - From these sums, you can compute the impurity of a split at each point, in parallel.

Parallel Accumulated Sums

- Given $a_1, a_2, \ldots, a_n$, compute $x_i = \sum_{j=1,2,\ldots,i} a_i$ for all i.
- Basis: if n=1, then $x_1 = a_1$.
- Induction: In parallel, compute the accumulated sums for the left half $a_1, \ldots, a_{n/2}$ and for the right half $a_{(n/2)+1}, \ldots, a_n$.
- Then, add $x_{n/2}$ from the left half to every sum from the right half, in one parallel step.
- Application: Let a_i be 1 if the i-th training example is positive, 0 if negative.
  - Or vice-versa.

Picture of Recursion



Partitioning Using a Discrete Attribute

- For each value of the discrete attribute A, compute the number of positive and negative examples in the training set.
- Sort the values of A by the fraction of positive examples.
- Visit each value in sorted order, keeping a running count of the number of examples in each class.
- For each prefix of the value list, apply the impurity measure, and remember the best.

Example: GINI Again

Value of attribute	Positive examples	Negative examples	M(S1) if S1 ends here	M(S2) if S2 begins below	Weighted average
a	10	1	.165	.499	.420
b	7	3	.308	.480	.401
c	6	5	.404	.408	.405
d	4	10			

Positive = 17; negative = 4.
$1 - (17/21)^2 - (4/21)^2 = .308$.

Best choice.
Count of S1 = 10+1+7+3 = 21.
Count of S2 = 6+5+4+10 = 25.
Weighted average =
.308*(21/46) + .480*(25/46).

Parallelization

- Counting positive and negative examples for each attribute value is a group-and-aggregate.
  - One round of MapReduce suffices.
- Sorting attribute values is $O(\log^2 n)$ at most.
  - Note: this "n" is the number of different values of the attribute, not the number of training examples.
- Accumulated sums of positive and negative examples is O(log n).

Global Parallelization

- As we build the decision tree top-down, we double the number of nodes at each level.
- But each training example goes with only one node at each level.
- Thus, the total work is the same at each level.
- And we can work on all the nodes at a level in parallel.

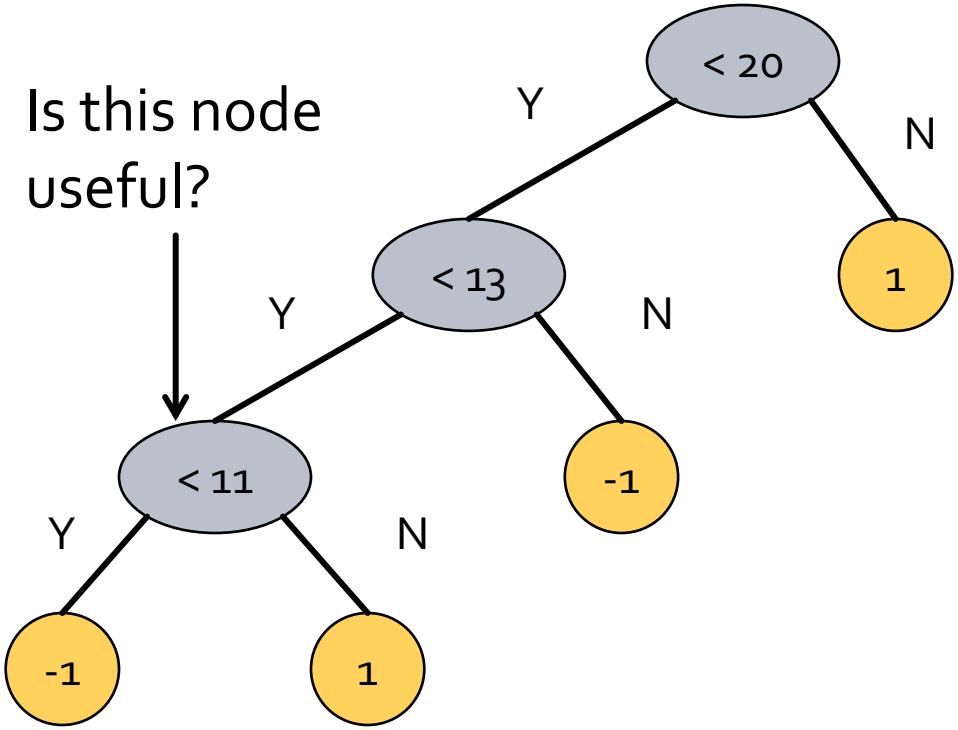
Avoiding Overfitting

- We can add levels to the tree until all leaves are pure.
- At that point, there is 100% accuracy on the training set.
- But there might be significant error on the test set, because we have wildly overfit to the training set.
- *Post-pruning* eliminates interior nodes if they do not contribute (much) to the accuracy on the test set.

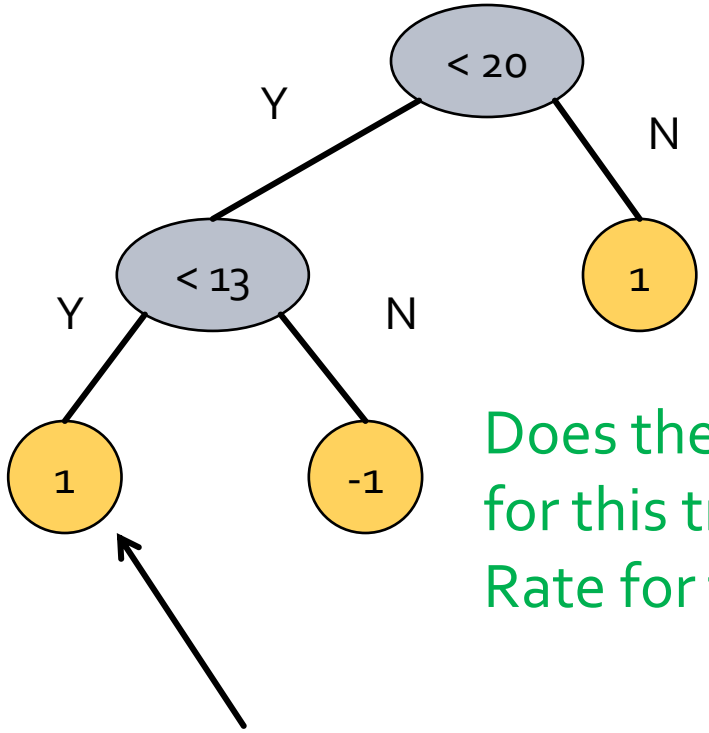
Post-Pruning

- Eliminate the test at a node N, both of whose children are leaves.
- Makes N a leaf; give it the output that is the majority of its children.
- Compare the performance of the original tree and the pruned tree on the test set.
- If difference is small, accept the pruning and repeat.
- If the difference is large, restore the children and look for other places to prune.

Example: Post-Pruning



Example: Post-Pruning



Does the error rate (on the test set) for this tree exceed by much the error Rate for the tree on the previous slide?

Since an equal number of training examples went to either side, we picked +1 arbitrarily.