Decision Trees for Classification and Regression, PLUS Ensembles

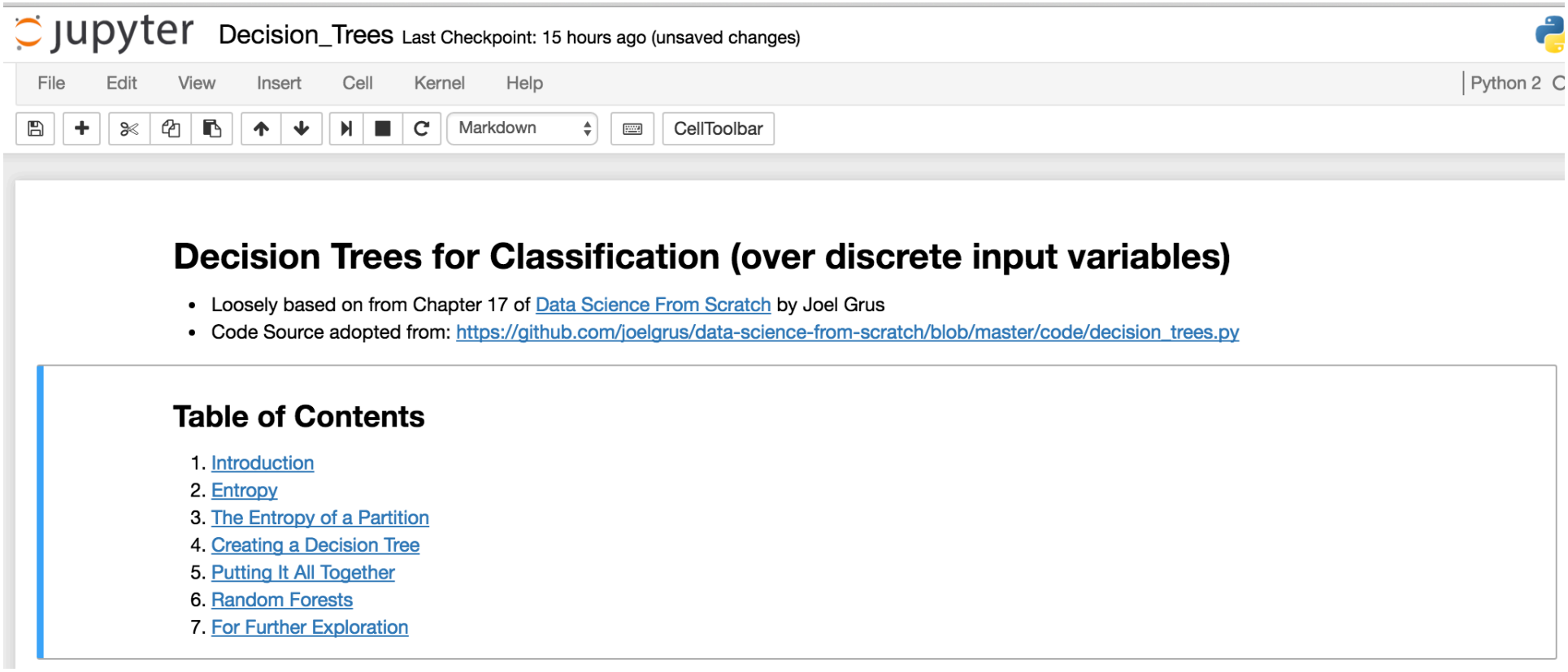
James G. Shanahan [1, 2]
Church and Duncan Group[2]*iSchool UC Berkeley, CA,*

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Lecture 13 August 5, 2016
Virtual Lecture

Companion Notebook

- http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/p99ro9v9z81wtwh/Decision_Trees.ipynb



Live Session Outline

- Gradient descent versus non-gradient descent approaches
- Decision Trees
- Classification DTs on discrete variables
- Regression DTs over continuous variables [Optional]
- Ensembles (of decision Trees) [Optional]
- Practical decision trees (single core/Spark)

Gradient Descent versus NonGradient descent

- **Gradient descent**
  - Sometimes convex therefore can find global optimal in a principled way
  - Linear regression, logistic regresion, Perceptrions, SVMs etc.
- **Non-gradient descent**
  - more local in nature;
  - Different types of search algorithms. E.g., hill climbing, heuristic in nature
  - Coordinate descent
  - Decision trees [Focus of this lecture]
    - Classification
    - Regression
  - Ensembles

Coordinate descent

https://en.wikipedia.org/wiki/Coordinate_descent

Coordinate descent is a non-derivative optimization algorithm. To find a local minimum of a function, one does line search along one coordinate direction at the current point in each iteration. One uses different coordinate directions cyclically throughout the procedure.

Contents [hide]

Description [edit]

Coordinate descent is based on the idea that the minimization of a multivariable function $F(\mathbf{x})$ can be achieved by minimizing it along one direction at a time, i.e., solving univariate (or at least much simpler) optimization problems in a loop.[1] In the simplest case of *cyclic coordinate descent*, one cyclically iterates through the directions, one at a time, minimizing the objective function with respect to each coordinate direction at a time. That is, in each iteration, for each variable k of the problem in turn, the algorithm solves the optimization problem

$$x_i^{k+1} = \arg\min_{y \in \mathbb{R}} f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, y, x_{i+1}^{k}, \ldots, x_n^{k}).$$

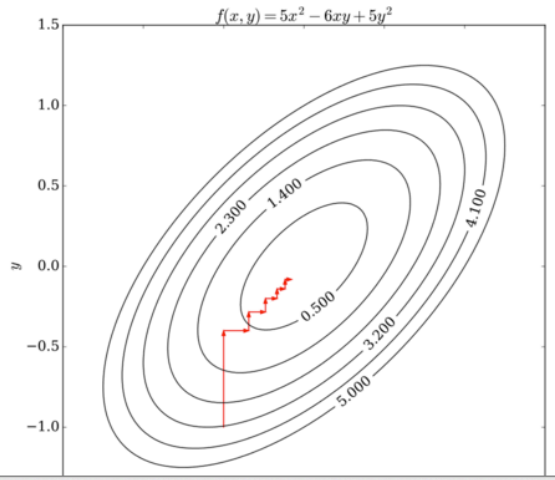
Thus, one begins with an initial guess $\mathbf{x}^0$ for a local minimum of F, and get a sequence $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots$ iteratively.

By doing line search in each iteration, one automatically has

$$F(\mathbf{x}^0) \geq F(\mathbf{x}^1) \geq F(\mathbf{x}^2) \geq \ldots.$$

It can be shown that this sequence has similar convergence properties as steepest descent. No improvement after one cycle of line search along coordinate directions implies a stationary point is reached.

This process is illustrated below.



Description [edit]

Coordinate descent is based on the idea that the minimization of a multivariable function $F(\mathbf{x})$ can be achieved by minimizing it along one direction at a time, i.e., solving univariate (or at least much simpler) optimization problems in a loop.[1] In the simplest case of *cyclic coordinate descent*, one cyclically iterates through the directions, one at a time, minimizing the objective function with respect to each coordinate direction at a time. That is, in each iteration, for each variable k of the problem in turn, the algorithm solves the optimization problem

$$x_i^{k+1} = \arg\min_{y \in \mathbb{R}} f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, y, x_{i+1}^{k}, \ldots, x_n^{k}).$$

Thus, one begins with an initial guess $\mathbf{x}^0$ for a local minimum of F, and get a sequence $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots$ iteratively.

By doing line search in each iteration, one automatically has

$$F(\mathbf{x}^0) \geq F(\mathbf{x}^1) \geq F(\mathbf{x}^2) \geq \ldots.$$

It can be shown that this sequence has similar convergence properties as steepest descent. No improvement after one cycle of line search along coordinate directions implies a stationary point is reached.

Coordinate descent

- Coordinate descent is a non-derivative optimization algorithm.

- To find a local minimum of a function, one does line search along one coordinate direction at the current point in each iteration.
- One uses different coordinate directions cyclically throughout the procedure.

Live Session Outline

- Gradient descent versus non-gradient descent approaches
- Decision Trees
- Classification DTs on discrete variables
- Regression DTs over continuous variables [Optional]
- Ensembles (of decision Trees) [Optional]
- Practical decision trees (single core/Spark)

Reading material and notebook for DTs

- **Chapter 17 on decision Trees,**
  - https://www.dropbox.com/s/5ca98ah5chqlcmn/Data_Science_from_Scratch%20%281%29.pdf?dl=0
    [Please do not share this PDF]
  - Notebook:
    - http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/p99ro9v9z81wtwh/Decision_Trees.ipynb
- **Entropy, and ID3 Decision Tree Learning algorithm**
  - https://www.dropbox.com/s/l0pp4ed5wwv9s1h/FMLPDA_SampleChapter_InformationBasedLearning-DT-Excellent-Examples.pdf?dl=0
- **Many other sources online (one of the most talked/written about approaches to ML)**

Decision Trees are powerful

- **Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of problem at hand (classification or regression).**

- **Methods like decision trees, random forest, gradient boosting are being popularly used in all kinds of data science problems. Hence, for every analyst (fresher also), it's important to learn these algorithms and use them for modeling.**

Decision Tree Approach

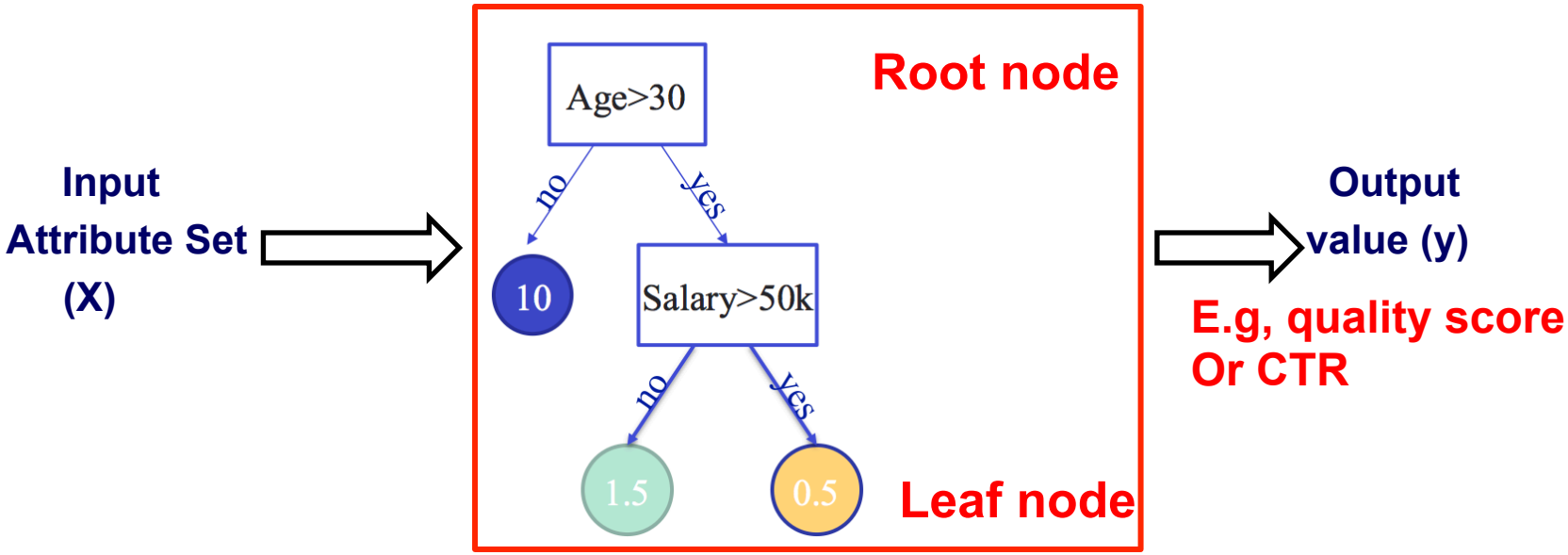
- **A decision tree represents a hierarchical segmentation of the data**
- **The original segment is called the *root node* and is the entire data set**
- **The root node is partitioned into two or more segments by applying a series of simple rules over an input variables**
  - For example, risk = low, risk = not low
  - Each rule assigns the observations to a segment based on its input value

Decision Tree Approach

- **Each resulting segment can be further partitioned into sub-segments, and so on**
  - For example risk = low can be partitioned into income = low and income = not low
- **The segments are also called *nodes*, and the final segments are called *leaf nodes* or *leaves***

Decision Trees

- A decision tree represents a hierarchical segmentation of the world/data
- DT maps observations about an item to conclusions about the item's target value.



The diagram show the classification as task of mapping an input attribute set x into its class label y

Decision Trees are very versatile

- TASKS: Regression, classification, ranking
- Feature selection
- Robust
- Easy to use; no preprocessing required
- Highly scaleable and distributed learning

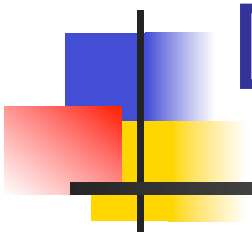
Live Session Outline

- Gradient descent versus non-gradient descent approaches
- Decision Trees
- Classification DTs on discrete variables
- Regression DTs over continuous variables [Optional]
- Ensembles (of decision Trees) [Optional]
- Practical decision trees (single core/Spark)

- **In this section we are going focus on the basics of learning a Decision Tree for classification**

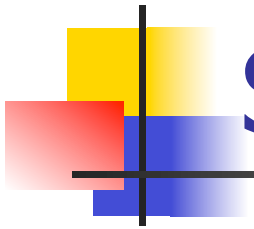
Decision Trees

- Decision tree learning learns a decision tree as a predictive model
- DT maps observations about an item to conclusions about the item's target value.
- Decision tree learning is a supervised machine learning approach whose goal is to recursively partition the world into homogenous zones, i.e., zones where the target value is homogenous (constant)



Decision Trees for Classification

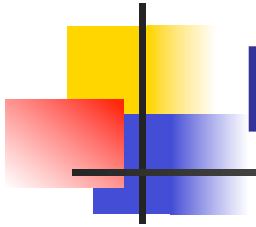
- Decision tree representation
- ID3 learning algorithm
- Entropy, information gain
- Overfitting



Supplimentary material

www

- http://dms.irb.hr/tutorial/tut_dtrees.php
- http://www.cs.uregina.ca/~dbd/cs831/notes/ml/dtrees/4_dtrees1.html

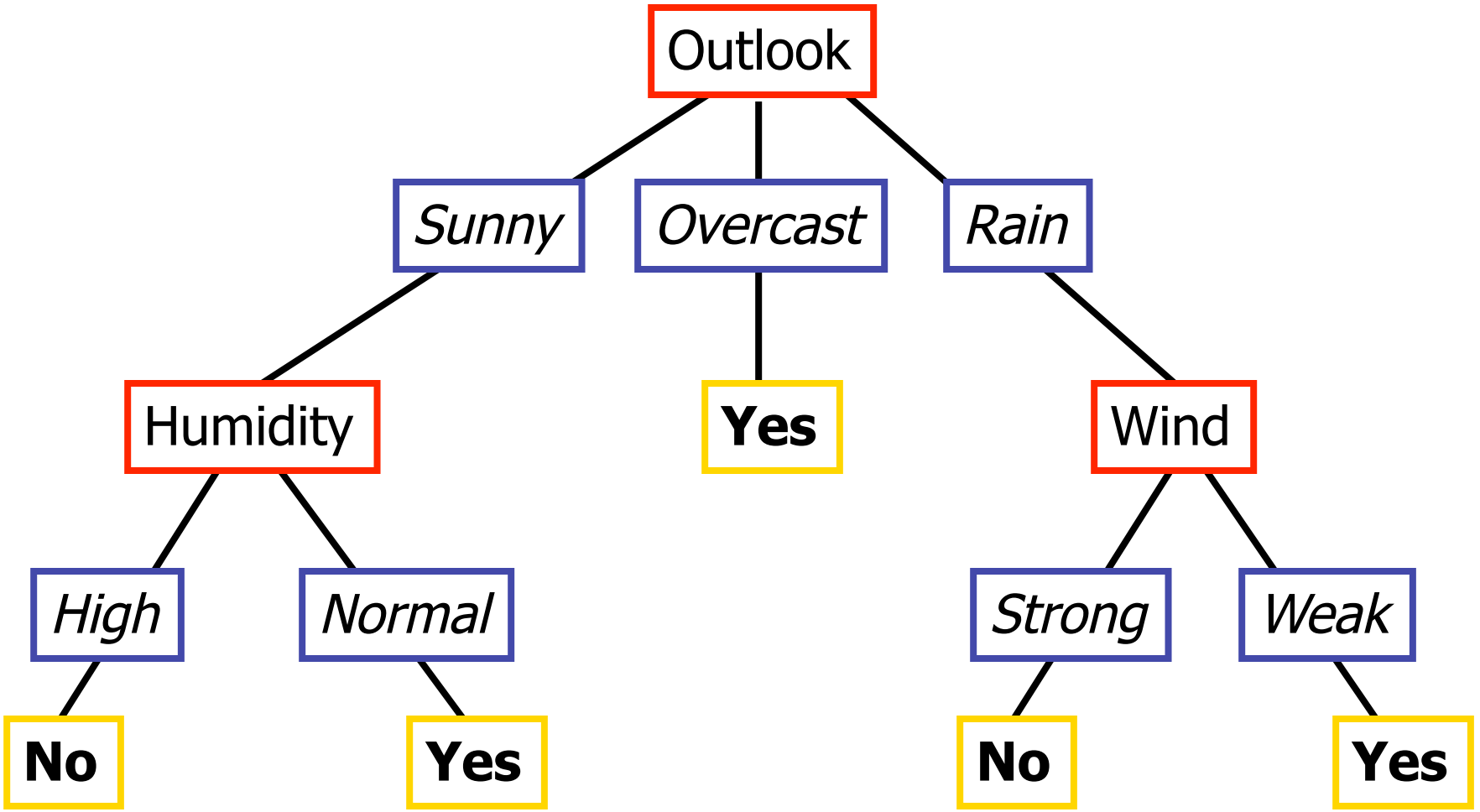


Decision Tree for PlayTennis

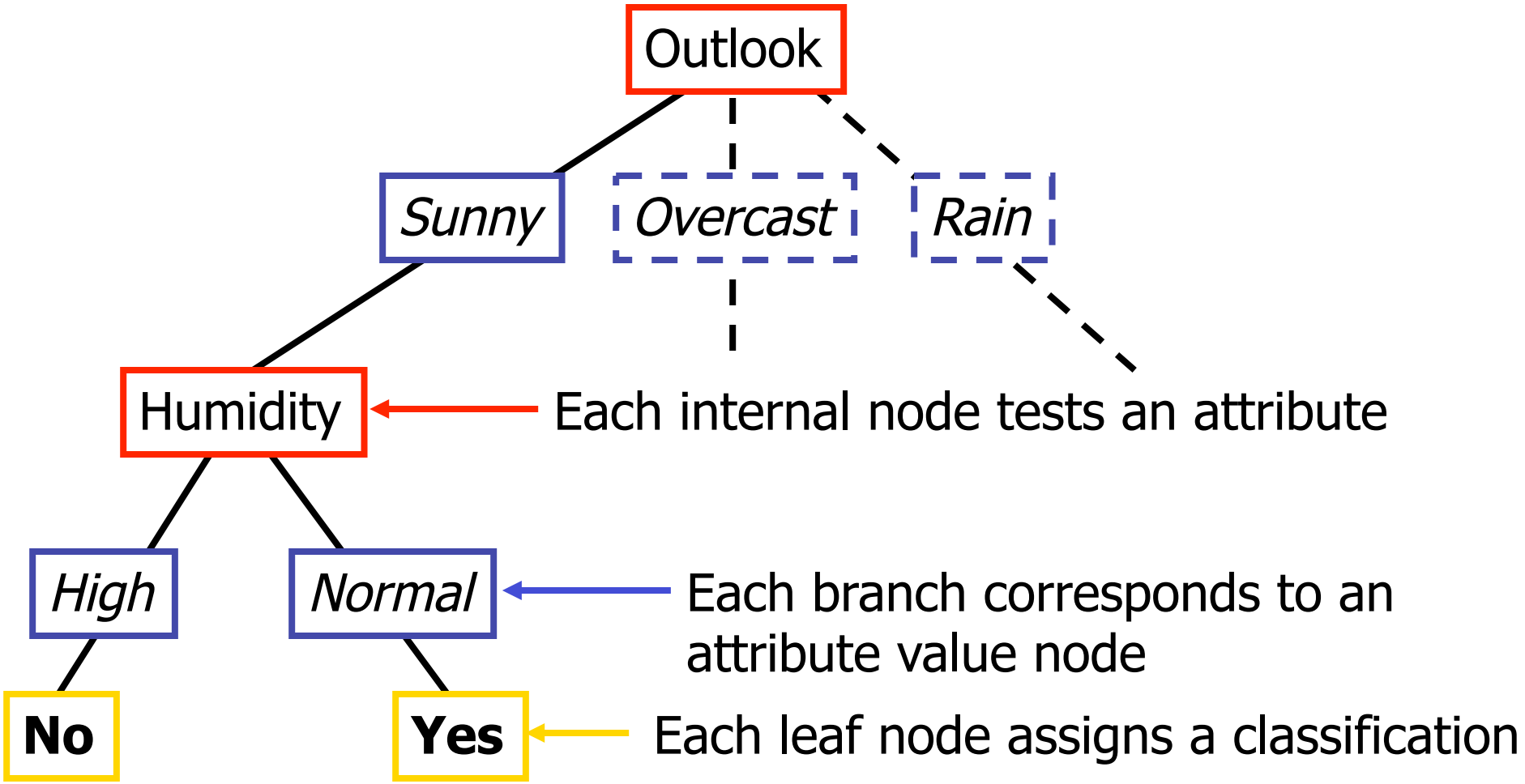
- Attributes and their values:
  - Outlook: *Sunny, Overcast, Rain*
  - Humidity: *High, Normal*
  - Wind: *Strong, Weak*
  - Temperature: *Hot, Mild, Cool*

  - Target concept - Play Tennis: *Yes, No*

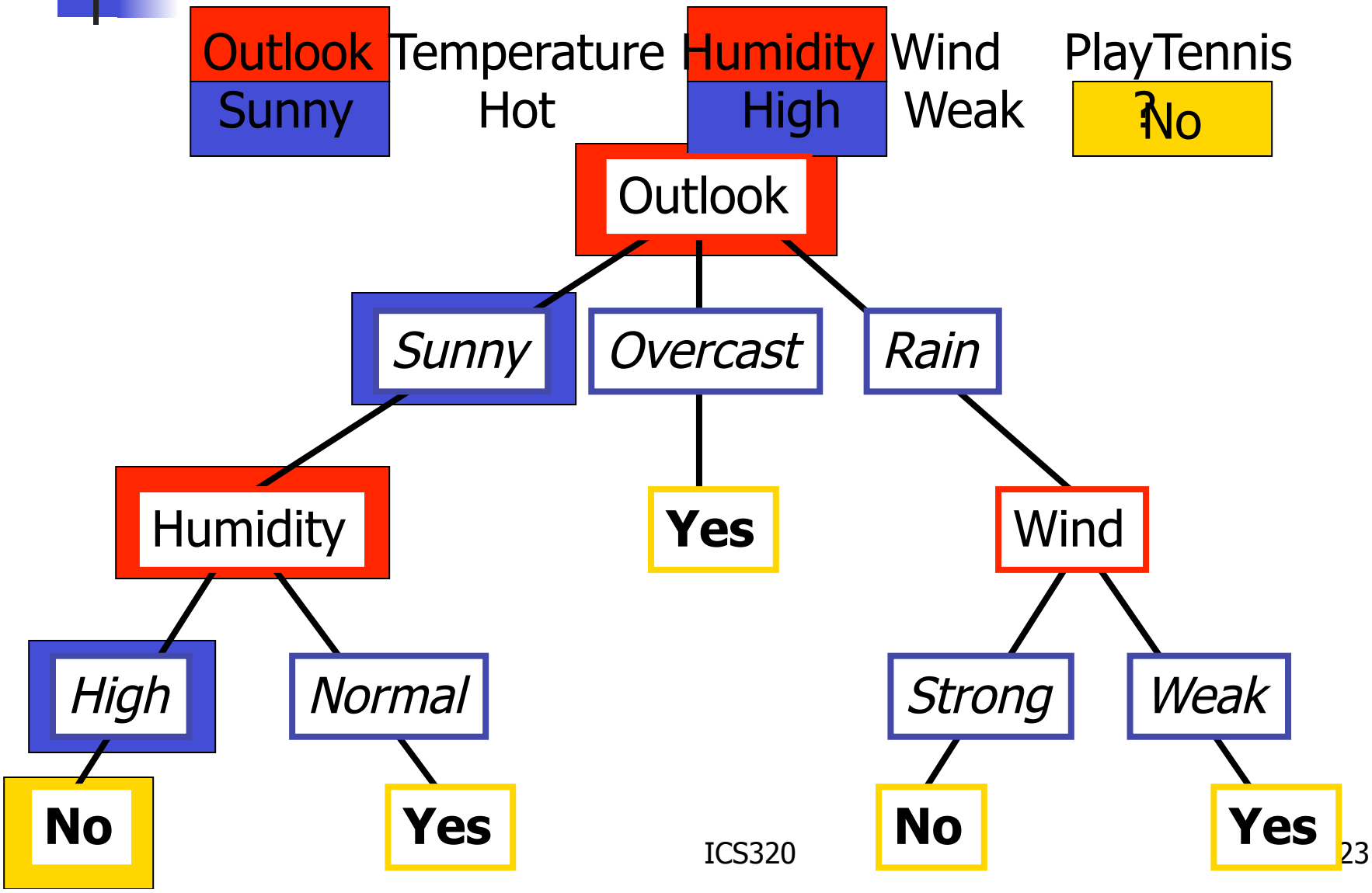
Decision Tree for PlayTennis



Decision Tree for PlayTennis

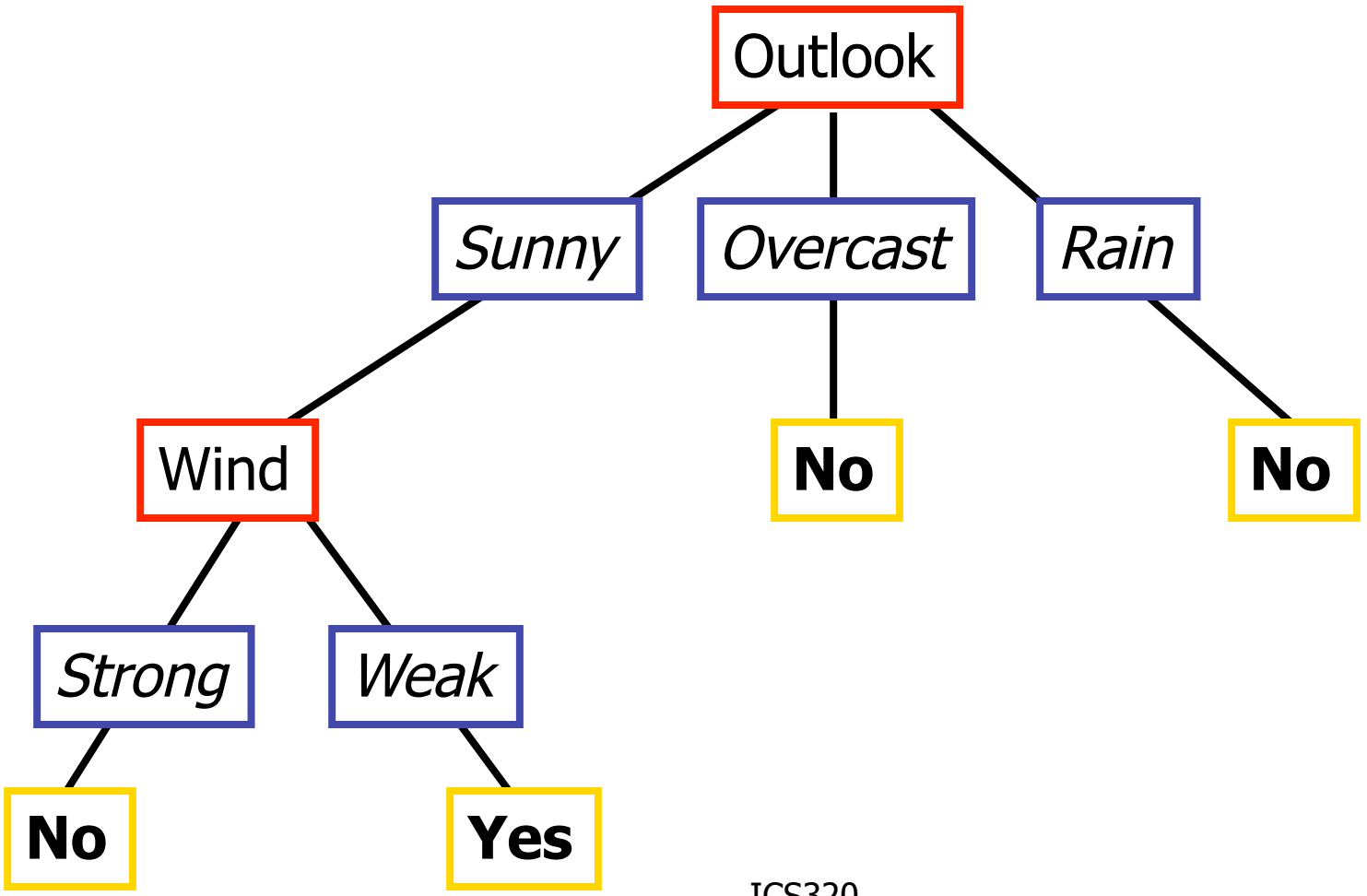


Decision Tree for PlayTennis



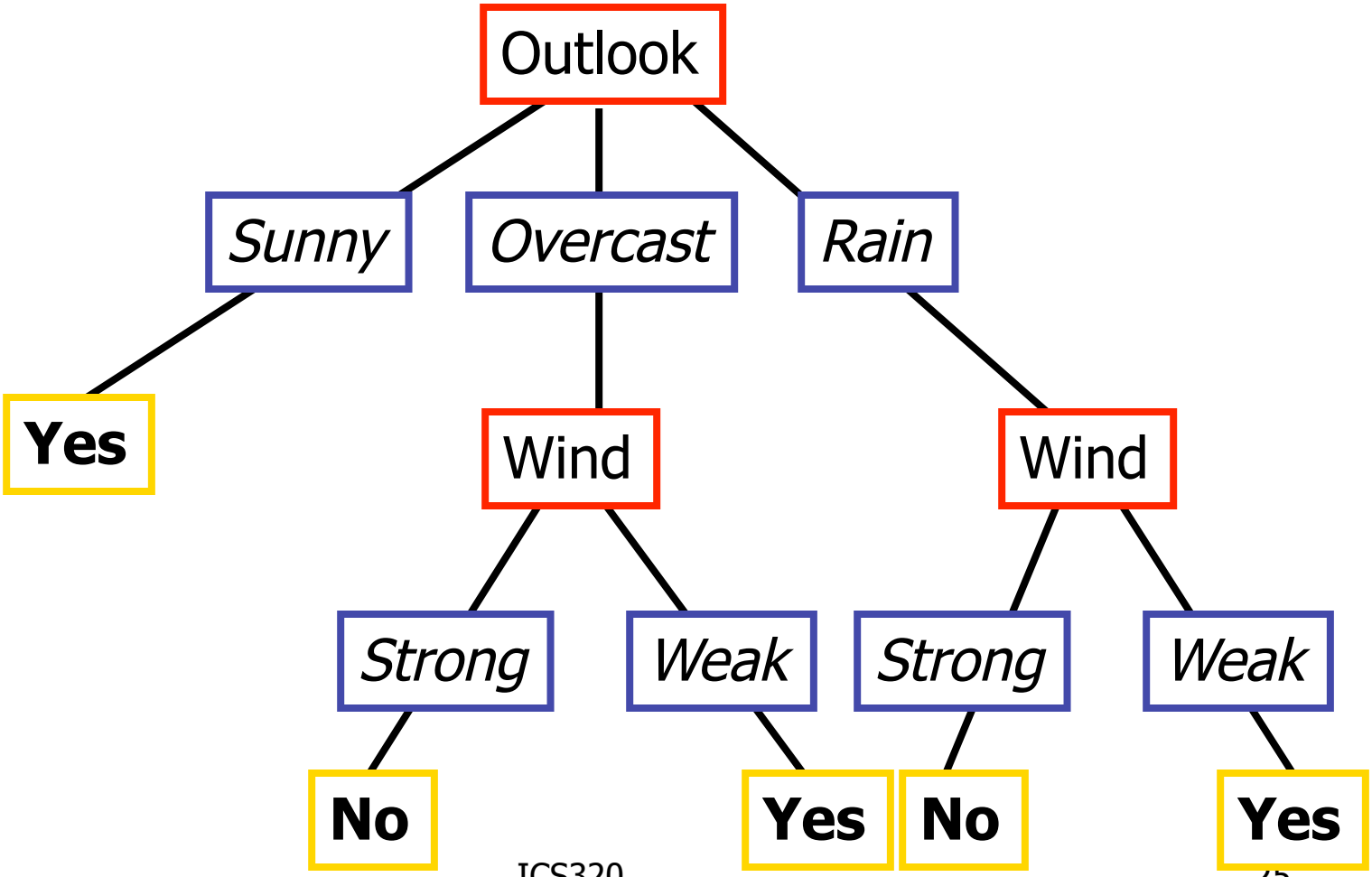
Decision Tree for Conjunction

Outlook=Sunny ∧ Wind=Weak



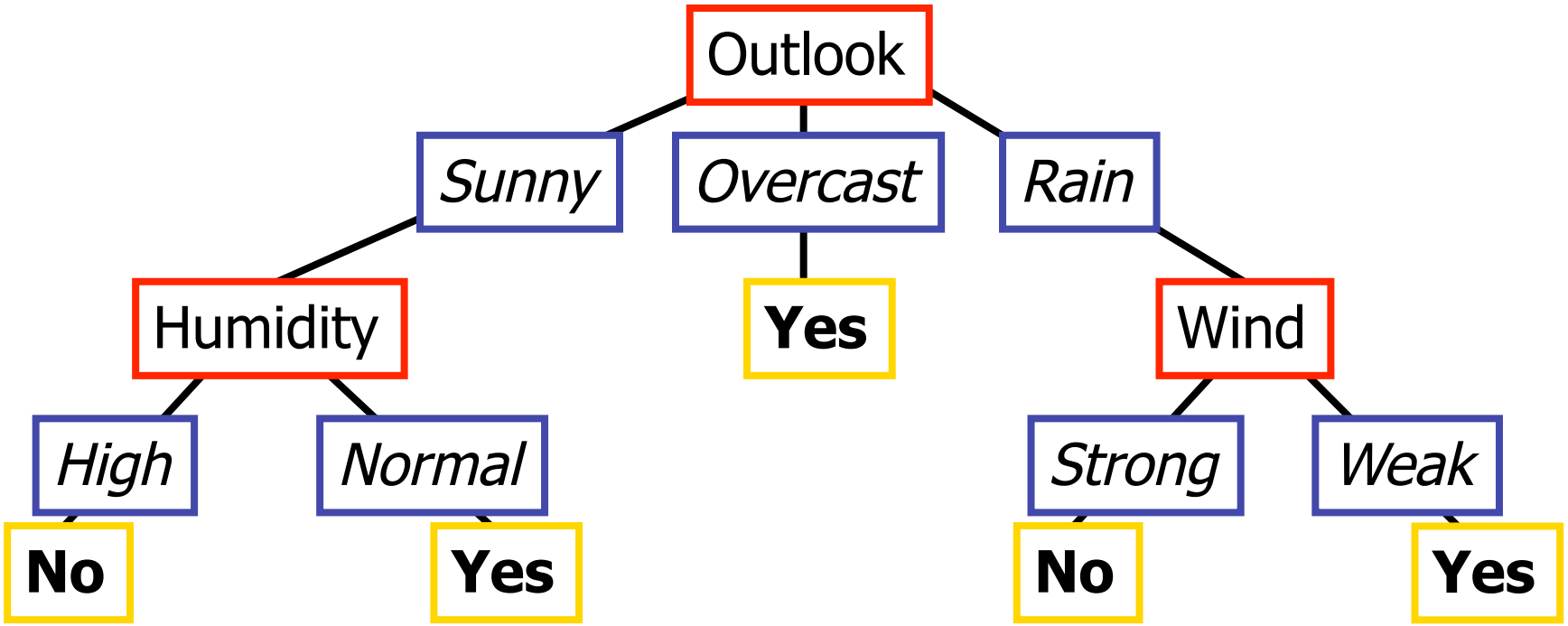
Decision Tree for Disjunction

Outlook=Sunny ∨ Wind=Weak



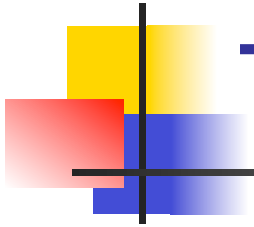
Decision Tree

- decision trees represent disjunctions of conjunctions



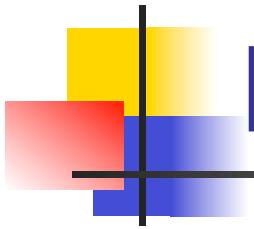
(Outlook=Sunny ∧ Humidity=Normal)
∨ (Outlook=Overcast)
∨ (Outlook=Rain ∧ Wind=Weak)

When to consider Decision Trees



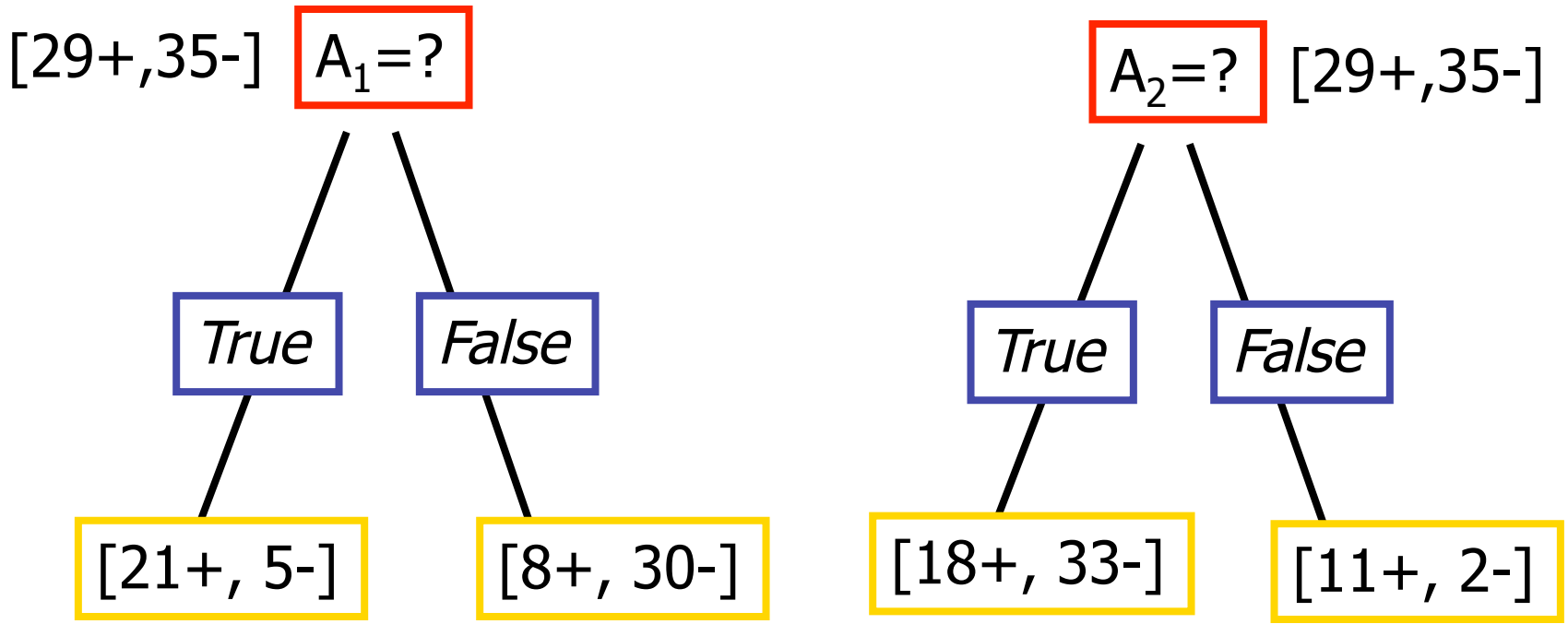
- Instances describable by attribute-value pairs
  - e.g Humidity: *High, Normal*
- Target function is discrete valued
  - e.g Play tennis; *Yes, No*
- Disjunctive hypothesis may be required
  - e.g *Outlook=Sunny* $\vee$ *Wind=Weak*
- Possibly noisy training data
- Missing attribute values
- Application Examples:
  - Medical diagnosis
  - Credit risk analysis
  - Object classification for robot manipulator (Tan 1993)

Top-Down Induction of Decision Trees ID3

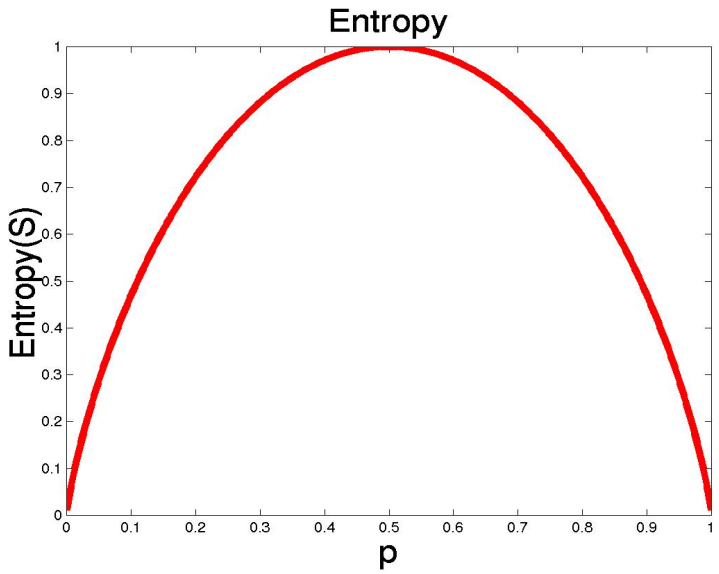


1. A ← the “best” decision attribute for next *node*
2. Assign A as decision attribute for *node*
3. For each value of A create new descendant
4. Sort training examples to leaf node according to the attribute value of the branch
5. If all training examples are perfectly classified (same value of target attribute) stop, else iterate over new leaf nodes.

Which Attribute is "best"?

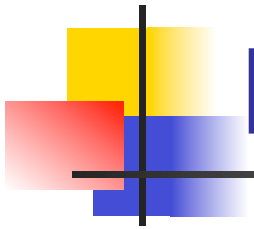


Entropy



- S is a sample of training examples
- p_+ is the proportion of positive examples
- p_- is the proportion of negative examples
- Entropy measures the impurity of S

$$\text{Entropy}(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$



Entropy

- Entropy(S)= expected number of bits needed to encode class (+ or -) of randomly drawn members of S (under the optimal, shortest length-code)

Why?

- Information theory optimal length code assign
  $-\log_2 p$ bits to messages having probability p.
- So the expected number of bits to encode (+ or -) of random member of S:

$$-p_+ \log_2 p_+ - p_- \log_2 p_-$$

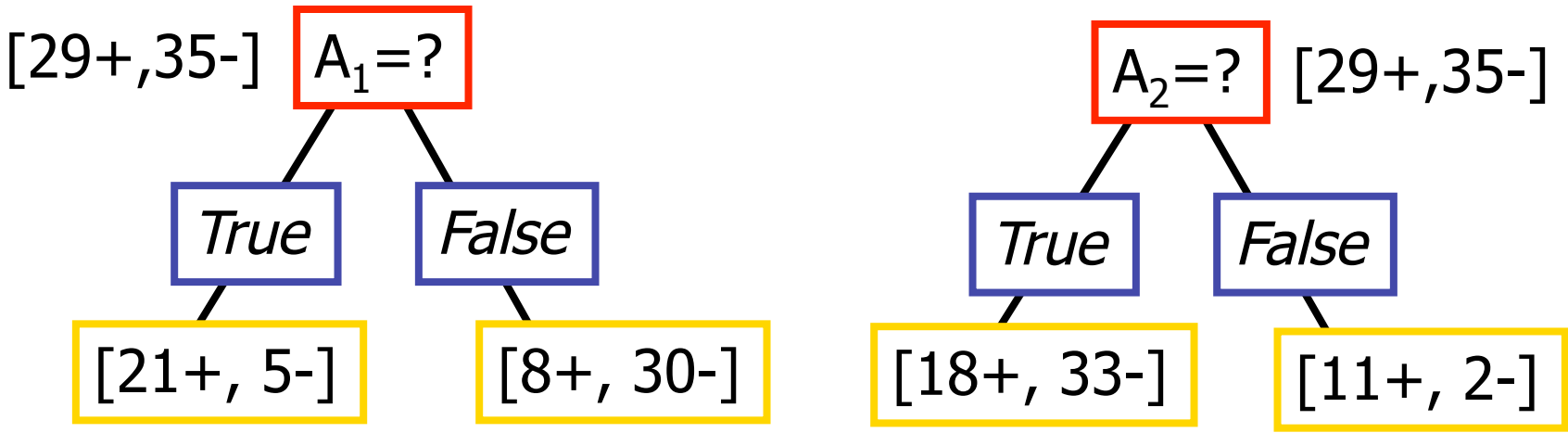
Note that: $0\text{Log}_2 0 = 0$

Information Gain

- Gain(S,A): expected reduction in entropy due to sorting S on attribute A

$$Gain(S,A)=Entropy(S) - \sum_{v \in values(A)} |S_v|/|S| \; Entropy(S_v)$$

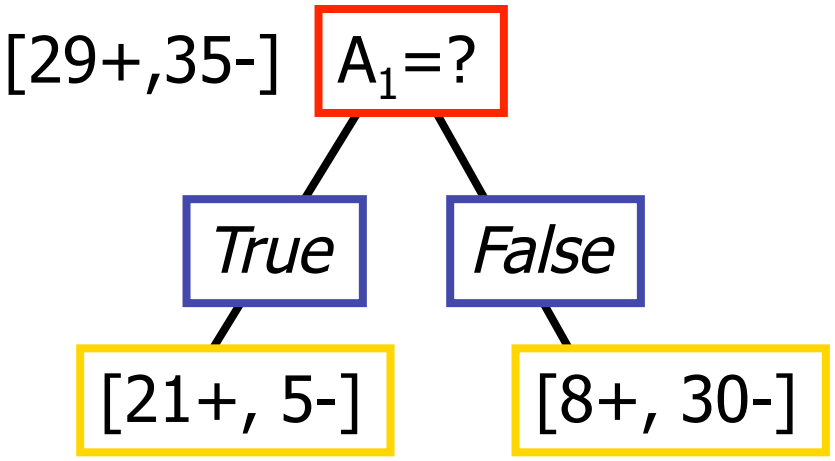
$$Entropy([29+,35-]) = -29/64 \log_2 29/64 - 35/64 \log_2 35/64 = 0.99$$

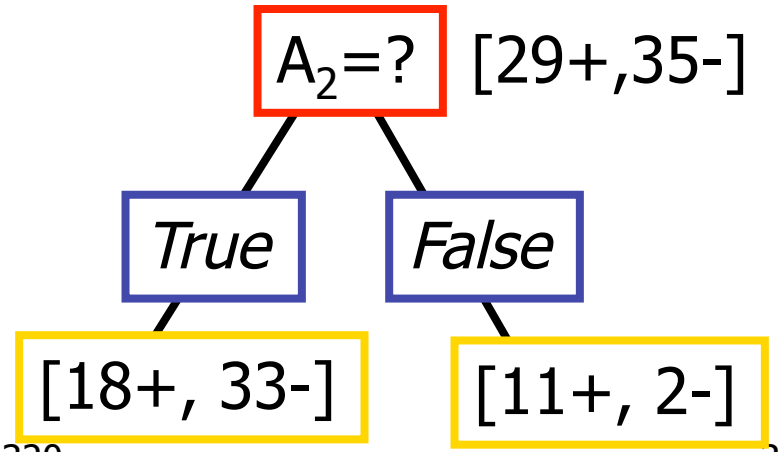


Information Gain

Entropy([21+,5-]) = 0.71
Entropy([8+,30-]) = 0.74
Gain(S,A_1)=Entropy(S)
-26/64*Entropy([21+,5-])
-38/64*Entropy([8+,30-])
=0.27

Entropy([18+,33-]) = 0.94
Entropy([8+,30-]) = 0.62
Gain(S,A_2)=Entropy(S)
-51/64*Entropy([18+,33-])
-13/64*Entropy([11+,2-])
=0.12

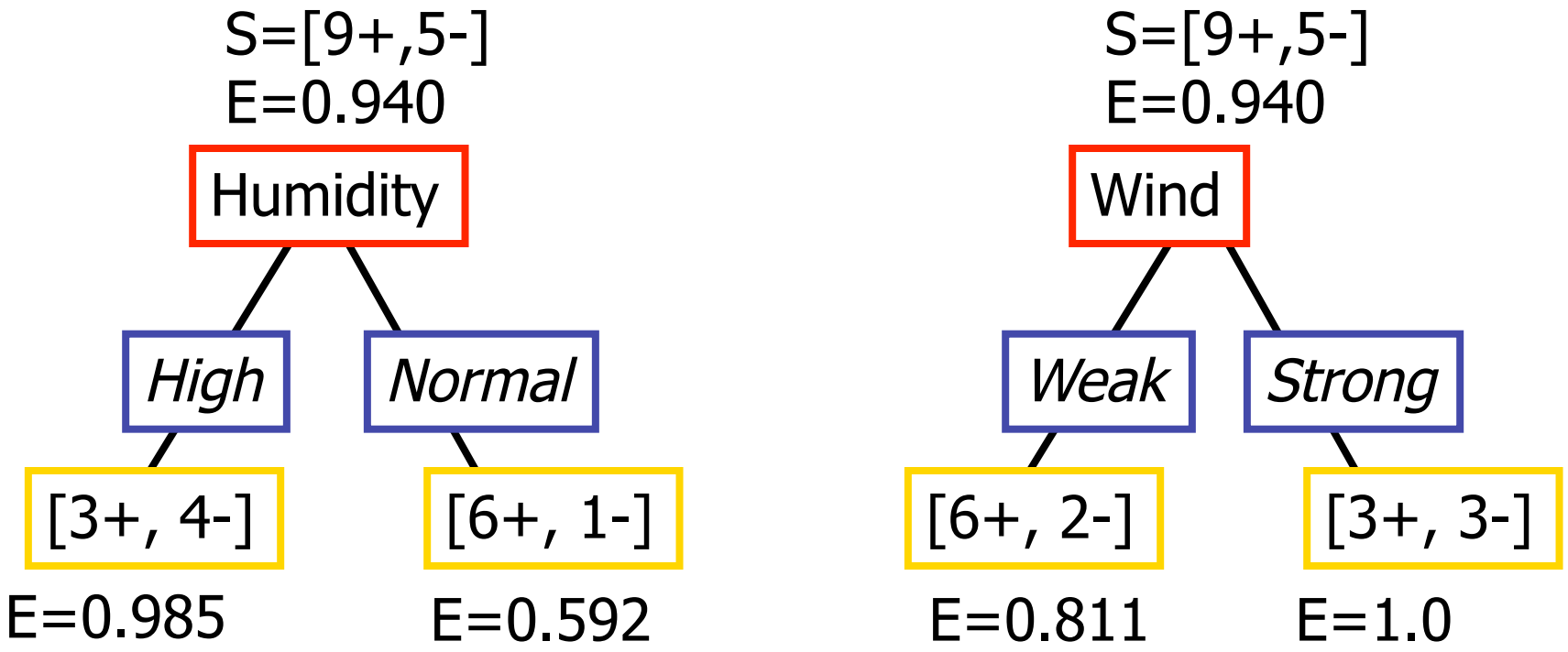




Training Examples

Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Selecting the Next Attribute

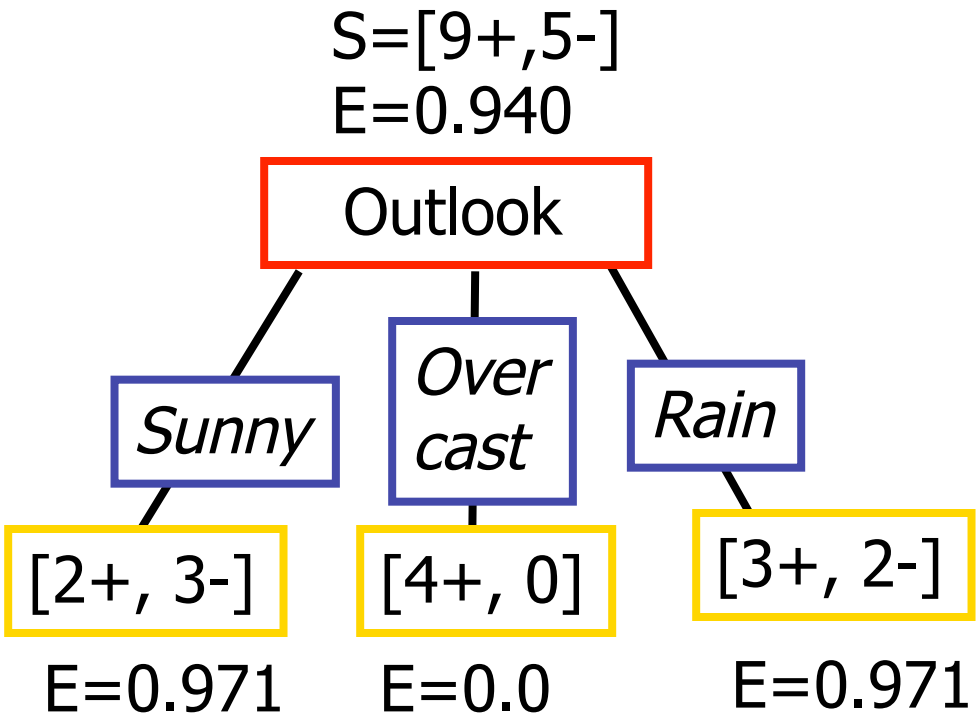


Gain(S,Humidity)
=0.940-(7/14)*0.985
– (7/14)*0.592
=0.151

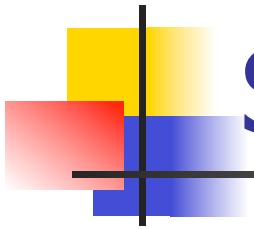
Gain(S,Wind)
=0.940-(8/14)*0.811
– (6/14)*1.0
=0.048

Humidity provides greater info. gain than Wind, w.r.t target classification.

Selecting the Next Attribute



Gain(S,Outlook)
=0.940-(5/14)*0.971
-(4/14)*0.0 – (5/14)*0.0971
=0.247



Selecting the Next Attribute

The information gain values for the 4 attributes are:

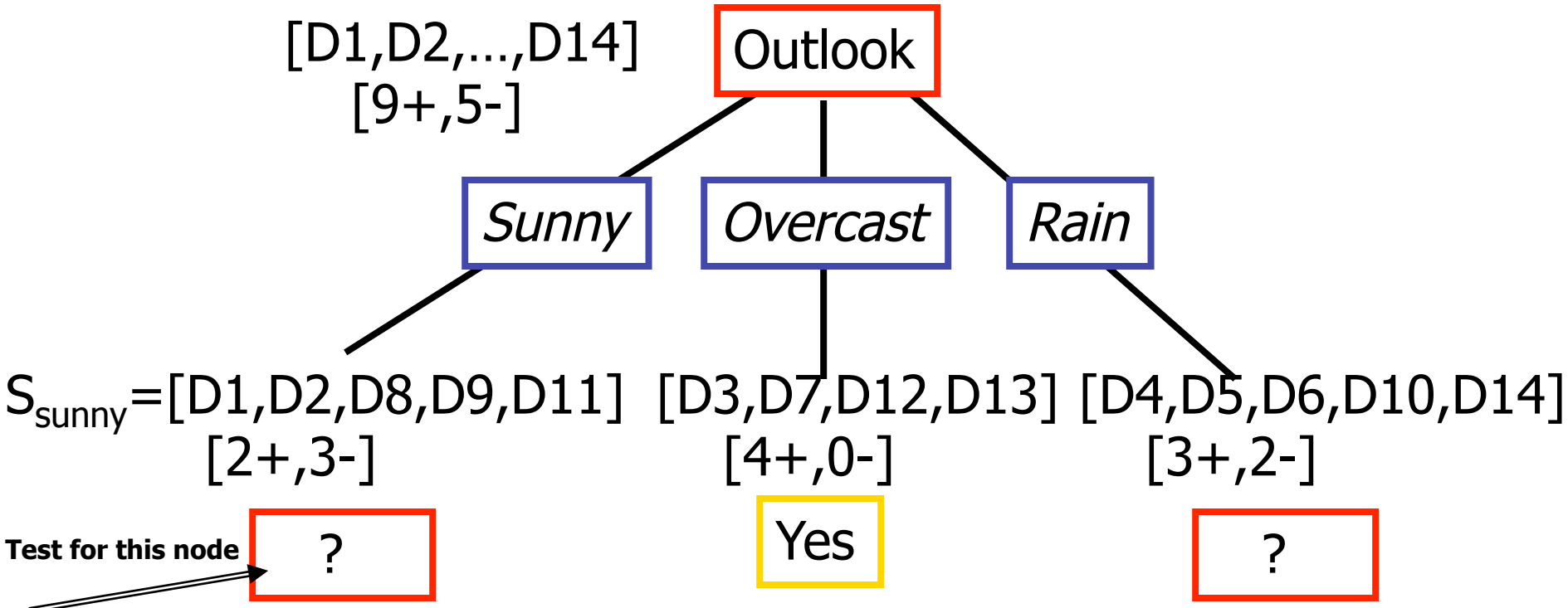
- Gain(S,Outlook) =0.247
- Gain(S,Humidity) =0.151
- Gain(S,Wind) =0.048
- Gain(S,Temperature) =0.029

where S denotes the collection of training examples

Note: $0\text{Log}_2 0 = 0$

ID3 Algorithm

Note: $0\text{Log}_2 0 = 0$

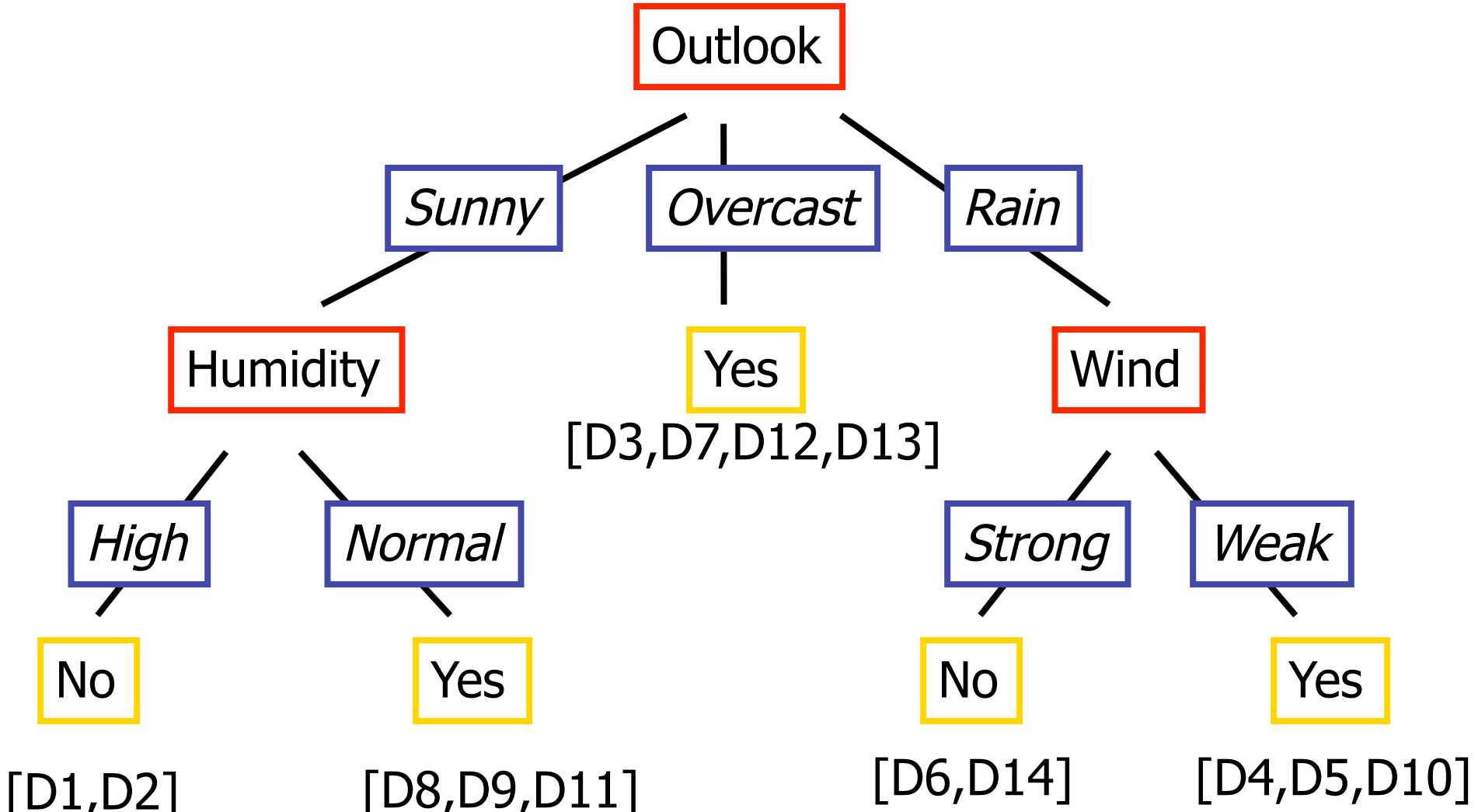


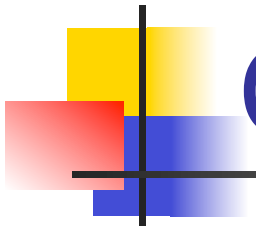
$\text{Gain}(S_{sunny}, \text{Humidity}) = 0.970 - (3/5)0.0 - 2/5(0.0) = 0.970$

$\text{Gain}(S_{sunny}, \text{Temp.}) = 0.970 - (2/5)0.0 - 2/5(1.0) - (1/5)0.0 = 0.570$

$\text{Gain}(S_{sunny}, \text{Wind}) = 0.970 = -(2/5)1.0 - 3/5(0.918) = 0.019$

ID3 Algorithm





Occam's Razor

Why prefer short hypotheses?

Argument in favor:

- Fewer short hypotheses than long hypotheses
- A short hypothesis that fits the data is unlikely to be a coincidence
- A long hypothesis that fits the data might be a coincidence

Argument opposed:

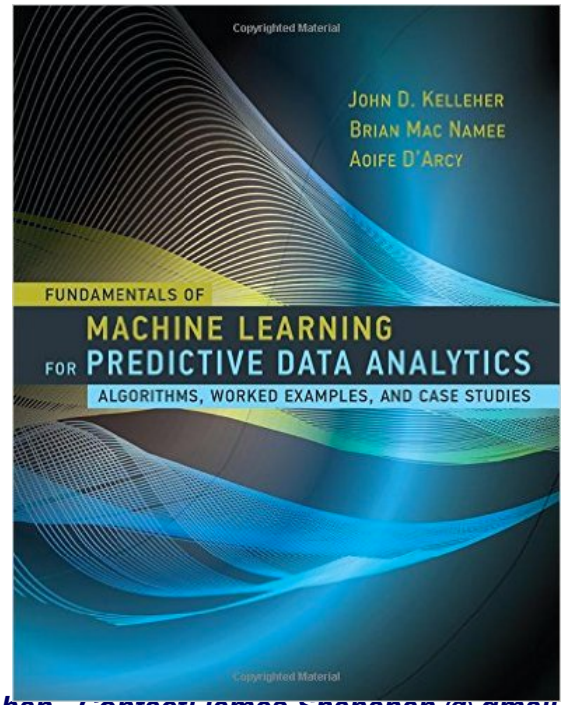
- There are many ways to define small sets of hypotheses
- E.g. All trees with a prime number of nodes that use attributes beginning with "Z"
- What is so special about small sets based on *size* of hypothesis

Entropy and ID3

- **Entropy, and ID3 Decision Tree Learning algorithm**
  - https://www.dropbox.com/s/l0pp4ed5wwv9s1h/FMLPDA_SampleChapter_InformationBasedLearning-DT-Excellent-Examples.pdf?dl=0

https://www.dropbox.com/s/l0pp4ed5wwv9s1h/FMLPDA_SampleChapter_InformationBasedLearning-DT-Excellent-Examples.pdf?dl=0

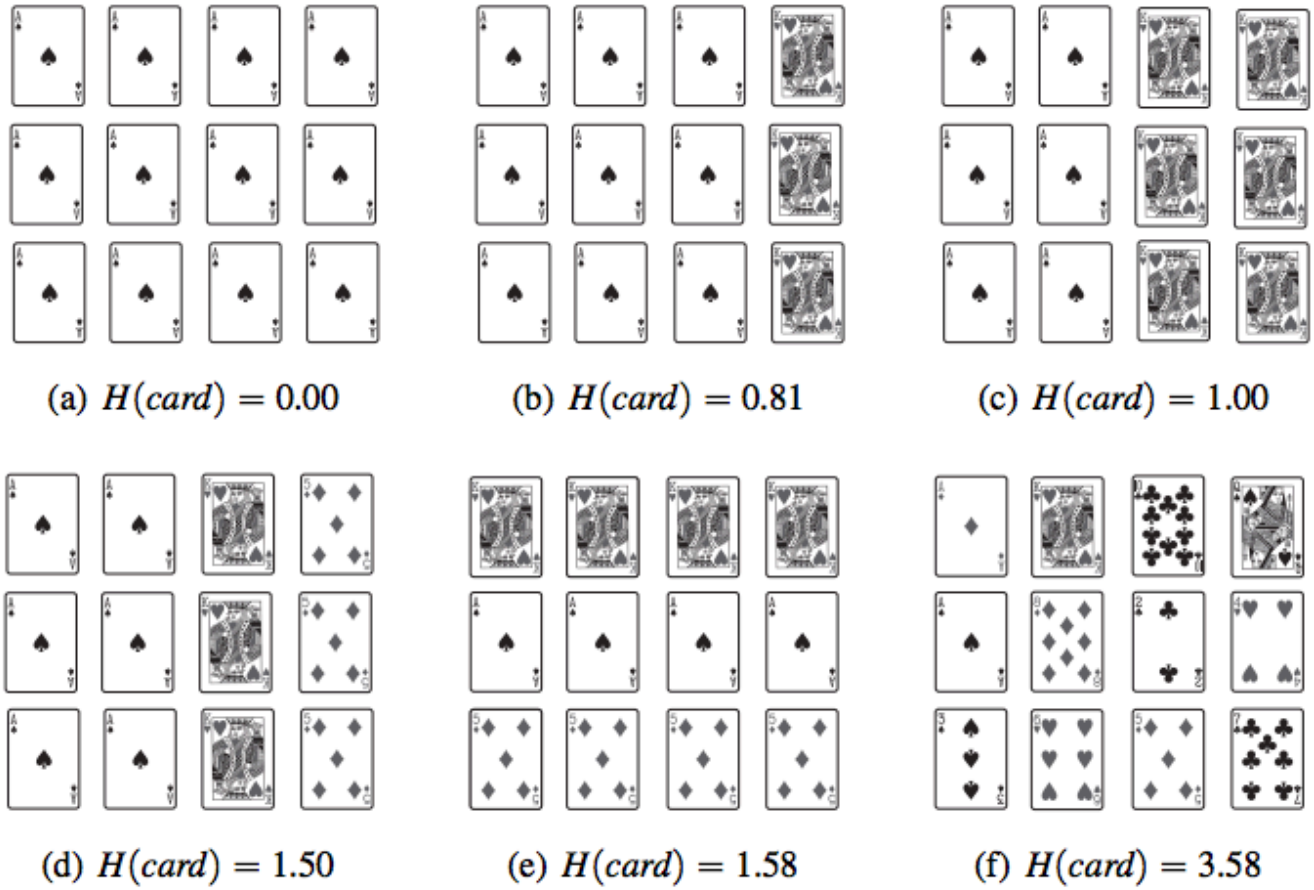
http://machinelearningbook.com/



Information Theory

- Claude Shannon's entropy model defines a computational measure of the impurity of the elements in a set.
- Entropy, impurity, heterogeneity

Shannon's Entropy Model



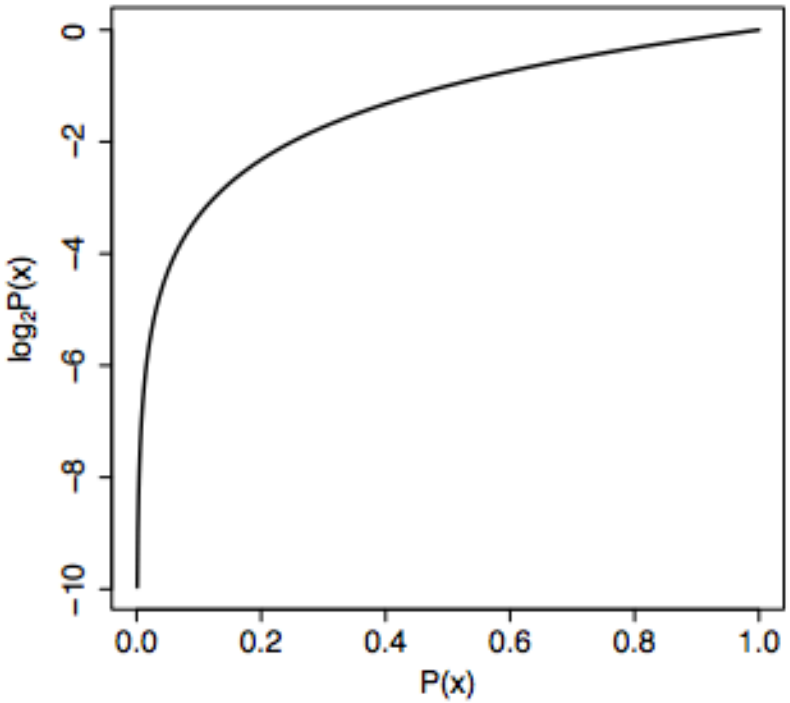
(a) $H(card) = 0.00$ (b) $H(card) = 0.81$ (c) $H(card) = 1.00$

(d) $H(card) = 1.50$ (e) $H(card) = 1.58$ (f) $H(card) = 3.58$

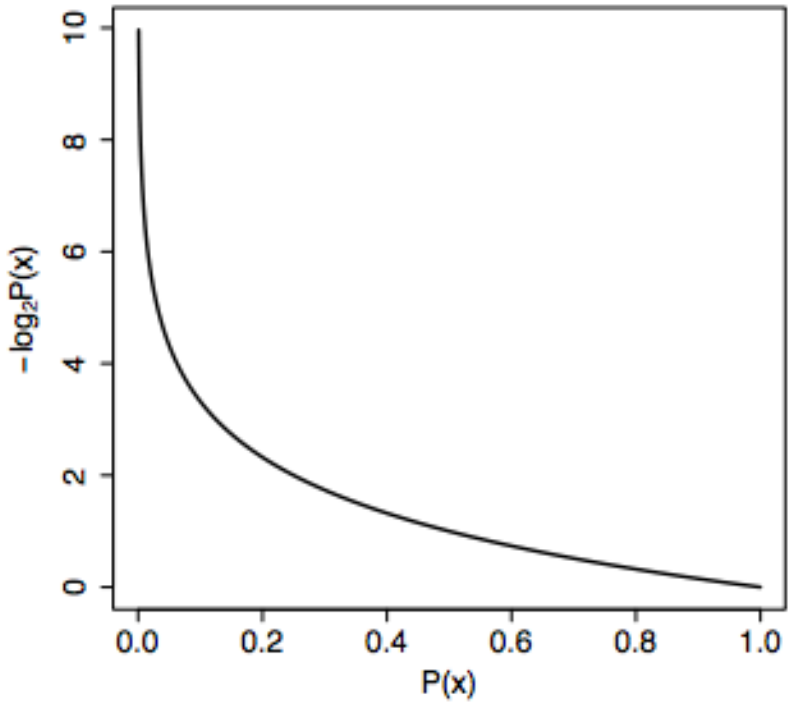
Figure 4.5

The entropy of different sets of playing cards measured in bits.

http://machinelearningbook.com/wp-content/uploads/2015/07/FMLPDA_SampleChapter_InformationBasedLearning.pdf



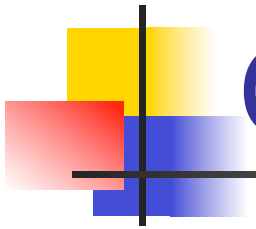
(a) $P(x)$ and $log_2 P(x)$



(b) $P(x)$ and $-log_2 P(x)$

Figure 4.6

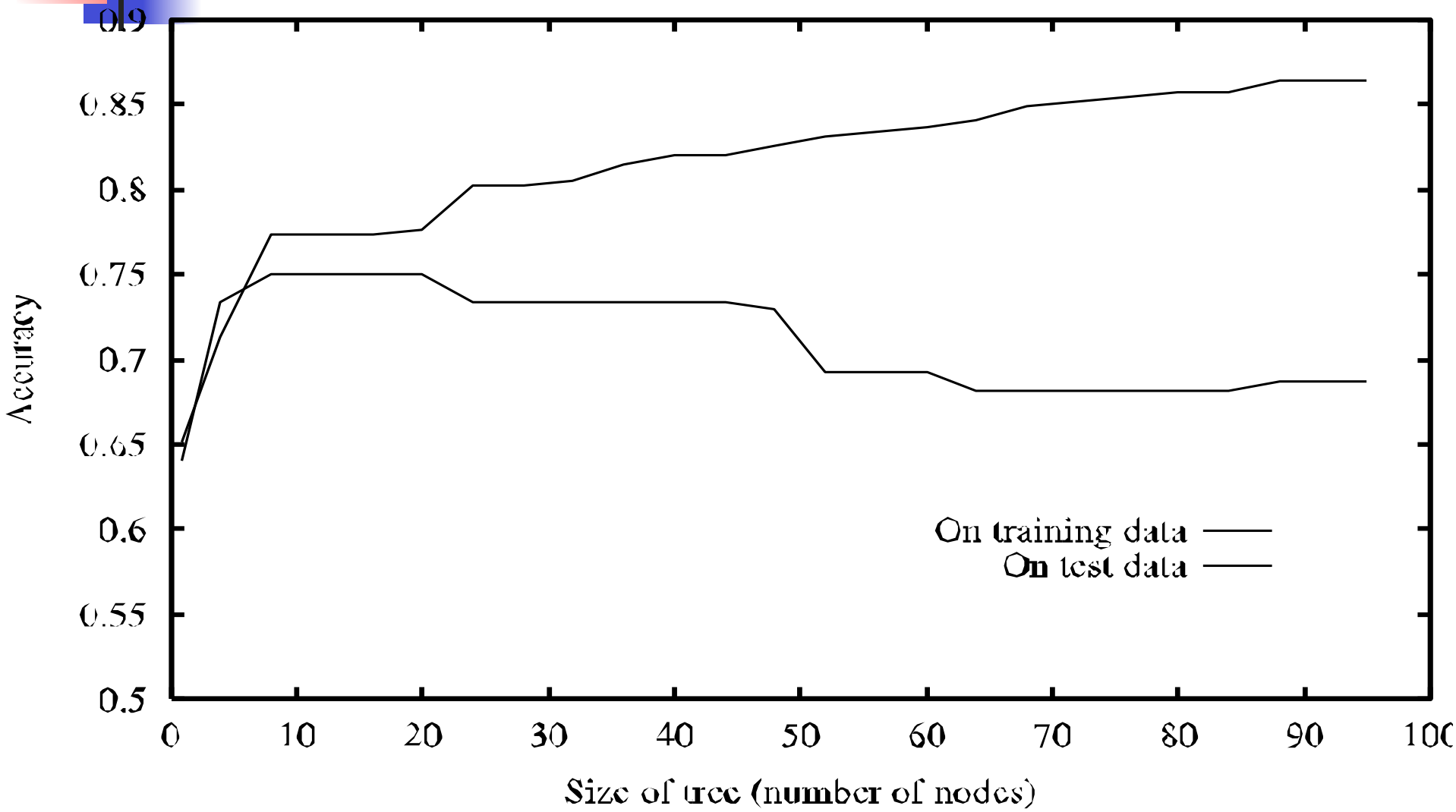
(a) A graph illustrating how the value of a binary log (the log to the base 2) of a probability changes across the range of probability values; (b) the impact of multiplying these values by -1.

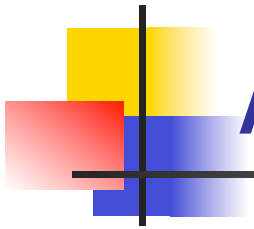


Overfitting

- One of the biggest problems with decision trees is **Overfitting**

Overfitting in Decision Tree Learning





Avoid Overfitting

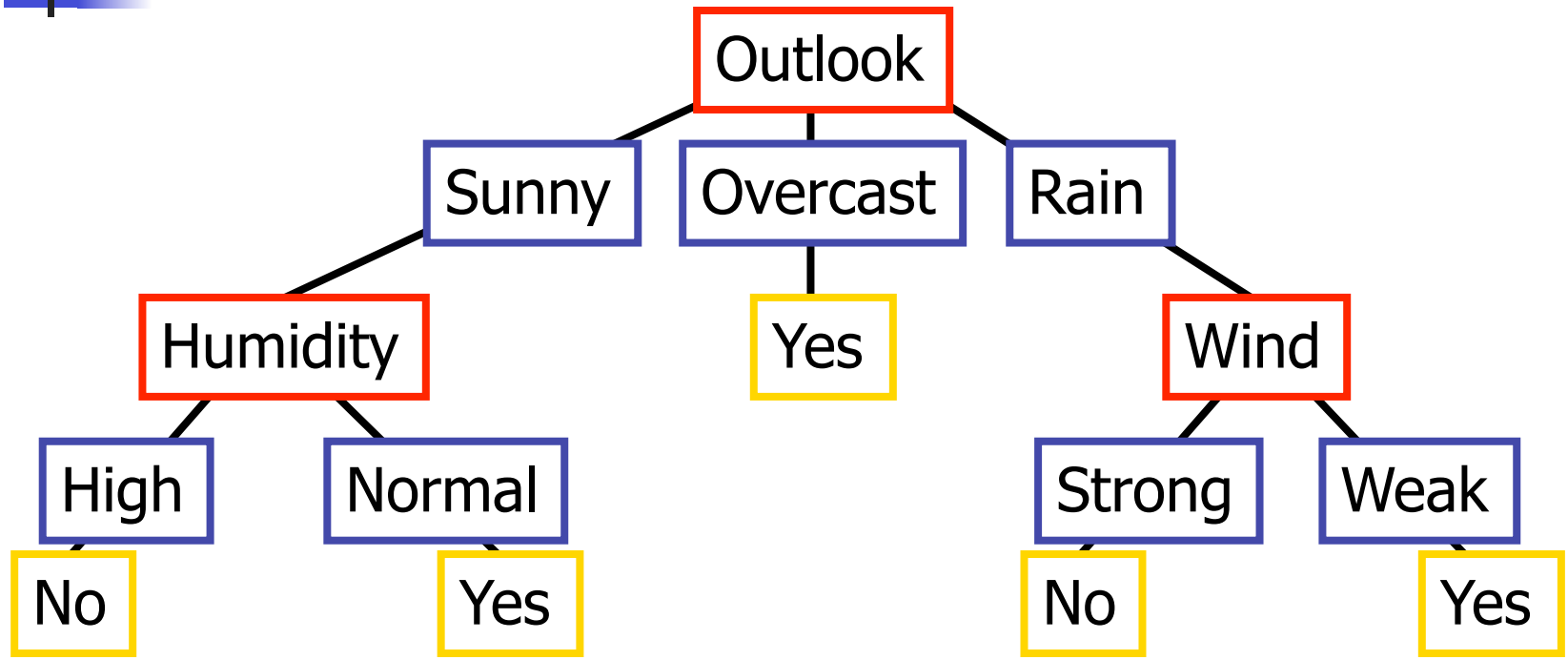
How can we avoid overfitting?

- Stop growing when data split not statistically significant
- Grow full tree then post-prune
- Minimum description length (MDL):

  Minimize:

  size(tree) + size(misclassifications(tree))

Converting a Tree to Rules



R_1: If (Outlook=Sunny) ∧ (Humidity=High) Then PlayTennis=No
R_2: If (Outlook=Sunny) ∧ (Humidity=Normal) Then PlayTennis=Yes
R_3: If (Outlook=Overcast) Then PlayTennis=Yes
R_4: If (Outlook=Rain) ∧ (Wind=Strong) Then PlayTennis=No
R_5: If (Outlook=Rain) ∧ (Wind=Weak) Then PlayTennis=Yes

Continuous Valued Attributes

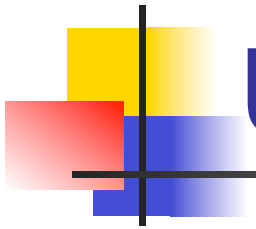
Create a discrete attribute to test continuous

- Temperature = 24.5^{0}C
- (Temperature > 20.0^{0}C) = {true, false}

Where to set the threshold?

Temperature	15^{0}C	18^{0}C	19^{0}C	22^{0}C	24^{0}C	27^{0}C
PlayTennis	No	No	Yes	Yes	Yes	No

(see paper by [Fayyad, Irani 1993]



Unknown Attribute Values

What if some examples have missing values of A?

Use training example anyway sort through tree

- If node n tests A, assign most common value of A among other examples sorted to node n.
- Assign most common value of A among other examples with same target value
- Assign probability pi to each possible value vi of A
  - Assign fraction pi of example to each descendant in tree

Classify new examples in the same fashion

What makes DTs so attractive?

Decision Trees

- Decision tree learning learns a decision tree as a predictive model
- DT maps observations about an item to conclusions about the item's target value.
- Decision tree learning is a supervised machine learning approach whose goal is to recursively partition the world into homogenous zones, i.e., zones where the target value is homogenous (constant)

Classification versus regression Trees

- Tree models where the target variable can take a finite set of values are called classification trees.
- In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

- Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

Dtrees are desirable: plug and play

- One of the most popular approaches to ML in practice
- Can handle numeric, categorical, nominal features
- No preprocessing required, no standarization
- Handles Missing values naturally and Nas does not affect performance metrics
- Interaction features
- Highly Scaleable
- Variable selection
- Excellent performance on a variety of problems
- Off the shelf with very few hyperparameters

Dtrees are desirable: plug and play

1. *Ability to deal with irrelevant inputs.* Since at every node, we scan all the variables and pick the best, trees naturally do variable selection. And, thus, anything you can measure, you can allow as a candidate without worrying that they will unduly skew your results.

   Trees also provide a variable importance score based on the contribution to error (risk) reduction across all the splits in the tree (see Chapter 5).

2. *No data preprocessing needed.* Trees naturally handle numeric, binary, and categorical variables.

   Numeric attributes have splits of the form $x_j < cut\_value$; categorical attributes have splits of the form $x_j \in \{value1,\ value2, \ldots\}$.

   Monotonic transformations won't affect the splits, so you don't have problems with input outliers. If $cut\_value = 3$ and a value x_j is 3.14 or 3,100, it's greater than 3, so it goes to the same side. Output outliers can still be influential, especially with squared-error as the loss.

3. *Scalable computation.* Trees are very fast to build and run compared to other iterative techniques. Building a tree has approximate time complexity of $O\ (nN \log N)$.

4. *Missing value tolerant.* Trees do not suffer much loss of accuracy due to missing values.

   Some tree algorithms treat missing values as a separate categorical value. CART handles them via a clever mechanism termed "surrogate" splits (Breiman et al., 1993); these are substitute splits in case the first variable is unknown, which are selected based on their ability to approximate the splitting of the originally intended variable.

   One may alternatively create a new binary variable $x_j\_is\_NA$ (not available) when one believes that there may be information in x_j's being missing – i.e., that it may not be "missing at random."

5. *"Off-the-shelf" procedure: there are only few tunable parameters.* One can typically use them within minutes of learning about them.

Live Session Outline

- Gradient descent versus non-gradient descent approaches
- Decision Trees
- Classification DTs on discrete variables
- Regression DTs over continuous variables [Optional]
- Ensembles (of decision Trees) [Optional]
- Practical decision trees (single core/Spark)

- **Regression Trees**

Decision Tree Approach

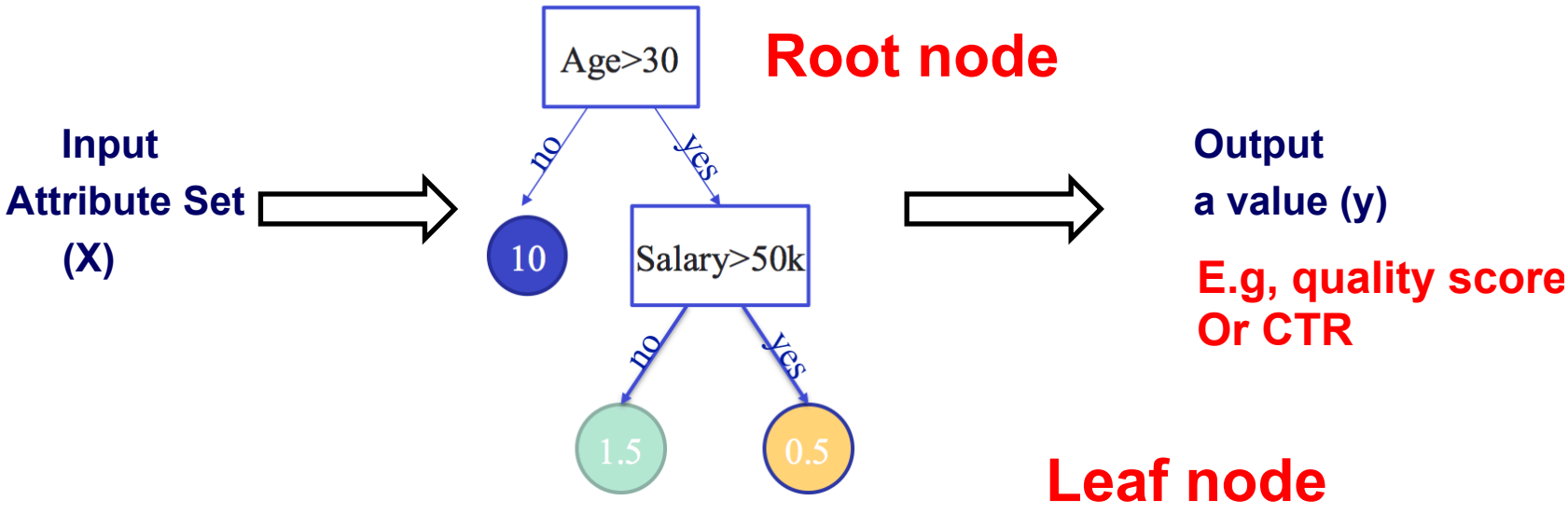
- **A decision tree represents a hierarchical segmentation of the data**
- **The original segment is called the *root node* and is the entire data set**
- **The root node is partitioned into two or more segments by applying a series of simple rules over an input variables**
  - For example, risk = low, risk = not low
  - Each rule assigns the observations to a segment based on its input value

Decision Tree Approach

- **Each resulting segment can be further partitioned into sub-segments, and so on**
  - For example risk = low can be partitioned into income = low and income = not low
- **The segments are also called *nodes*, and the final segments are called *leaf nodes* or *leaves***

Decision Trees

- A decision tree represents a hierarchical segmentation of the world/data
- DT maps observations about an item to conclusions about the item's target value.



The diagram show the classification as task of mapping an input attribute set x into its class label y

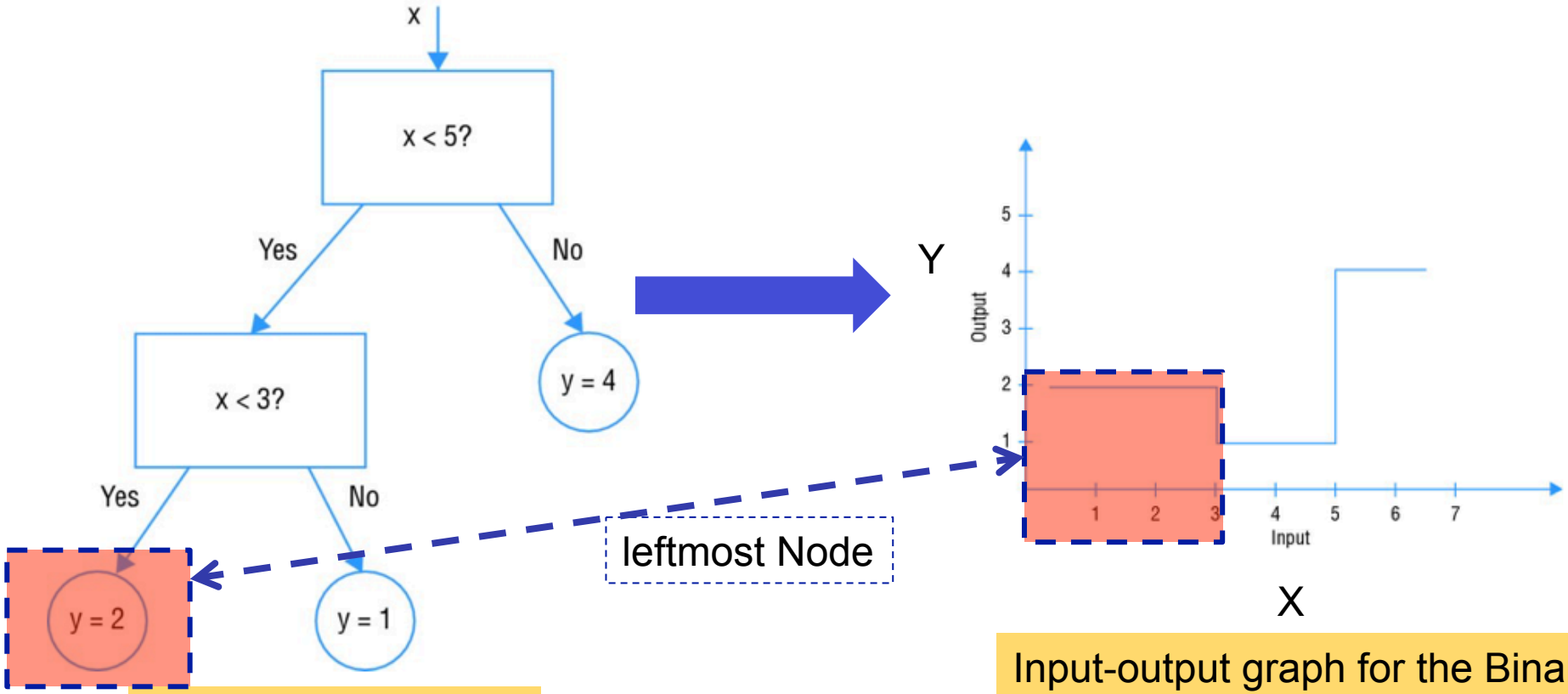
Question on DT

- **DTs for regression? For classification?**
  - Yes and yes
- **Better at regression than classification?**
  - Equally good, especially in ensemble mode

Binary Decision Tree

Slide improvisation

- Classification versus regression



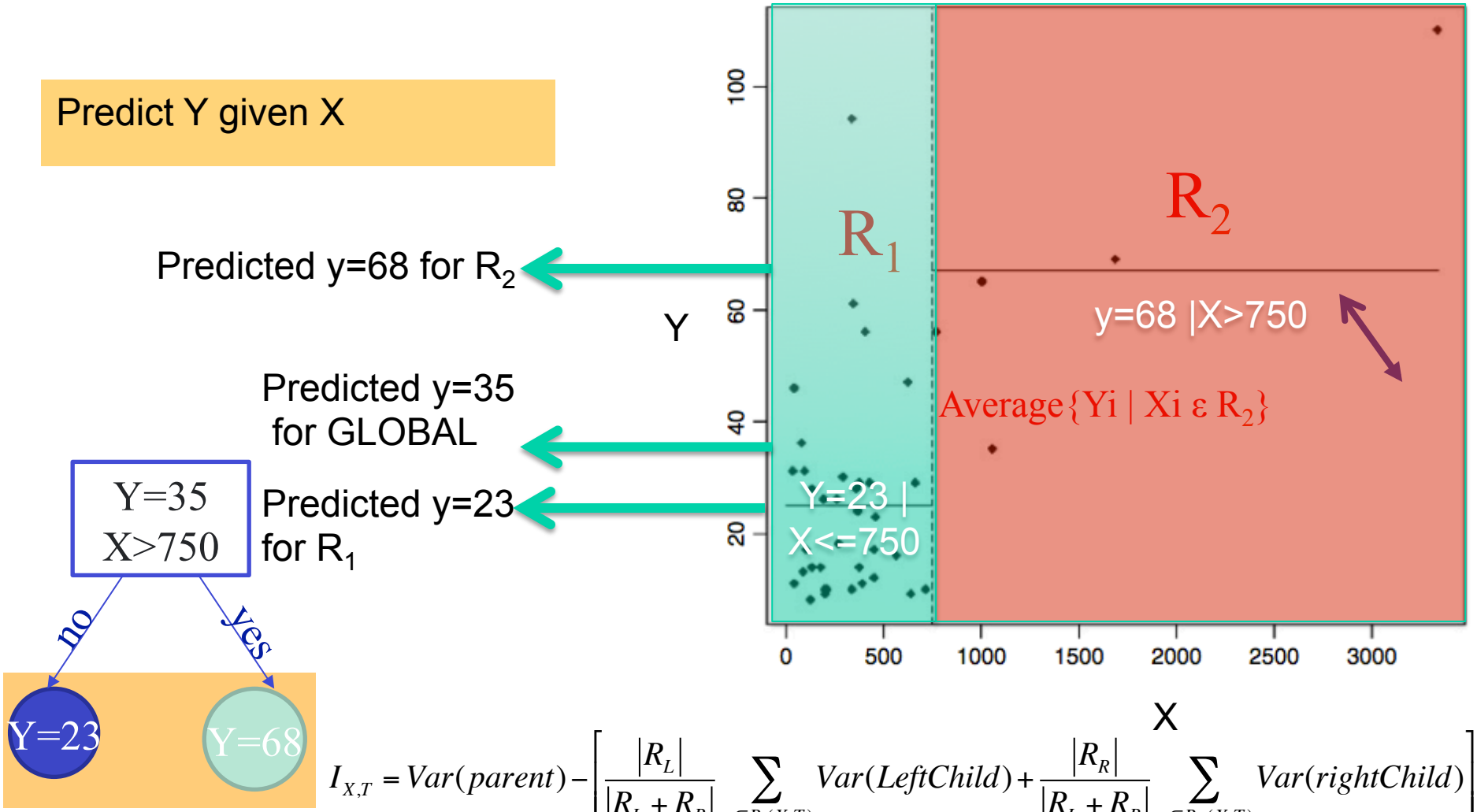
Binary Decision Tree

Input-output graph for the Binary Decision Tree example

Mean value prediction for regression trees with squared error loss

- **Partition the space into *M* regions: R_1, R_2**

Predict Y given X

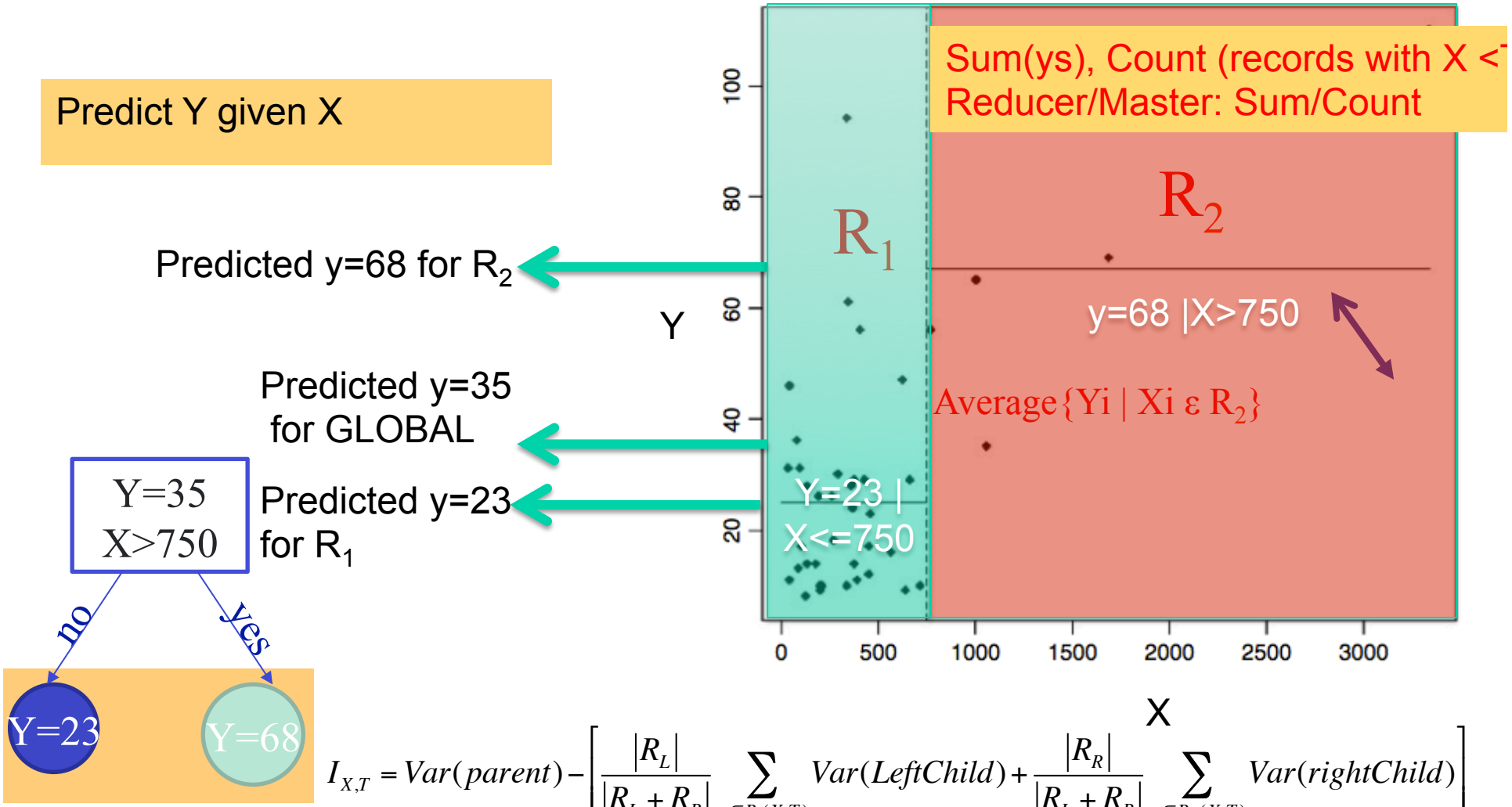


$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} Var(rightChild) \right]$$

- **How can we do this at scale?**

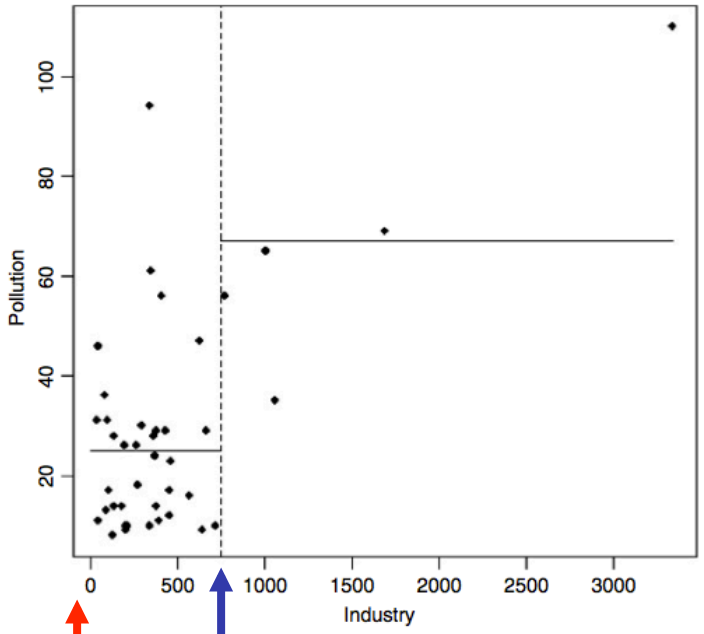
Mean value prediction for regression trees with squared error loss

- **Partition the space into *M* regions: R_1, R_2**



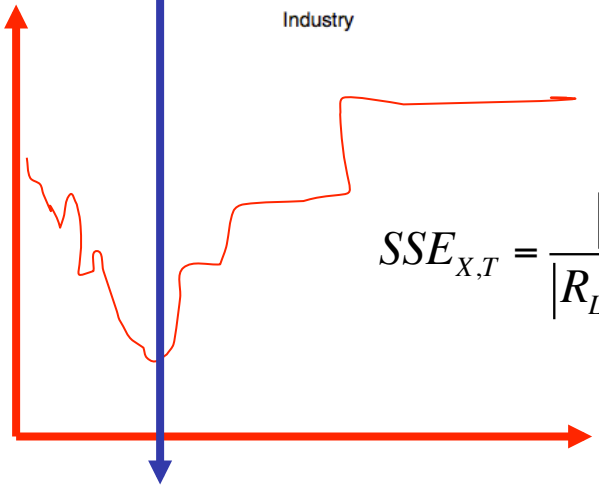
$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} Var(rightChild) \right]$$

Regression Tree: Find optimal first split point



Find optimal first split point to split global region into left and right regions

$SSError_T$ As the split point of Industry is varied



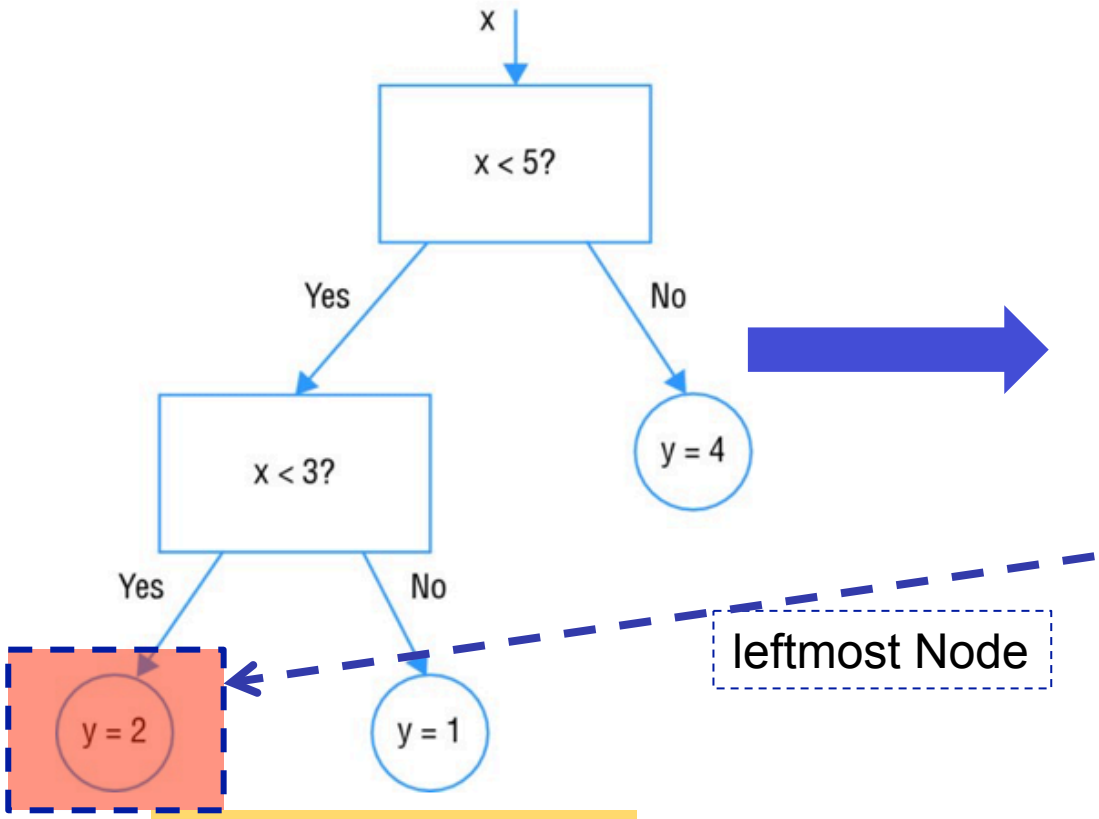
$$SSE_{X,T} = \frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} (y_i - \mu_{R_l})^2 + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} (y_i - \mu_{R_R})^2$$

Optimal Split point

Binary Decision Tree: Missing Data

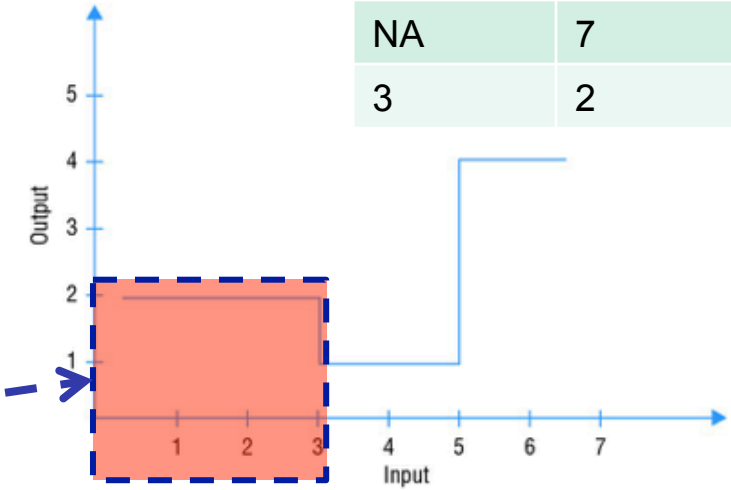
Replace by average
Delete records with missing values

X	Y
1	2
3	2
5	4
6	4
3	1
NA	7
NA	7
3	2



Binary Decision Tree

leftmost Node

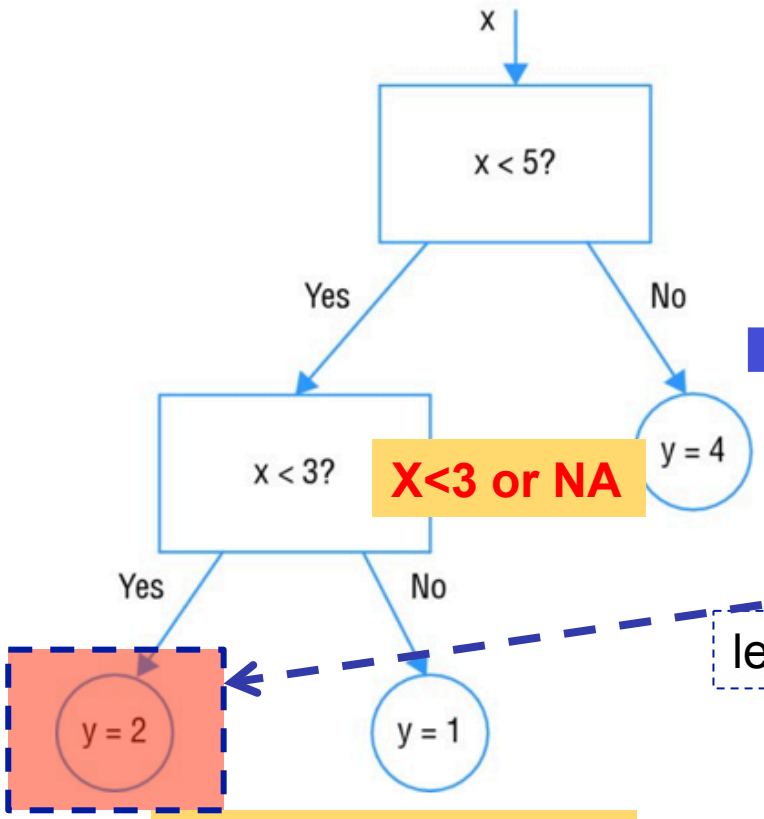


Input-output graph for the Binary Decision Tree example

Binary Decision Tree: Missing Data

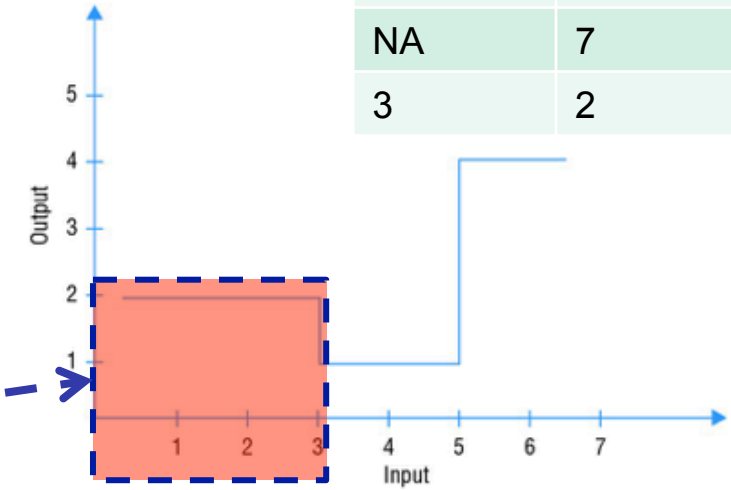
X	Y
1	2
3	2
5	4
6	4
3	1
NA	1.7
NA	7
3	2

X==NA



Binary Decision Tree

leftmost Node



Input-output graph for the Binary Decision Tree example

Classification versus regression Trees

- Tree models where the target variable can take a finite set of values are called classification trees.
- In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

- Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

Dtrees are desirable: plug and play

- One of the most popular approaches to ML in practice
- can handle numeric, categorical, nominal
- No preprocessing required, no standarization

- Handles Missing values naturally and Nas does not affect performance metrics
- Interaction features
- Highly Scaleable
- Variable selection
- Excellent performance on a variety of problems

- Off the shelf with very few hyperparameters

- **In practice we generally use ensembles of decision trees**

- **Go back to basics first for regression trees (DTs for regression)**

modes, medians and means and ML

Summary Statistics and Loss Functions

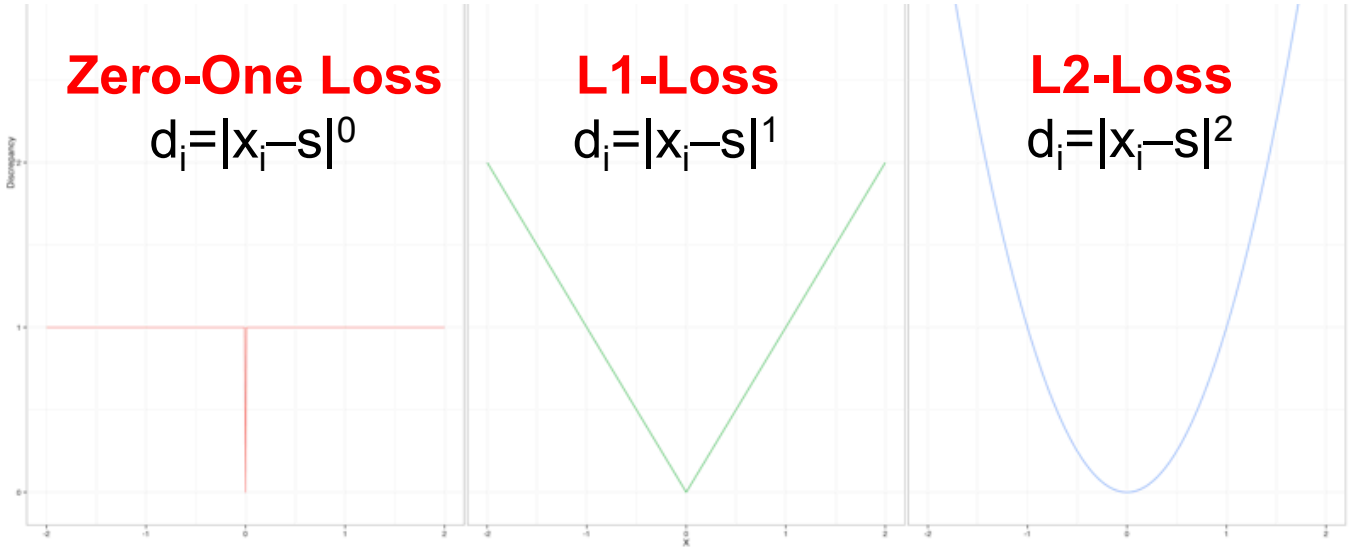
- To see how modes, medians and means arise, let's assume that we have a list of numbers, (x1,x2,…,xn), that we want to summarize (in terms of central behavior).
- We want our summary to be a single number, which we'll call s.
- How should we select s so that it summarizes the numbers, (x1,x2,…,xn), effectively?

- To answer that, we'll assume that s is an effective summary of the entire list if the typical discrepancy/error between s and each of the xi is small.

From summary stats → Loss functions

- **Consider three definitions of the discrepancy, d_i:**

- **$d_i=|x_i-s|^0$** **# Remember $|0|^0=0$ $|\epsilon|^0=1$**
- **$d_i=|x_i-s|^1$** 0-loss
- **$d_i=|x_i-s|^2$** Absolute
  Squared



http://www.johnmyleswhite.com/notebook/2013/03/22/modes-medians-and-means-an-unifying-perspective/

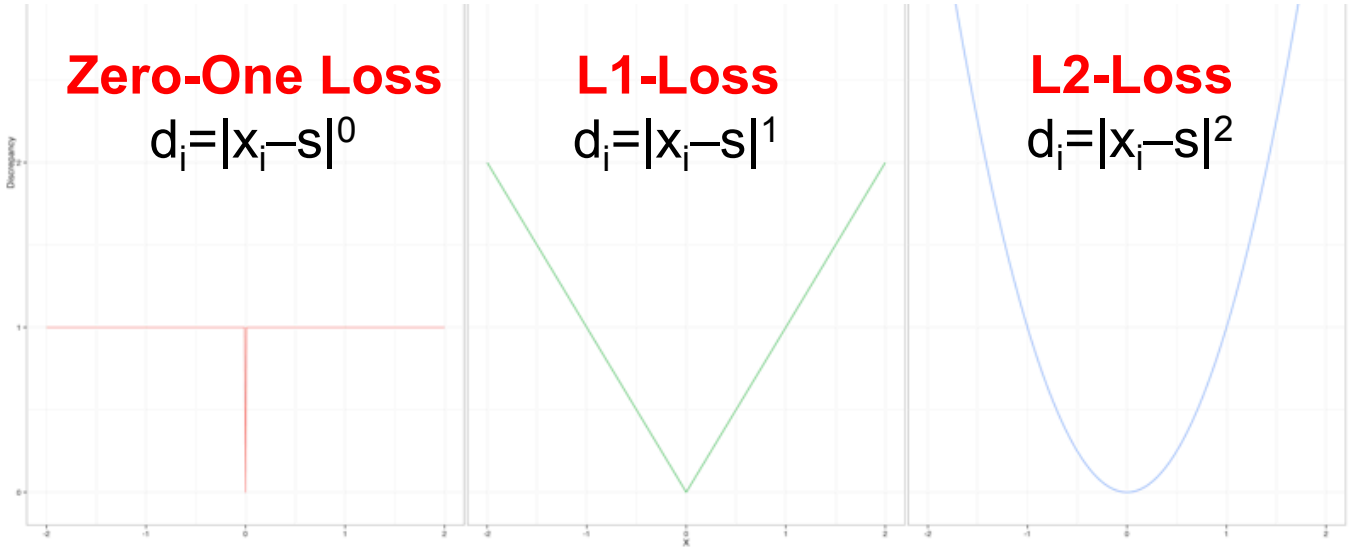
Summary Statistics and Loss Functions
Remember $|0|^0=0$ $|\epsilon|^0=1$

- **Let's all agree to assume that $0^0=0$.**
- **In particular, we'll want to assume that $|0|^0=0$, even though $|\epsilon|^0=1$ for all $\epsilon>0$.**
- **This definition is non-standard, but it greatly simplifies what follows and emphasizes the conceptual unity of modes, medians and means with regard to loss functions**

The mode minimizes the number of times that one of the numbers in our summarized list is not equal to the summary that we use.

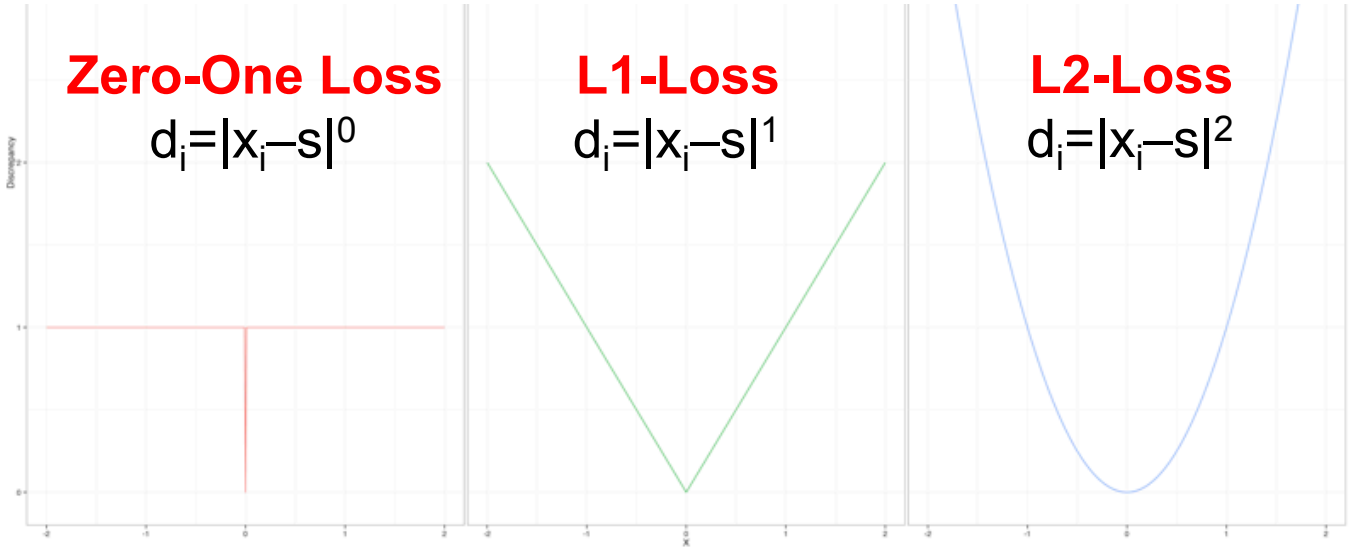
zero-one loss in ML

- The first definition, $d_i=|x_i–s|^0$, says that the discrepancy is 1 if $x_i \neq s$ and is 0 only when $x_i=s$.
- Note that this definition implies that anything other than exact equality produces a constant measure of discrepancy.
- Summarizing $x_i=2$ with $s=0$ is no better nor worse than using $s=1$.
- In other words, the discrepancy does not increase at all as s gets further and further from xi.



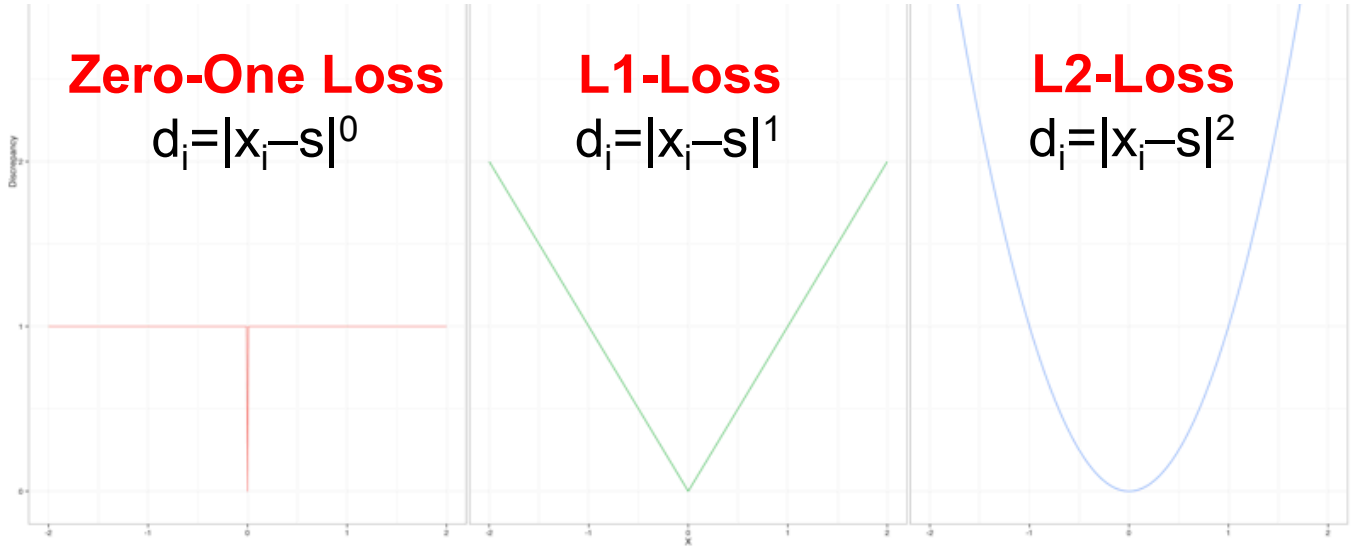
Deviation: $d_i=|x_i–s|^1$: L1-Norm

- The second definition, $d_i=|x_i–s|^1$, says that the discrepancy is equal to the distance between x_i and s.
- This is often called an absolute deviation in machine learning.
- Note that this definition implies that the discrepancy should increase linearly as s gets further and further from xi.



Squared error loss: L2 Norm

- The third definition, $d_i=|x_i–s|^2$, says that the discrepancy is the squared distance between x_i and s.
- Note that this definition implies that the discrepancy should increase super-linearly as s gets further and further from xi.
- For example, if $x_i=1$ and s=0, then the discrepancy is 1. But if xi=2 and s=0, then the discrepancy is 4.



Aggregate, minimize→ very different numbers

- When we write down these expressions in isolation, they don't look very different. But if we select s to minimize each of these three types of errors, we get very different numbers. And, surprisingly, each of these three numbers will be very familiar to us.

$$E_0 = \sum_i |x_i - s|^0.$$

$$E_1 = \sum_i |x_i - s|^1.$$

$$E_2 = \sum_i |x_i - s|^2.$$

Minimizing Aggregate Loss

To sum up, we've just seen that the three most famous single number summaries of a data set are very closely related: they all minimize the average discrepancy between s and the numbers being summarized. They only differ in the type of discrepancy being considered:

1. The mode minimizes the number of times that one of the numbers in our summarized list is not equal to the summary that we use.
2. The median minimizes the average distance between each number and our summary.
3. The mean minimizes the average squared distance between each number and our summary.

In equations,

1. The mode of $x_i = \arg\min_s \sum_i |x_i - s|^0$
2. The median of $x_i = \arg\min_s \sum_i |x_i - s|^1$
3. The mean of $x_i = \arg\min_s \sum_i |x_i - s|^2$

MathBox 3.1

Proof That the Sample Mean Is the Least Squares Estimate of μ

To show that the sample mean is the least squares estimate of the population mean involves only very elementary differential calculus. First we set out the formula for the sum of squares, expand it, then differentiate this expanded formula with respect to the estimate (denoted by M), set this derivative to 0, and finally solve for M.

We need to find the value of M that yields the minimum sum of the squared residuals, SS.

The sum of squares if given by the expression

$$SS = \Sigma(Y - M)^2$$

Expanding the right-hand side, we have

$$SS = \Sigma(Y^2 - 2YM + M)^2$$
$$= \Sigma Y^2 - 2M\Sigma Y - \Sigma M^2$$

We now differentiate with respect to M:

$$\frac{\partial SS}{\partial M} = -2\Sigma Y + 2\Sigma M$$

Because the desired value of M is a constant, $\Sigma M = nM$ (M is added to itself n times). Thus

$$\frac{\partial SS}{\partial M} = -2\Sigma Y + 2nM$$

Setting this derivative to 0 gives

$$-2\Sigma Y + 2nM = 0$$

This relation gives us

$$M = \frac{\Sigma Y}{n}$$

which is the sample mean, $\overline{Y}$.

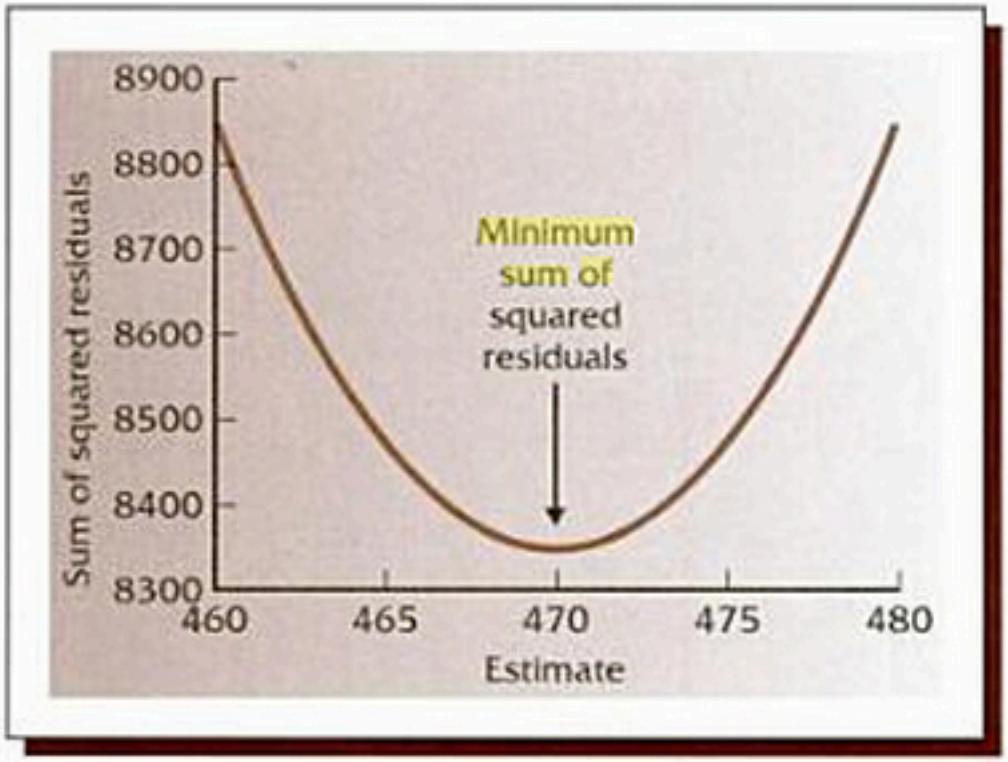


Figure 3.1
The sum of squared residuals varies in size, depending on the value used to estimate the parameter μ. The sum of squares is a minimum when the mean of the data is used as the estimate.

Decision tree learning : Real-valued Target

- **Decision tree learning for real-valued target variable comes down to summary statistics.**

- **For centrality depending on the loss function:**
  - Mean
  - Median
- **For spread: used in variable selection**
  - Variance
  - F-Test

$$\mathrm{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2.$$

Variance of a discrete Random Variable

Discrete random variable [edit]

If the random variable X is discrete with probability mass function $x_1 \mapsto p_1, \ldots, x_n \mapsto p_n$, then

$$\operatorname{Var}(X) = \sum_{i=1}^{n} p_i \cdot (x_i - \mu)^2 = \sum_{i=1}^{n} (p_i \cdot x_i^2) - \mu^2$$

where μ is the expected value, i.e.

$$\mu = \sum_{i=1}^{n} p_i \cdot x_i \,.$$

(When such a discrete weighted variance is specified by weights whose sum is not 1, then one divides by the sum of the weights.)

The variance of a set of n equally likely values can be written as

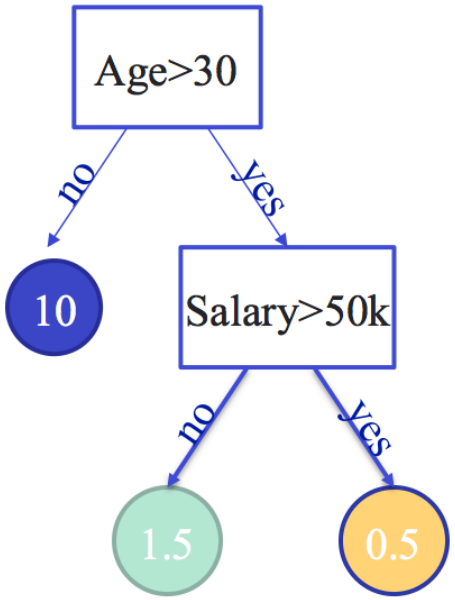
$$\operatorname{Var}(X) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2.$$

where μ is the expected value, i.e.

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Decision tree learning

- **Decision tree learning is a supervised machine learning approach whose goal is to recursively partition the world into homogenous zones, i.e., zones where the target value is homogenous (constant)**



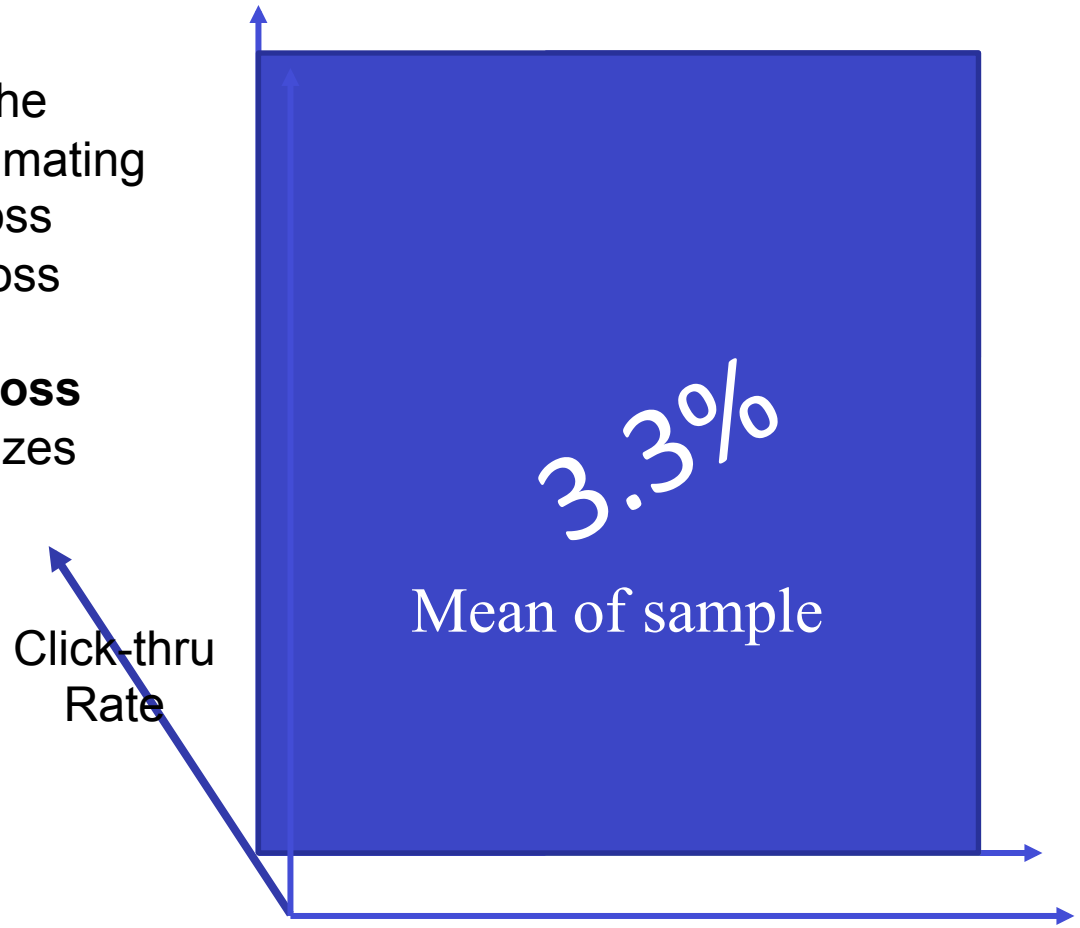
Decision Tree-based Segmentation

$$eCPM_{Ad} = CR_{Ad} \times Bid_{Ad}$$

HeatMap

-

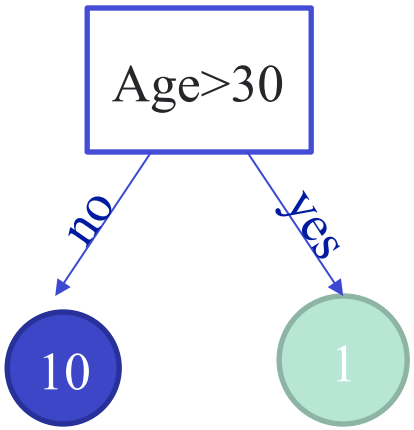
MEAN under SSE Loss Function
Under typical statistical assumptions, the mean or average is the statistic for estimating location that minimizes the expected loss experienced under the squared-error loss function,
MEDIAN under absolute-difference loss
the median is the estimator that minimizes expected loss experienced under the absolute-difference loss function.

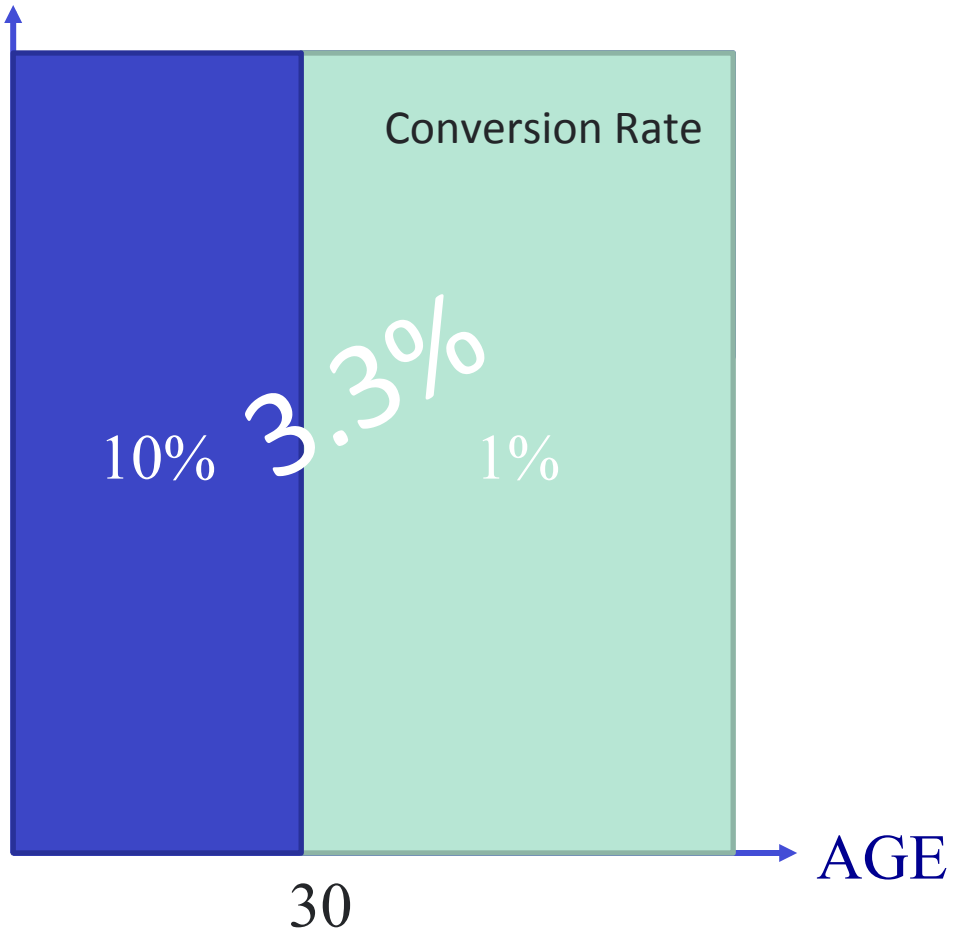


Decision Tree-based Segmentation

$$eCPM_{Ad} = CR_{Ad} \times Bid_{Ad}$$

HeatMap



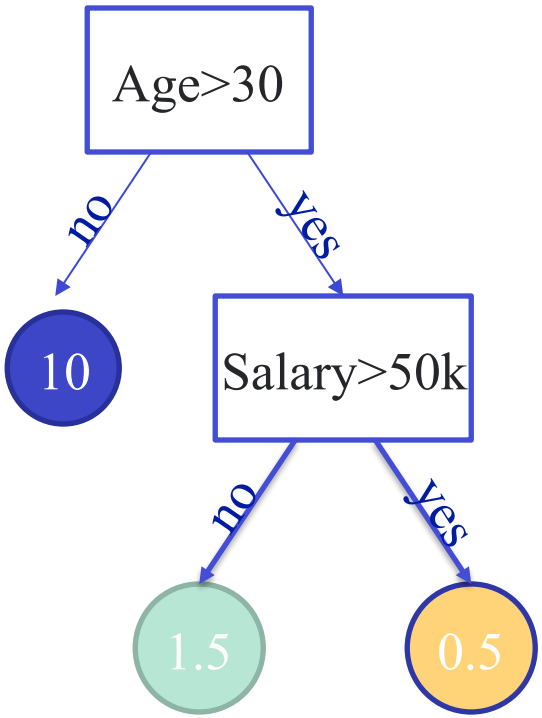


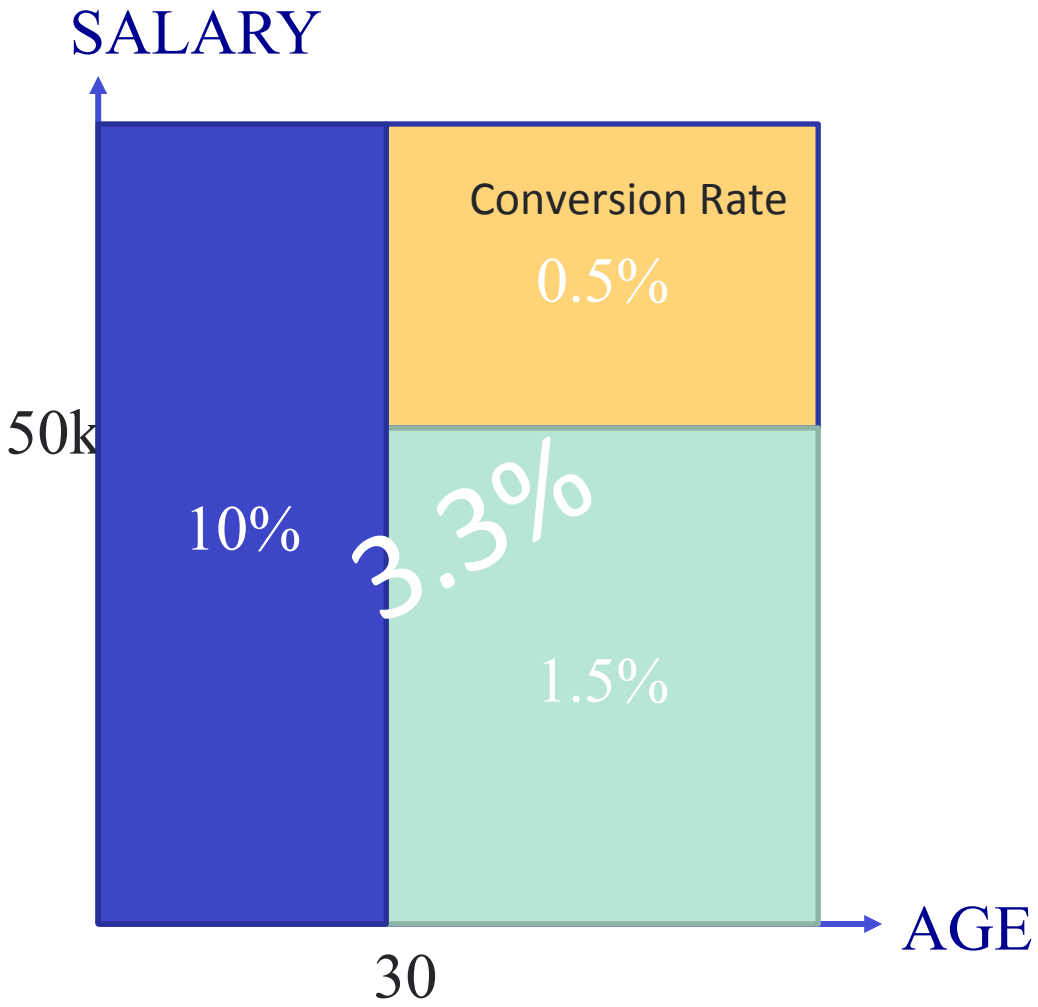
If Age > 30
Then predict CR = 1
Else predict CR =10

Decision Tree-based Segmentation (3D)

$$eCPM_{Ad} = CR_{Ad} \times Bid_{Ad}$$

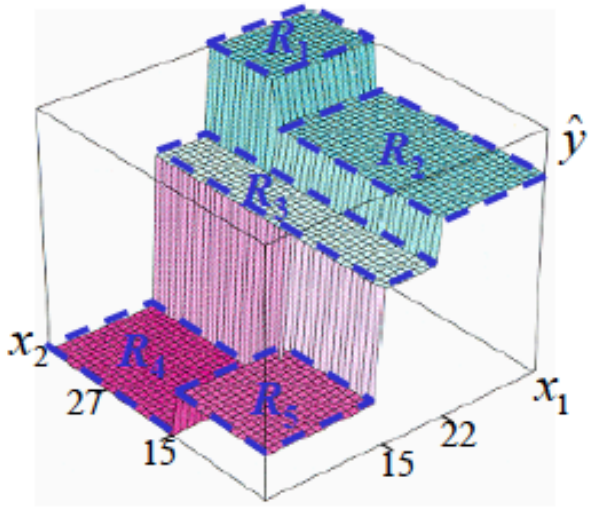
HeatMap





Trees in 3D and as conjunctive rules

- Trees as collection of conjunctive rules: $T_m(\mathbf{x}) = \sum_{j=1}^{J} \hat{c}_{jm} I(\mathbf{x} \in \hat{R}_{jm})$



$$R_1 \Rightarrow r_1(\mathbf{x}) = I(x_1 > 22) \cdot I(x_2 > 27)$$

$$R_2 \Rightarrow r_2(\mathbf{x}) = I(x_1 > 22) \cdot I(0 \le x_2 \le 27)$$

$$R_3 \Rightarrow r_3(\mathbf{x}) = I(15 < x_1 \le 22) \cdot I(0 \le x_2)$$

$$R_4 \Rightarrow r_4(\mathbf{x}) = I(0 \le x_1 \le 15) \cdot I(x_2 > 15)$$

$$R_5 \Rightarrow r_5(\mathbf{x}) = I(0 \le x_1 \le 15) \cdot I(0 \le x_2 \le 15)$$

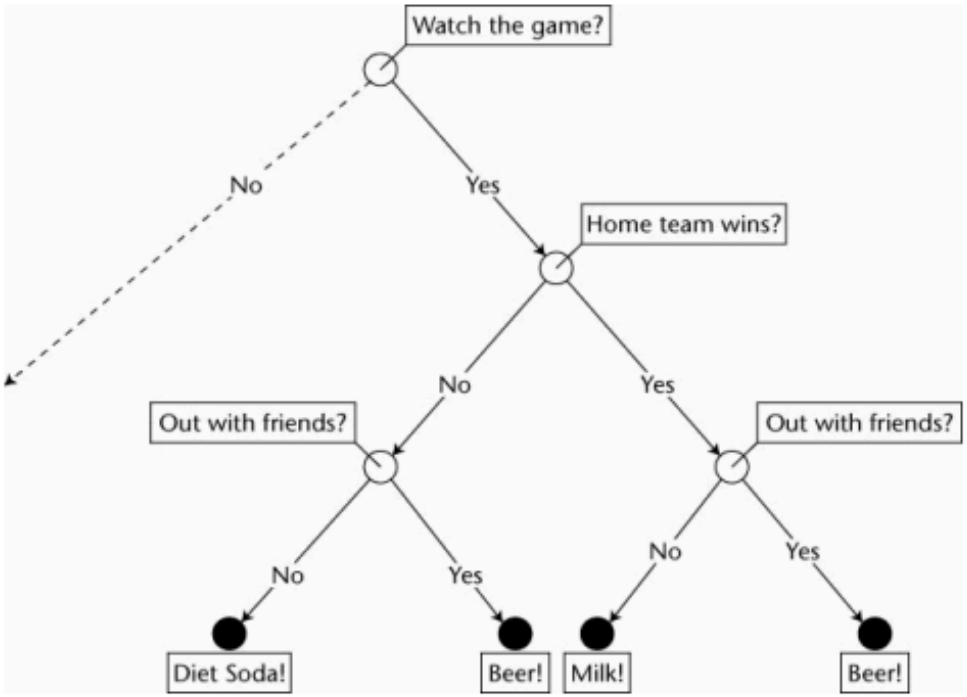
- These simple rules, $r_m(\mathbf{x}) \in \{0,1\}$, can be used as base learners
- Main motivation is *interpretability*

$T_m(\mathbf{x}) = \sum_{j=1}^{J} \hat{c}_{jm} I(\mathbf{x} \in \hat{R}_{jm})$ compact means of representing a D; and inference using DTs

C_{jm} corresponds to the mean of the target variable of the examples falling into the R_{jm}; Indicator function I()

Rules from decision trees

IF Handset Churn Rate >= 3.8%
AND Call Trend < 0.0056
AND Total Amt. Overdue < 4855
THEN likelihood of Churn is 67.3% (110 cases, 66% on Validation)



Conditions for beer

IF Watch the game
AND Out with friends
THEN Beer

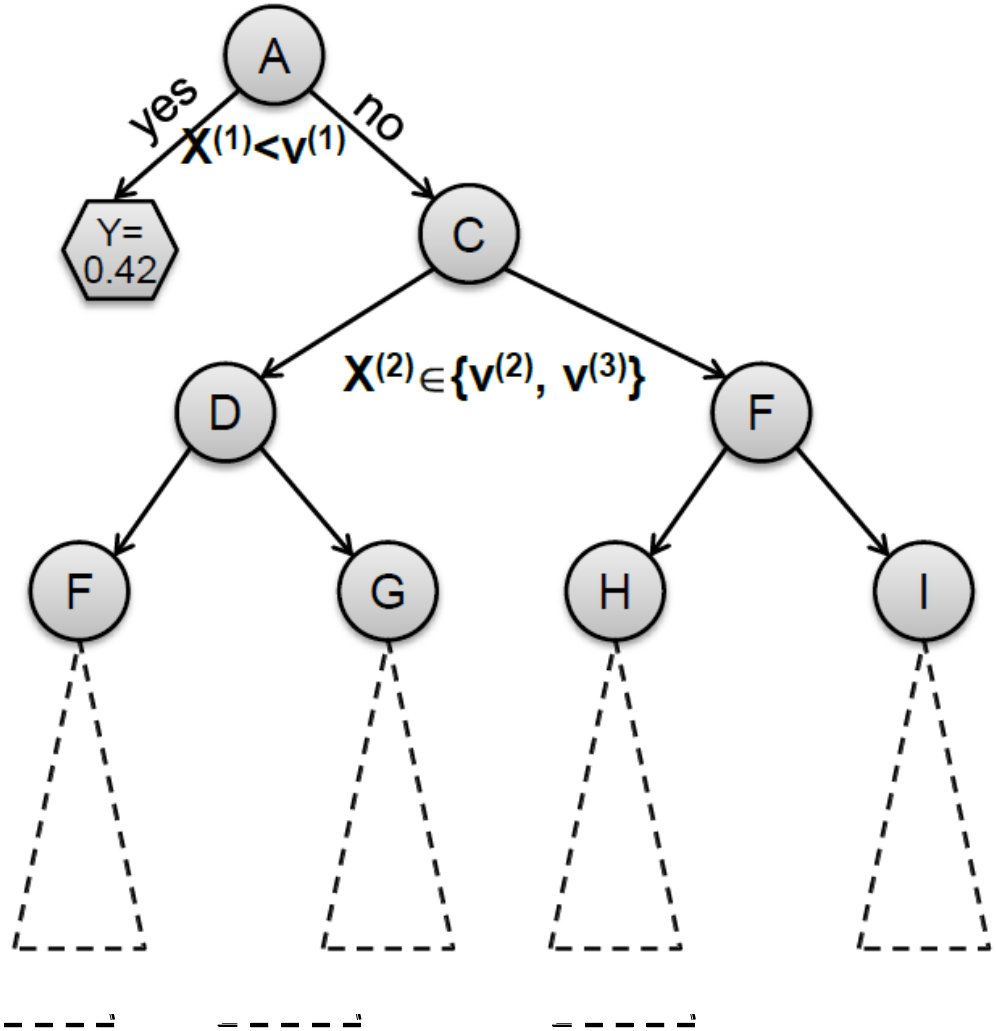
C5.0 – further generalized rules
More compact description

Decision Tree Learning

- **Give one attribute (e.g., lifespan), try to predict the value of new people's lifespans by means of some of the other available attribute**
- **Input attributes:**
  - **d** features/attributes: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots \mathbf{x}^{(d)}$
  - Each $\mathbf{x}^{(j)}$ has **domain** $\mathbf{O}_j$
    - **Categorical:** $\mathbf{O}_j$ = {red, blue}
    - **Numerical:** $\mathbf{H}_j$ = (0, 10)
  - $\boldsymbol{Y}$ is output variable with domain $\boldsymbol{O}_Y$:
    - **Categorical:** Classification, **Numerical:** Regression
- **Data D:**
  - $\boldsymbol{n}$ examples $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ where $\boldsymbol{x}_i$ is a $\boldsymbol{d}$-dim feature vector, $\boldsymbol{y}_i \in \boldsymbol{O}_Y$ is output variable
- **Task:**
  - Given an input data vector $\boldsymbol{x}$ predict $\boldsymbol{y}$

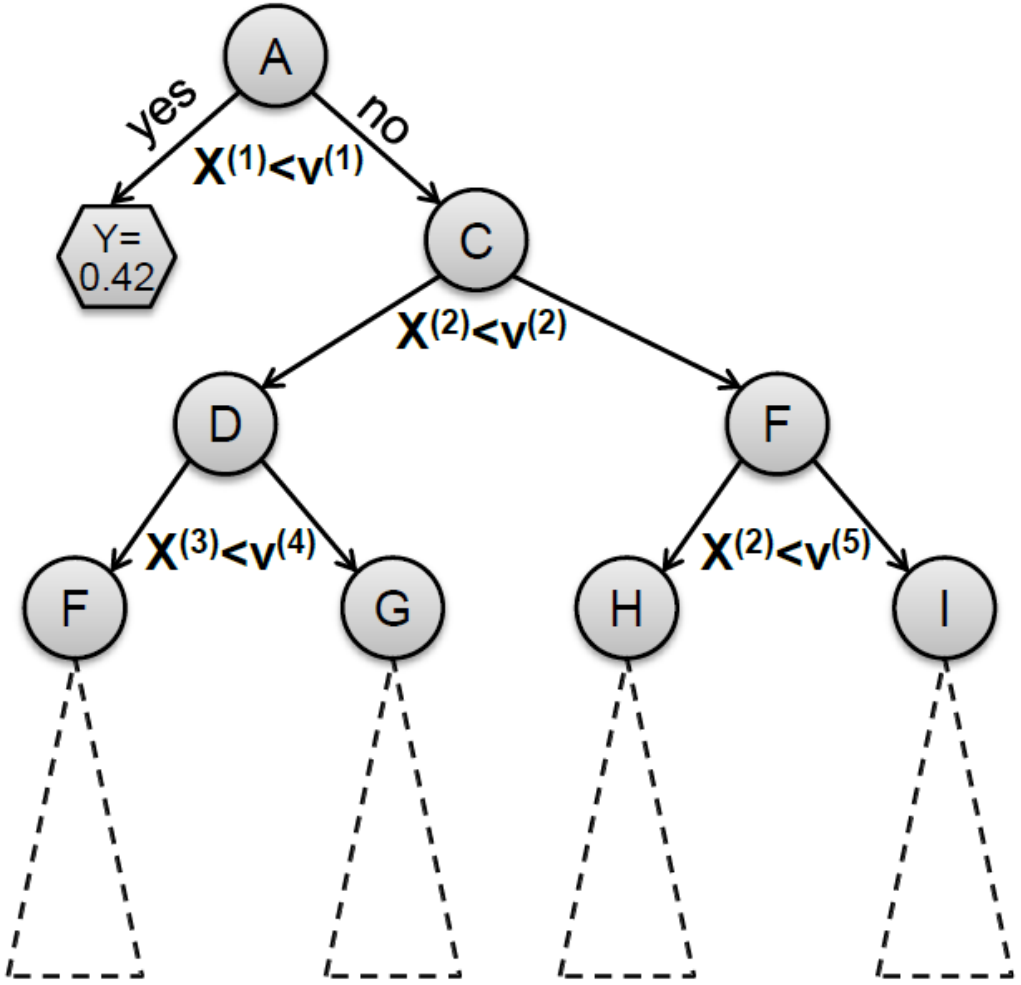
Decision Tree Learning

- A **Decision Tree** is a tree-structured plan of a set of attributes to test in order to predict the output



Decision Tree

- **Decision trees:**
  - Split the data at each internal node
  - Each leaf node makes a prediction
- **Lecture today:**
  - Binary splits: $X^{(j)}<v$
  - Numerical attrs.
  - Regression



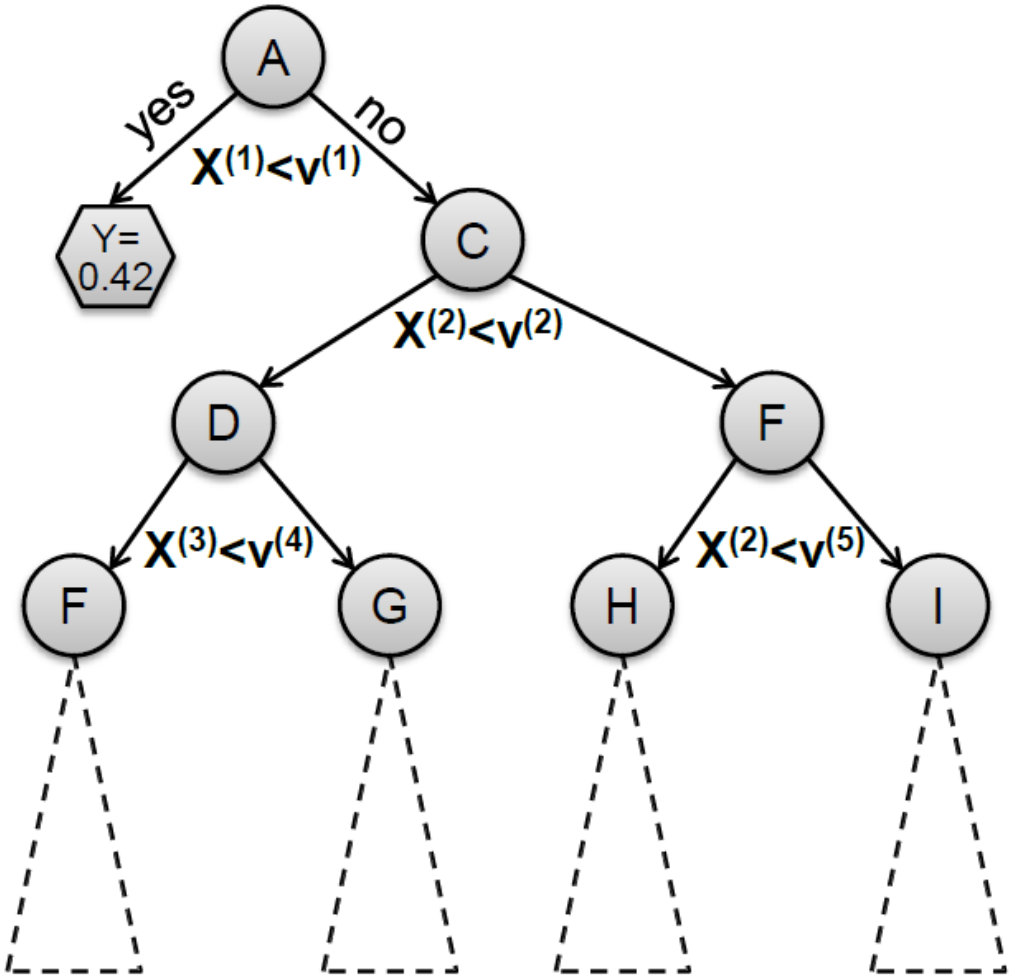
How to make predictions?

- **Input:** Example x_i
- **Output:** Predicted y_i'

- "Drop" x_i down the tree until it hits a leaf node

- Predict the value stored in the leaf that x_i hits

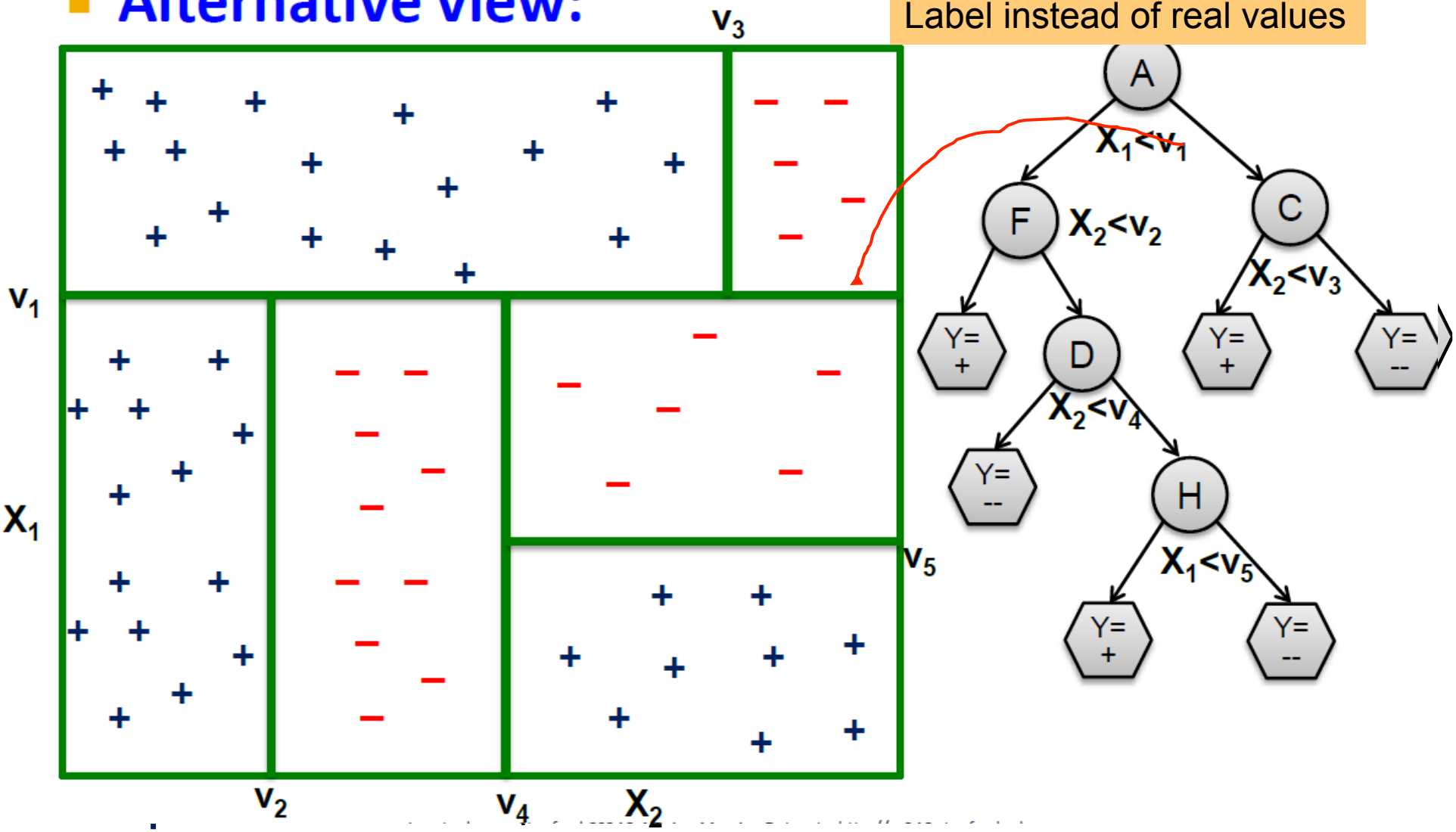
Assign an ID to each node in the tree



Classification Decision Trees Vs. SVM

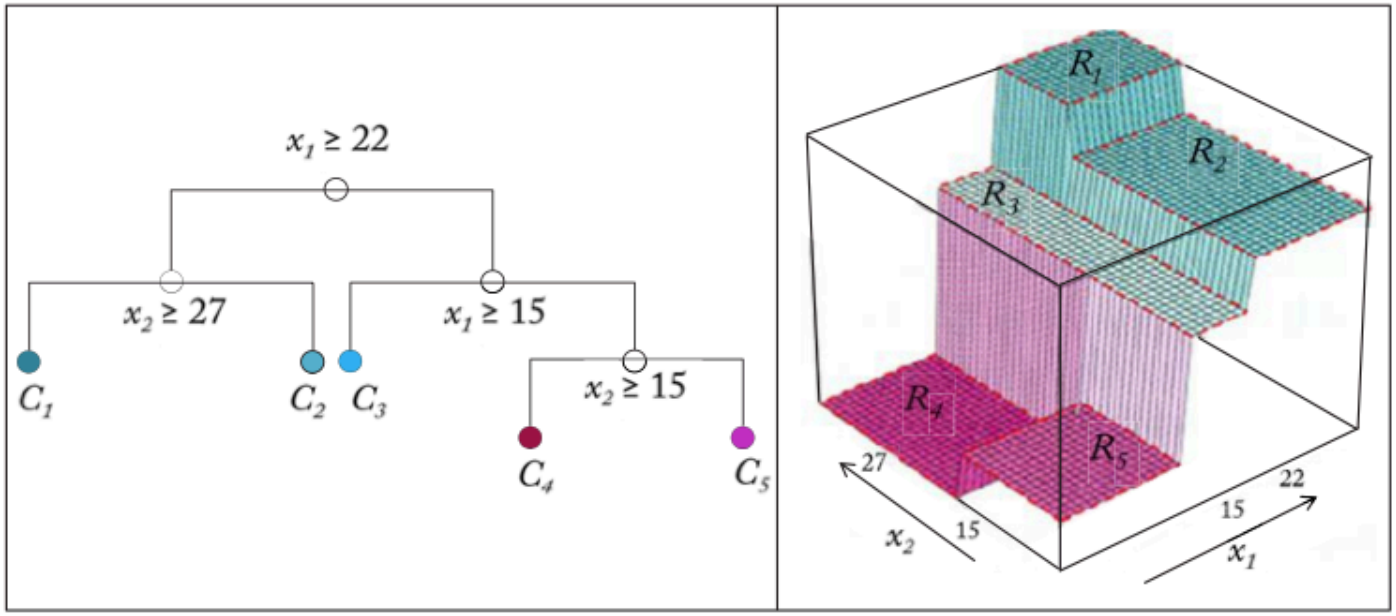
- **Alternative view:**

Leaf nodes
Label instead of real values



- **In this section we are going focus on the basics of learning a regression tree**

Regression Tree Learning: learn an m-leaf node tree (m-regions)



$$\hat{y} = T(\mathbf{x}) = \sum_{m=1}^{M} \hat{c}_m I_{\hat{R}_m}(\mathbf{x})$$

Hyper rectangles

Figure 2.3: Sample regression tree and corresponding surface in input ($\mathbf{x}$) space (adapted from (Hastie et al., 2001)).

If we choose to use squared-error loss, then the search problem, finding the tree $T(\mathbf{x})$ with lowest prediction risk, is stated:

$$\left\{ \hat{c}_m, \hat{R}_m \right\}_1^M = \arg\min_{\{c_m, R_m\}_1^M} \sum_{i=1}^{N} [y_i - T(\mathbf{x}_i)]^2$$

$$= \arg\min_{\{c_m, R_m\}_1^M} \sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} c_m I_{R_m}(\mathbf{x}_i) \right]^2$$

learn m-leaf node tree (m-regions) That minimizes this loss function over the m-regions (squared error loss function)

Regression Tree: Loss Functions: L2 (squared error) or L1 (absolute error)

In this section, we look more closely at the algorithm for building decision trees. Figure 2.3 shows an example surface built by a regression tree. It's a piece-wise constant surface: there is a "region" R_m in input space for each terminal node in the tree – i.e., the (hyper) rectangles induced by tree cuts. There is a constant associated with each region, which represents the estimated prediction $\hat{y} = \hat{c}_m$ that the tree is making at each terminal node.

Formally, an M-terminal node tree model is expressed by:

$$\hat{y} = T(\mathbf{x}) = \sum_{m=1}^{M} \hat{c}_m I_{\hat{R}_m}(\mathbf{x})$$

Disjoint

where $I_A(\mathbf{x})$ is 1 if $\mathbf{x} \in A$ and 0 otherwise. Because the regions are disjoint, every possible input $\mathbf{x}$ belongs in a single one, and the tree model can be thought of as the sum of all these regions.

Trees allow for different loss functions fairly easily. The two most used for regression problems are *squared-error* where the optimal constant $\hat{c}_m$ is the mean and the *absolute-error* where the optimal constant is the median of the data points within region R_m (Breiman et al., 1993).

DT Learning: joint optimization of regions and constants?

If we choose to use squared-error loss, then the search problem, finding the tree $T(\mathbf{x})$ with lowest prediction risk, is stated:

$$\left\{ \hat{c}_m, \hat{R}_m \right\}_1^M = \underset{\{c_m, R_m\}_1^M}{\arg\min} \sum_{i=1}^{N} [y_i - T(\mathbf{x}_i)]^2$$

$$= \underset{\{c_m, R_m\}_1^M}{\arg\min} \sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} c_m I_{R_m}(\mathbf{x}_i) \right]^2$$

To solve, one searches over the space of all possible constants and regions to minimize average loss. Unrestricted optimization with respect to $\{R_m\}_1^M$ is very difficult, so one universal technique is to restrict the shape of the regions (see Figure 2.4).

Joint optimization with respect to $\{\mathbf{R_m}\}_1^{\mathbf{M}}$ and $\{\mathbf{c_m}\}_1^{\mathbf{M}}$, simultaneously, is also extremely difficult, so a greedy iterative procedure is adopted. Start with a global region with all training data and computing the error (variance)

$$\hat{e}(R) = \frac{1}{N} \sum_{\mathbf{x} \in R} \left(y_i - \text{mean}\left(\{y_i\}_1^N \right) \right)^2$$

Then each input variable $\mathbf{x_j}$, and each possible split value $\mathbf{s_j}$ on that particular variable for splitting **R** into $\mathbf{R_L}$ (left region) and $\mathbf{R_r}$ (right region), is considered, and scores $\hat{\mathbf{e}}(\mathbf{R_{Left}})$ and $\hat{\mathbf{e}}(\mathbf{R_{Right}})$ computed.

Regression Tree with continuous inputs

- Assume a single input variable and a single target variable
- Both target and input variables are continuous/ real-valued variables

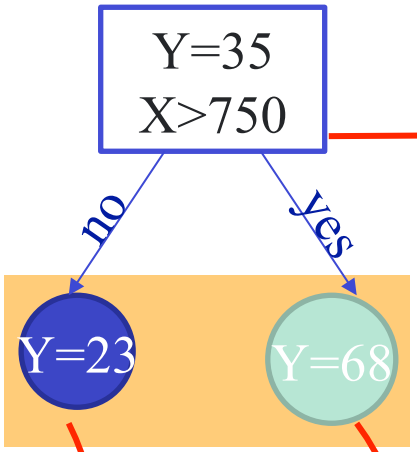
- **Learning a regression tree over real-valued variables**

- **Switch from 3D (2 independent variables) world**

- **to a 2D world (one independent variable)**

Improve the purity (or spread) of target variable in each new sub-region

- **Decision tree learning is a supervised machine learning approach whose goal is to recursively partition the world into homogenous zones, i.e., zones where the target value is homogenous (constant with low)**



$$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

$$\hat{I}(x_j, s_j) = \hat{e}(R) - \hat{e}(R^l) - \hat{e}(R^r)$$

$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|}\sum_{x_i \in R_l(X,T)} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|}\sum_{x_i \in R_R(X,T)} Var(LeftChild)\right]$$

Left

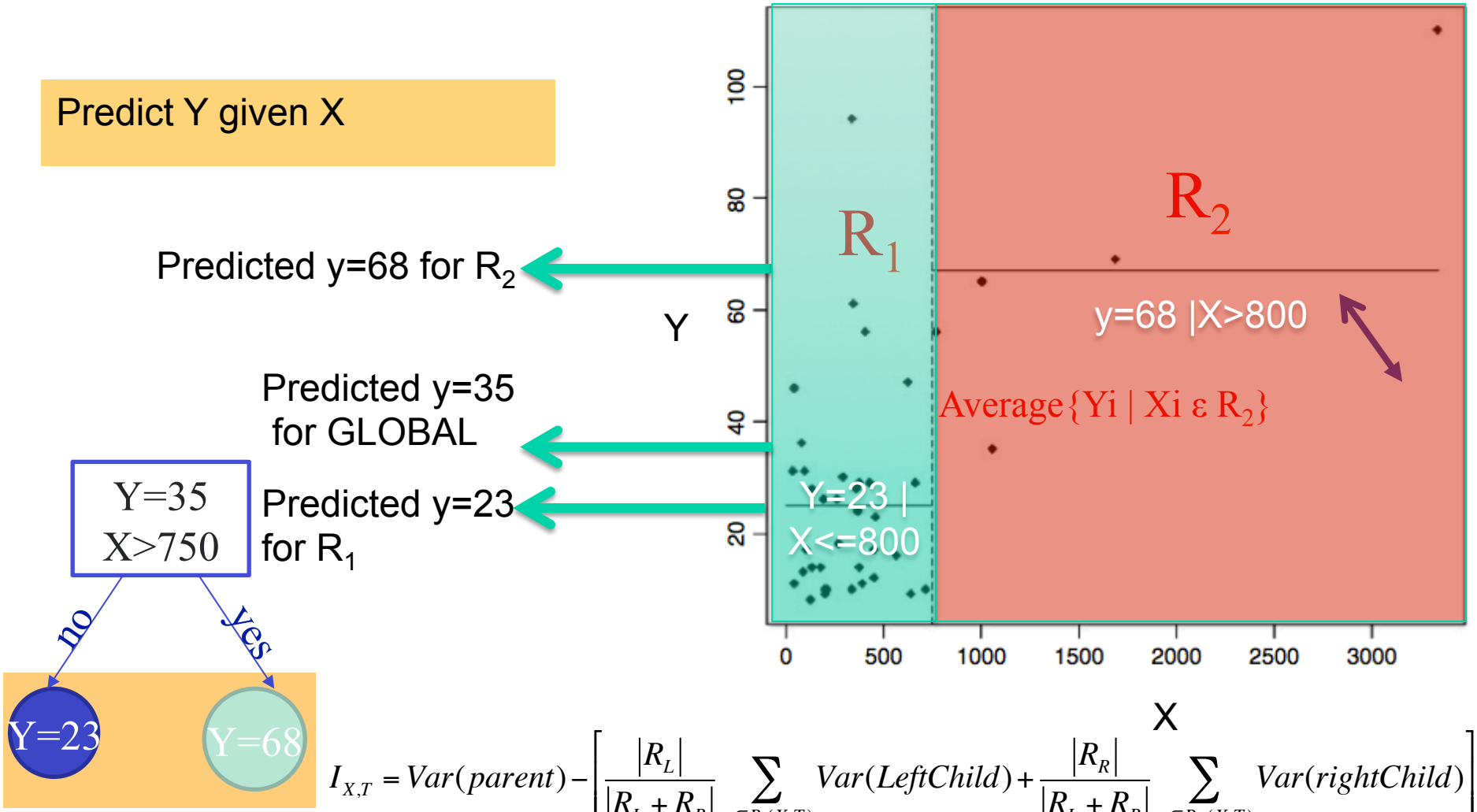
Right

$$SSE_{X,T} = \frac{|R_L|}{|R_L + R_R|}\sum_{x_i \in R_l(X,T)} (y_i - \mu_{R_l})^2 + \frac{|R_R|}{|R_L + R_R|}\sum_{x_i \in R_R(X,T)} (y_i - \mu_{R_R})^2]$$

Mean value prediction for regression trees with squared error loss

- **Partition the space into *M* regions: R_1, R_2**

Predict Y given X

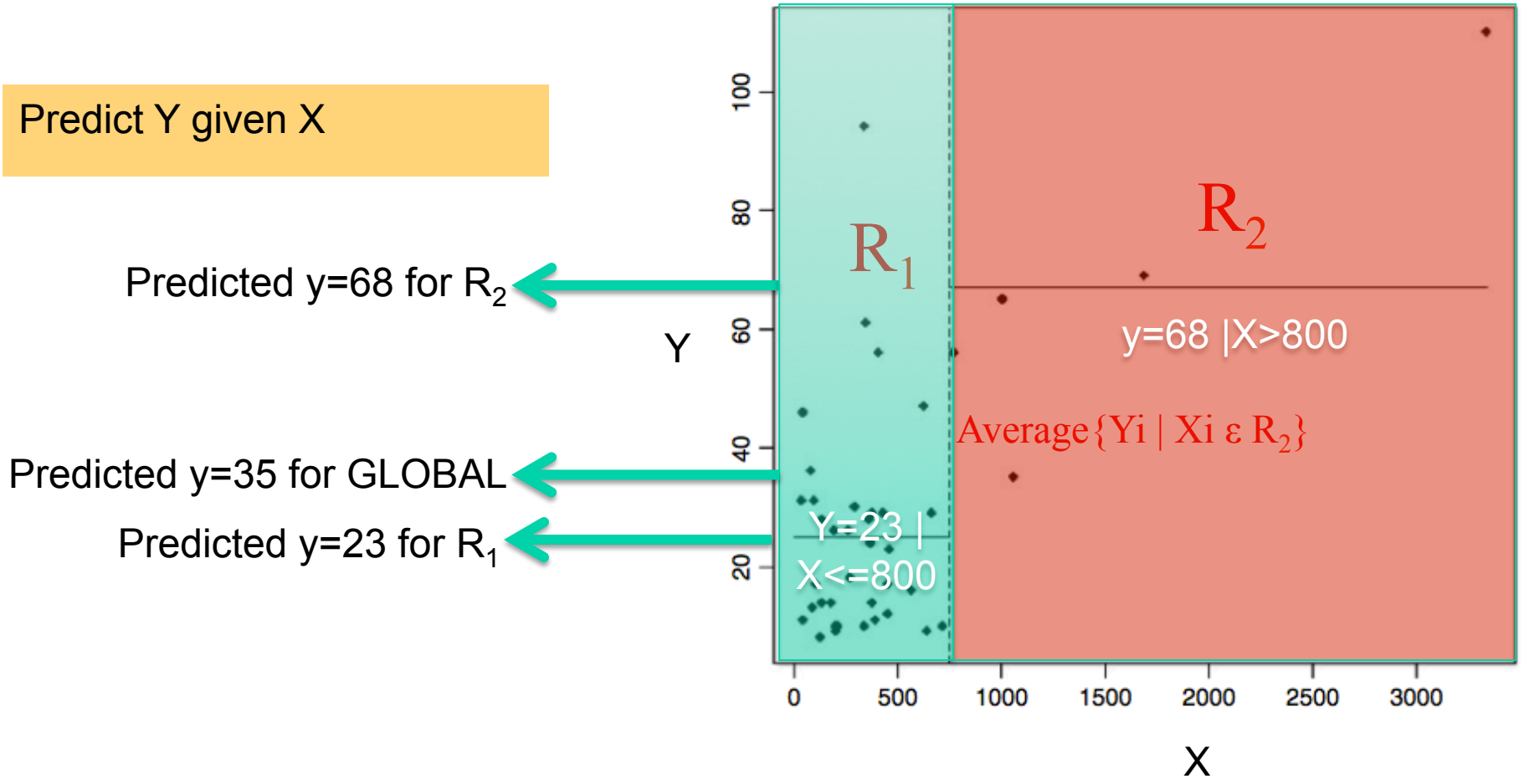


$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} Var(rightChild) \right]$$

Mean value prediction for regression trees with squared error loss

- **Partition the space into *M* regions: R_1, R_2**

Predict Y given X



Predict for X?

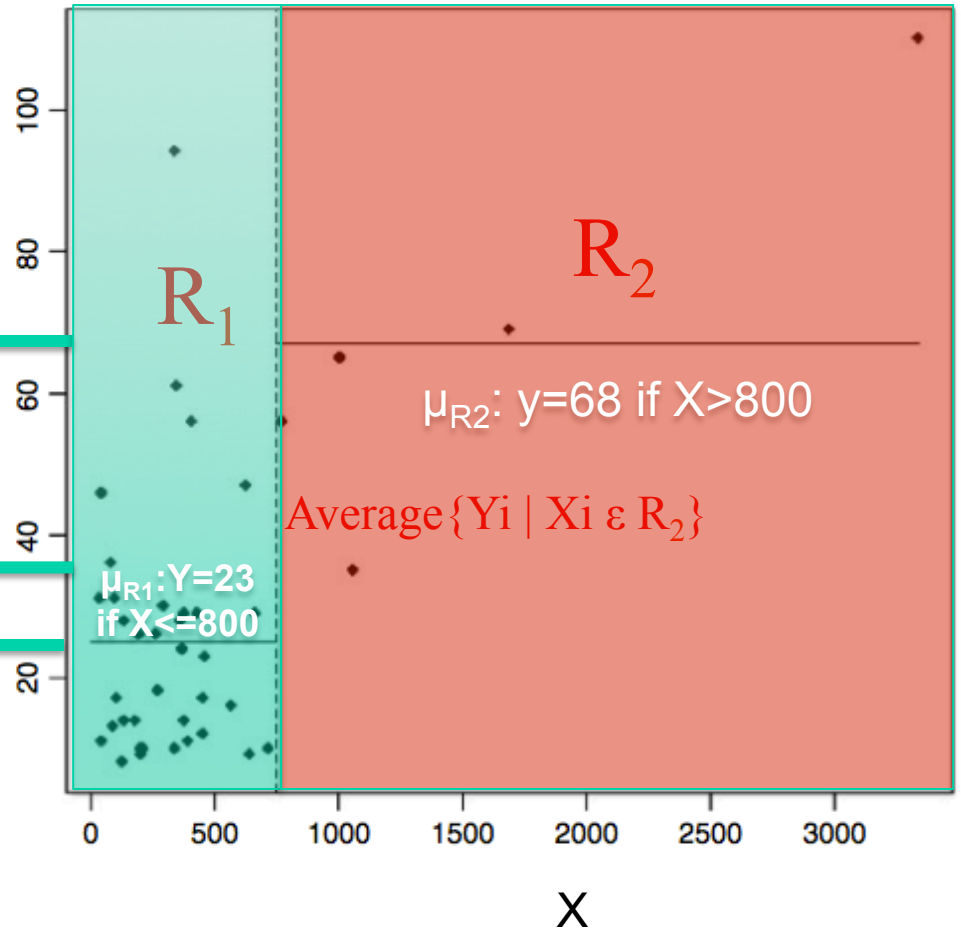
- **Partition the space into *M* regions: R_1, R_2**

Predict Y given X

μ_{R2} :Predicted μ_{R2} y=68 for R_2

μ_{Global} :Predicted y=35 for GLOBAL

μ_{R1} :Predicted y=23 for R_1



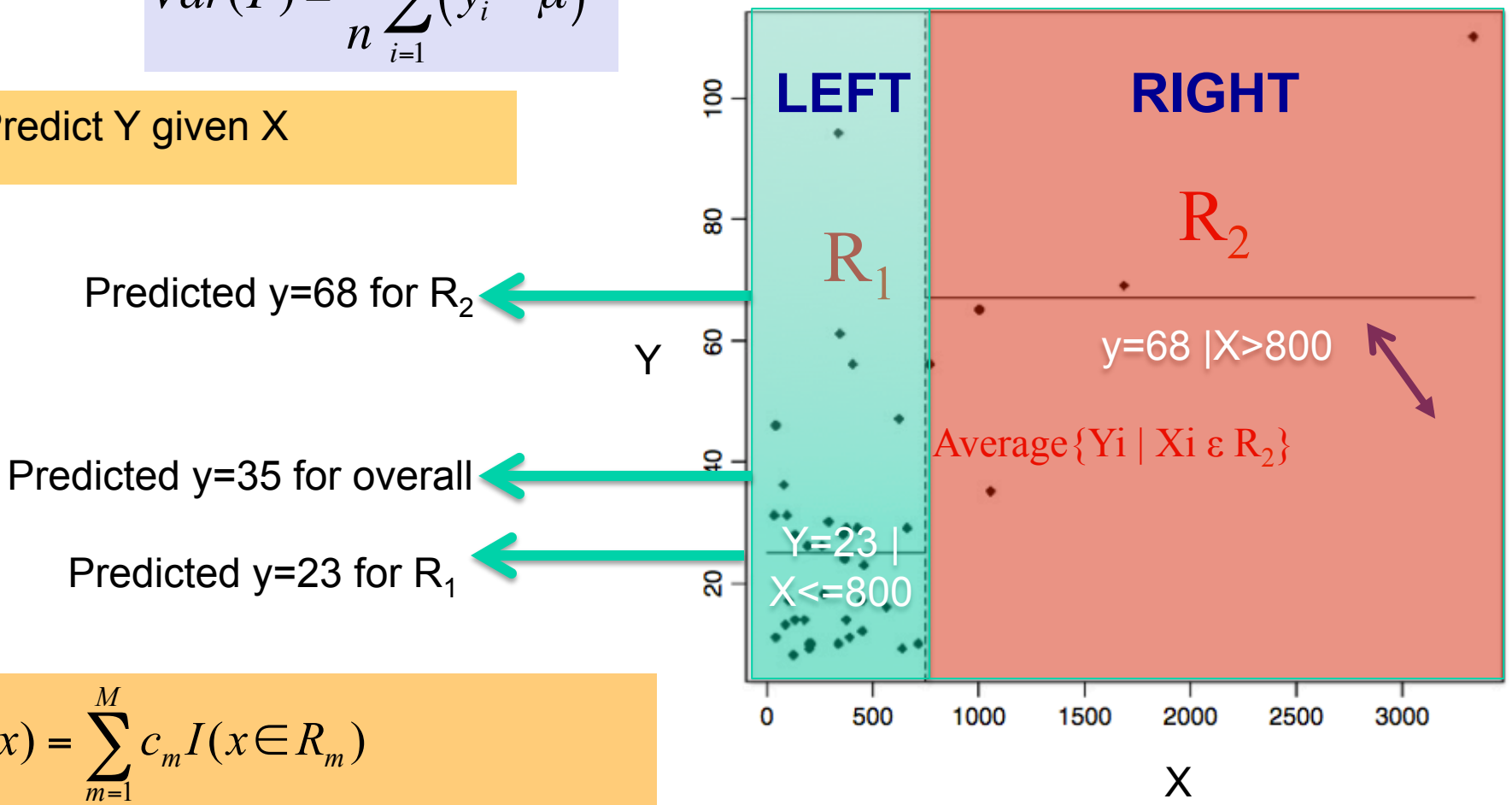
$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

$$where \quad \hat{c}_m = average(y_i | x_i \in R_m)$$

Which Split? Look for purity as measured by Var

$$Var(Y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mu)^2$$

Predict Y given X

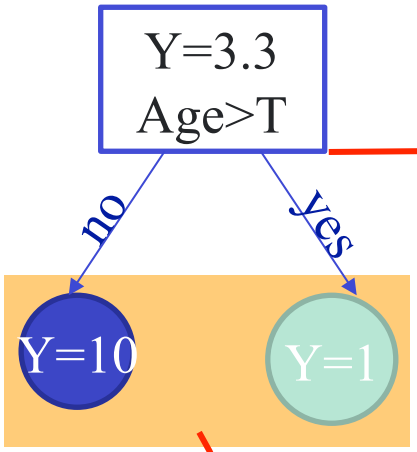


$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

$$where, \quad \hat{c}_m = average(y_i | x_i \in R_m)$$

Regression Trees – Grow the Tree

- **Measure quality of split based on variance reduction**
- **Measure quality of target variable partition in terms Variance (aka sum of squared error) for root node or parent node in term of MSE**
  - (aka variance in DT as the node represents a region as region wide constant):



$$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

- **Consider a single variable problem with a single input variable X_J. Find the threshold value T that minimizes the sum of squared error.**
- **This problem can be summarized as follows:**

Left Right

$$SSE_{X,T} = \frac{|R_L|}{|R_L + R_R|}\sum_{x_i \in R_l(X,T)}(y_i - \mu_{R_l})^2 + \frac{|R_R|}{|R_L + R_R|}\sum_{x_i \in R_R(X,T)}(y_i - \mu_{R_R})^2]$$

Recursively Split Variable Weight

Partition the space into *M* regions: R_1, R_2, …, R_M

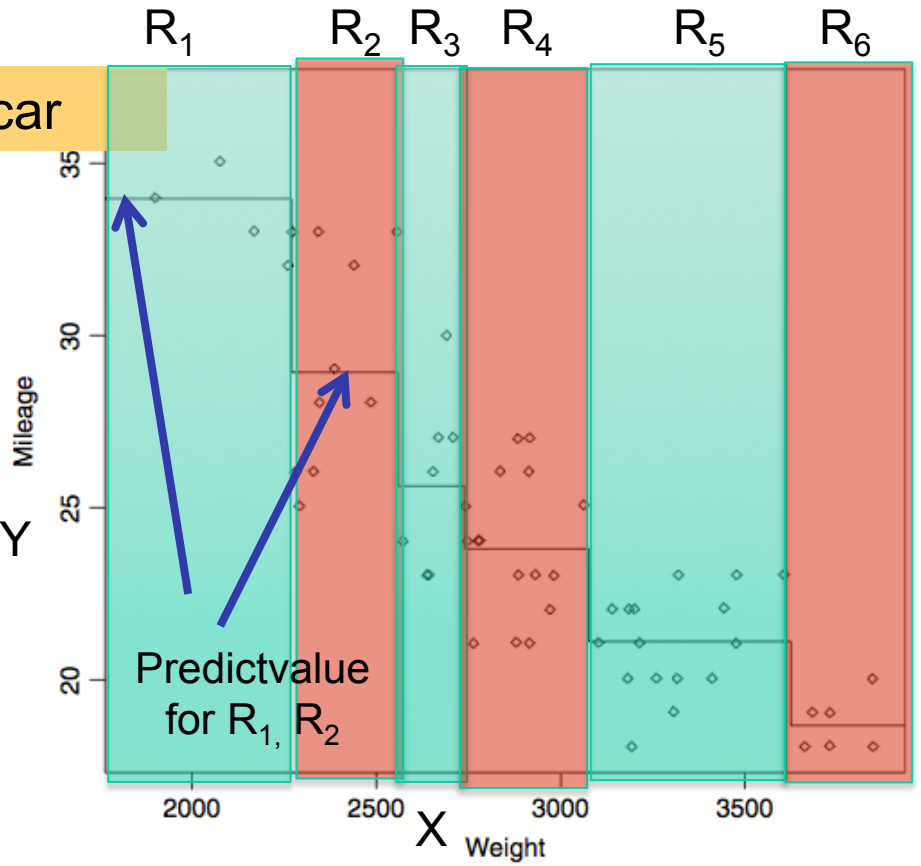
1. Find best binary split leading to two regions R_{left} and R_{Right};
2. foreach region in {regions R_{left} and R_{Right}}
    1. Repeat Step 1
    2. Stop when the number of data points is small or the variance is small

Predict car mileage based on weight of car

Try this at home!

```
library("rpart")
library(tree) # install.packages("tree")
head(car.test.frame)
car.model<-tree(Mileage~Weight, data =
car.test.frame)
plot(car.test.frame[,'Weight'],
car.test.frame[,'Mileage'])

wt<-seq(1500,4000,10)
y<-predict(car.model,list(Weight=wt))
lines(wt,y)
```



Regression Trees – Grow the Tree

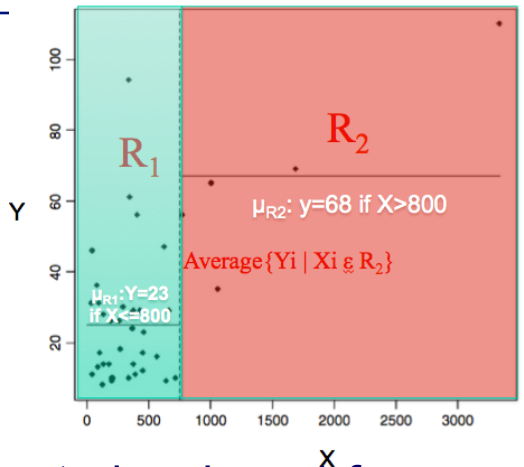
- **The best partition: to minimize the sum of squared error:**

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

- **Consider a single variable problem with a single input variable X_J. Find the threshold value *s* that minimizes the sum of squared error. This problem can be summarized as follows:**

$$\arg\min_{j,s}[\min_{c_1}\sum_{x_i \in R_1(j,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i \in R_2(j,s)}(y_i - c_2)^2]$$

Find best split point for single input and target



Given a real-valued variables X (input) and Y (target)

FindGoodSplitPoint(Input=X, Target=Y) {

#unique values of X; remove duplicates

candidateSplitPoints = unique(X)

Foreach value T in candidateSplitPoints {

Calculate the expected sum of squares error (AKA expected variance) for

$$SSE_{X,T} = \frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} (y_i - \mu_{R_l})^2 + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} (y_i - \mu_{R_R})^2]$$

where μ_{Rj} denotes the mean value of the target variable of tuples satisfying split condition

μ_{RL} for R_L: If $x_i \leq T$

μ_{RR} for R_R: If $x_i > T$

}

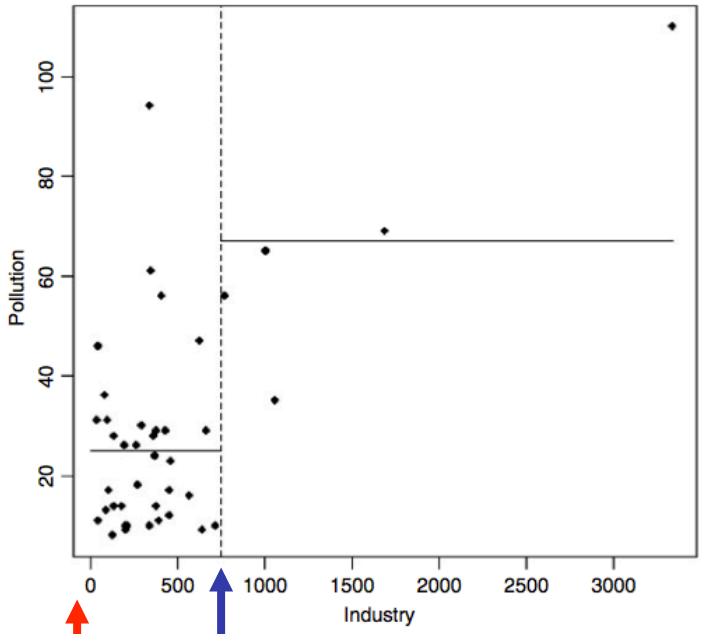
SplitPoint = Select threshold that has minimum SSE

#Return SplitPoint, μ_{Left} , μ_{Right} and the SSE

$$return\left(\underset{T,\mu_{R_1},\mu_{R2}}{\text{argmax}}(SSE_{X,T}),\ SSE_{X,T} \right)$$

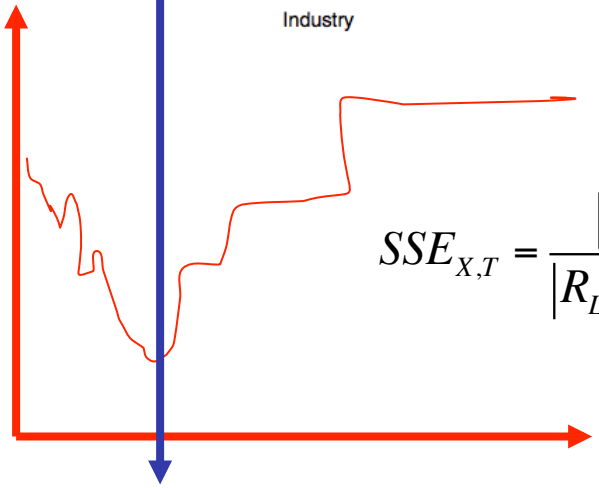
}

Regression Tree: Find optimal first split point



Find optimal first split point to split global region into left and right regions

$SSError_T$ As the split point of Industry is varied



$$SSE_{X,T} = \frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} (y_i - \mu_{R_l})^2 + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} (y_i - \mu_{R_R})^2$$

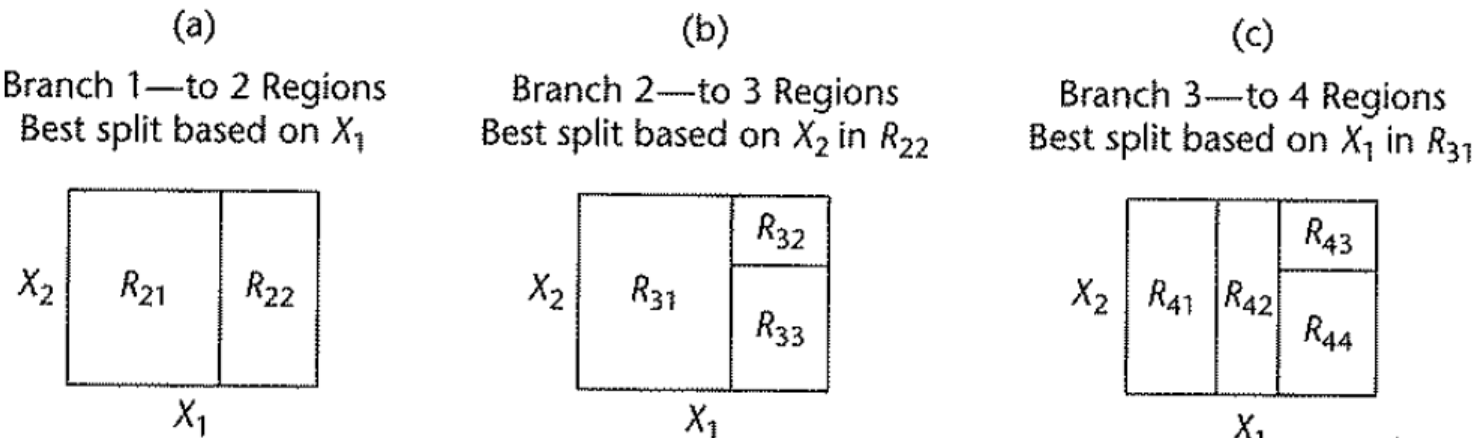
Optimal Split point

Multiple Input variables

- **Given training data X (a matrix where each row is an example and each column is a variable)**
- **Repeat**
  - For each input variable X_j

    SSS_{XjT} , T_j = FindGoodSplitPoint(X_j, Y)
  - Select input variable X_j and its corresponding split that has the minimum SSS_{XjT} and make it the root of the decision tree
  - Route training examples based on the decision ($X_j \leq T_{Xj}$ to left and right node child nodes
  - Recursively apply the above steps to grow the decision tree

FIGURE 11.11
Regression Tree Growth—Two-Predictor Example.



Regression Trees – Grow the Tree

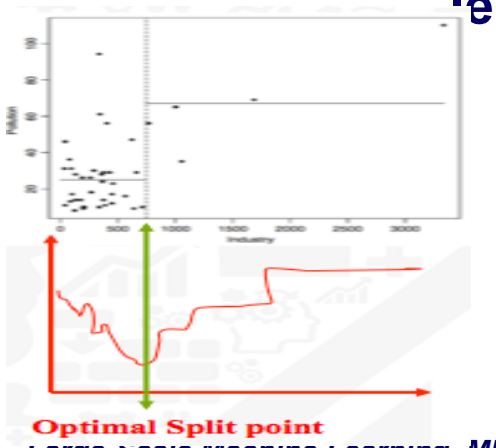
- **The best partition: to minimize the sum of squared error:**

$$\sum_{i=1}^{N} (y_i - f(x_i))^2$$

- **Finding the global minimum is computationally infeasible**
- **Greedy algorithm: at each level choose variable *j* and valu *s* as minimizes the following:**

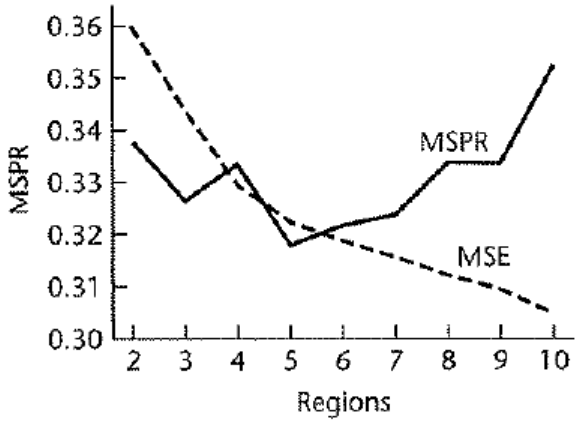
$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} Var(LeftChild) \right]$$

- **The greedy algorithm makes the tree unstable; The error made at the upper level will be propagated to the lower level**

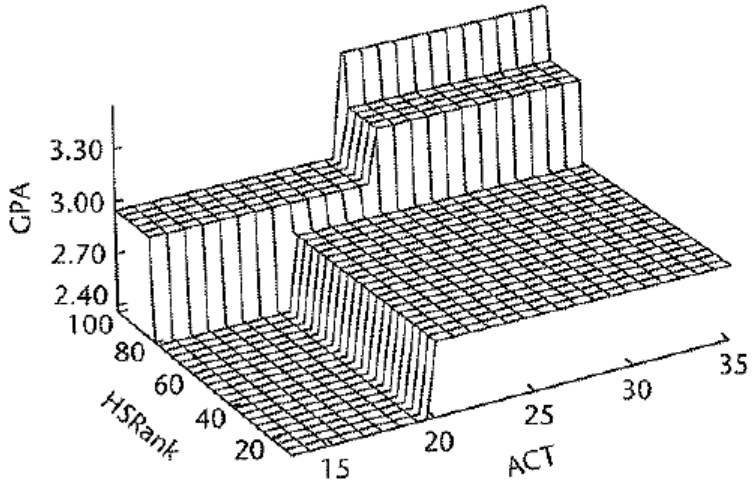


Variable J and Splitpoint s

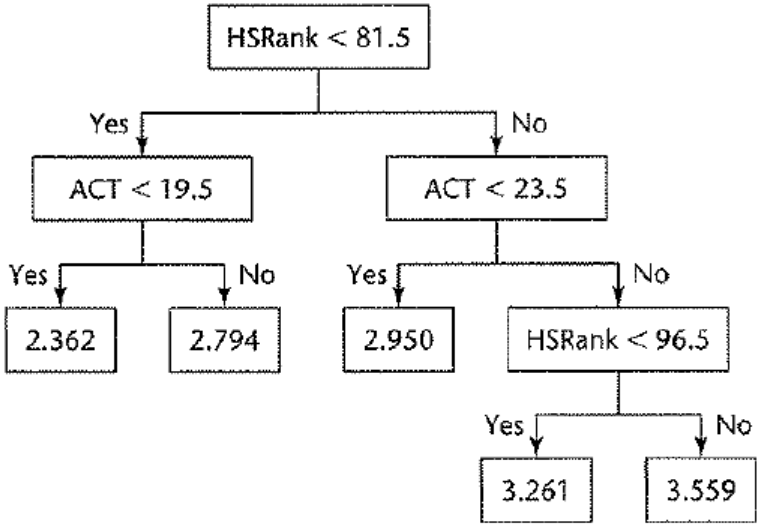
FIGURE 11.12 S-Plus Regression Tree Results—University Admissions Example.



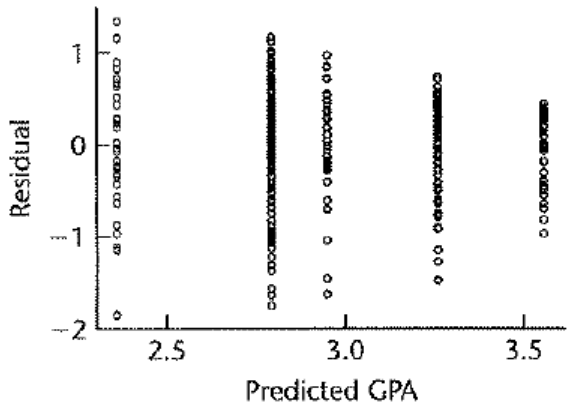
(a)



(b)



(c)



(d)

Source: Kutner, Nachtsheim, and Neter, 4th ed.

- ..

- Starting with a single region -- i.e., all given data
- At the m-th iteration:

for each region R *with enough "impure" data*

for each attribute x_j in R

for each possible split s_j of x_j

record change in score when we partition R into R^l and R^r

Choose (x_j , s_j) giving maximum improvement

Replace R with R^l; add R^r

Classification Tree with Qualitative target
Use variance reduction to select splits

- **Regression purity: variance reduction in the target variable**
- **Designed for quantitative target variables, but works for qualitative target with 2 categories also.**

$$Var(Y) = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu\right)^2$$

- **Example: predict if *loan* will be good or bad based on covariates:**
  - *History* variable {good, bad}
  - *Credit card variable* {yes, no}

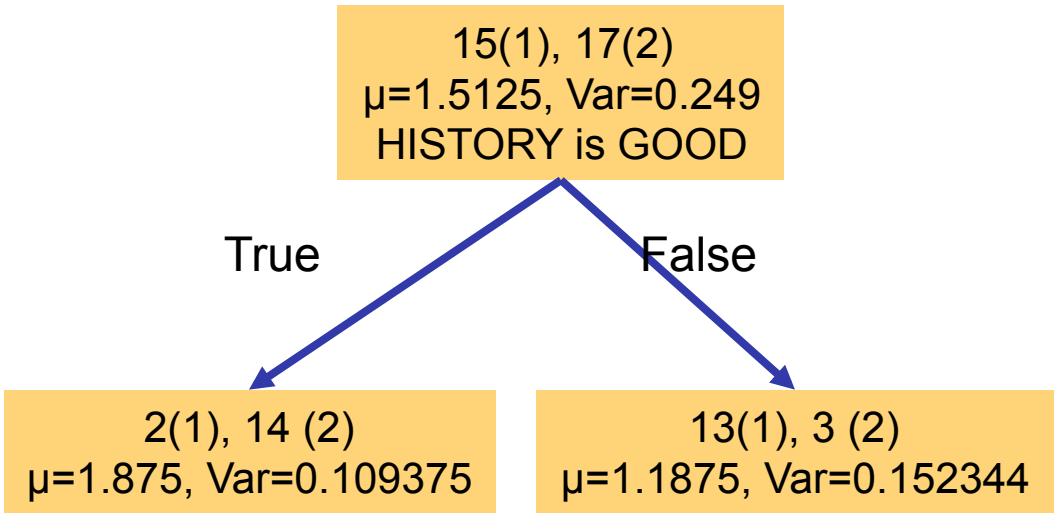
Binary valued inputs

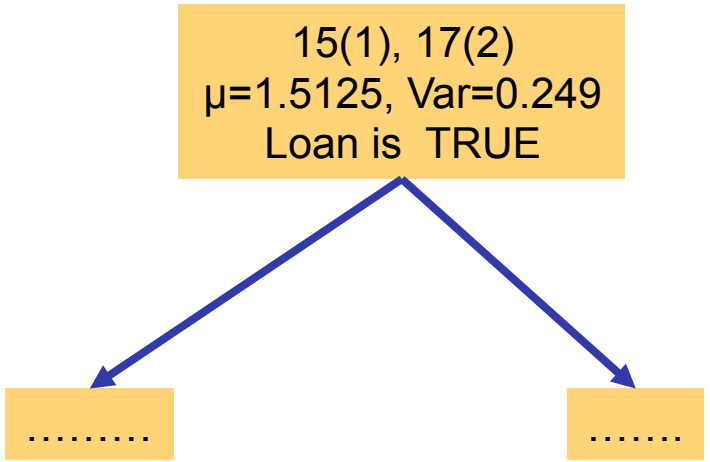
	loan		
	observed		
	bad	good	total
good hist	2	14	16
bad hist	13	3	16
total	15	17	32

Variance as a measure of purity works for binary class problems also

- **Grow tree with either History or Loan variables**
- **Target loan {bad/1, good/2}**
- **Make it real-valued 1, 2**

	loan		
	observed		
	bad	good	total
good hist	2	14	16
bad hist	13	3	16
total	15	17	32





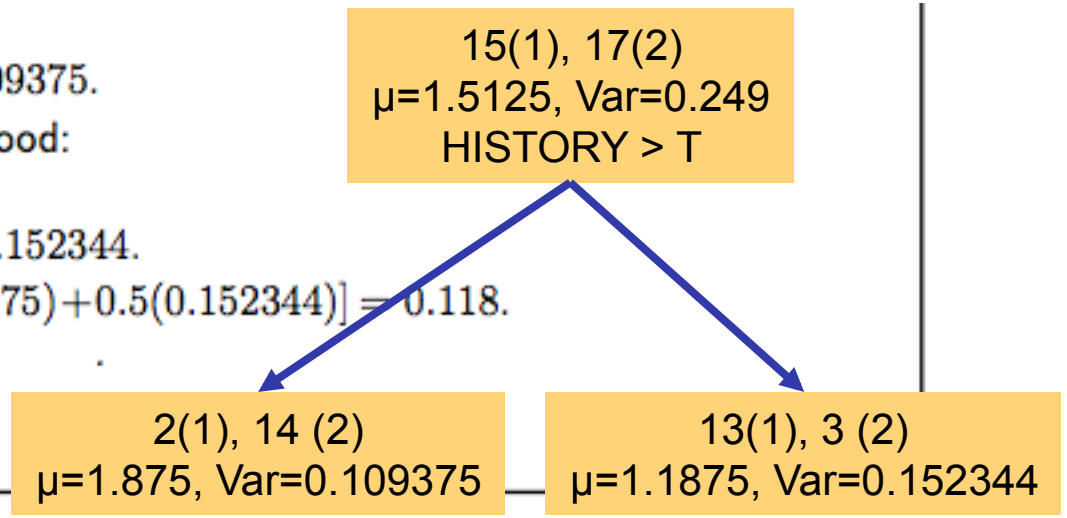
Variance Reduction of 0.118 the target variable Loan when using input variable History

Grow tree with either History or Loan variables where the Target variable is loan {bad/1, good/2}

Regression purity: variance reduction

- Designed for quantitative target variables, but works for qualitative target with 2 categories also.
- Class 9 example on credit risk, history split:
- Parent (root) node has 15 bad loans (target=1) and 17 good loans (target=2)
  - mean: $(15/32)1 + (17/32)2 = 1.53125$;
  - var: $[15(-0.53125)^2 + 17(0.46875)^2]/32 = 0.249$.
- History split first child node, 2 bad, 14 good:
  - mean: $(2/16)1 + (14/16)2 = 1.875$;
  - var: $[2(-0.875)^2 + 14(0.125)^2]/16 = 0.109375$.
- History split second child node, 13 bad, 3 good:
  - mean: $(13/16)1 + (3/16)2 = 1.1875$;
  - var: $[13(-0.1875)^2 + 3(0.8125)^2]/16 = 0.152344$.
- Reduction in variance $= 0.249 - [0.5(0.109375) + 0.5(0.152344)] = 0.118$.

	loan		
	observed		
	bad	good	total
good hist	2	14	16
bad hist	13	3	16
total	15	17	32



Hide this plz

Hide this plz

Quiz: 12.6.1 Grow tree with History or Credit Risk variable

	loan		
	observed		
	bad	good	total
good hist	2	14	16
bad hist	13	3	16
total	15	17	32

	loan		
	observed		
	bad	good	total
Credit Risk is low	3	13	16
Credit Risk is high	12	4	16
total	15	17	32

15(1), 17(2)
μ=1.5125, Var=0.249
HISTORY is GOOD

True

False

2(1), 14 (2)
μ=1.875, Var=0.109375

13(1), 3 (2)
μ=1.1875, Var=0.152344

15(1), 17(2)
μ=1.5125, Var=0.249
Credit Risk is low

$$\text{Reduction in variance} = 0.249 - [0.5(0.109375) + 0.5(0.152344)] = 0.118.$$

Given this data set which variable split should we chose when using a regression purity measure of variance reduction?

(A) Reduction in Variance for Credit Risk Split is 0.07908; therefore we select the History split to grow the decision tree [CORRECT answer]
(B) Reduction in Variance for Credit Risk Split is 0.07908; therefore we select the Credit Risk Split to grow the decision tree
(C) Reduction in Variance for Credit Risk Split is 0.04; therefore we select the History split to grow the decision tree
(D) None of the above

NOTE to self: Hidden stuff under here

F-Test is an alternative measure of purity

Regression purity: F-test

- Designed for quantitative target variables, but works for qualitative target with 2 categories also.
- Class 9 example on credit risk, history split:
- Parent 15/17 bad/good (1/2), mean = 1.53125.
- History split:
  - first child, 2/14, mean = 1.875;
  - second child, 13/3, mean = 1.1875.
- "Between" mean square error = $[16(0.34375)^2+16(-0.34375)^2]/(2-1) = 3.78125$.
- "Within" mean square error = $[2(-0.875)^2+14(0.125)^2 +13(-0.1875)^2+3(0.8125)^2]/(32-2) = 0.139583$.
- F = $3.78125/0.139583 = 27.09$, p-value = 0.00001 (`=FDIST(27.09,1,30)`).
- Better than credit card split with F = 0.133/0.261 = 0.511, p-value = 0.480.

13 / 16

Splitting Criteria for a Continuous (Interval) Target

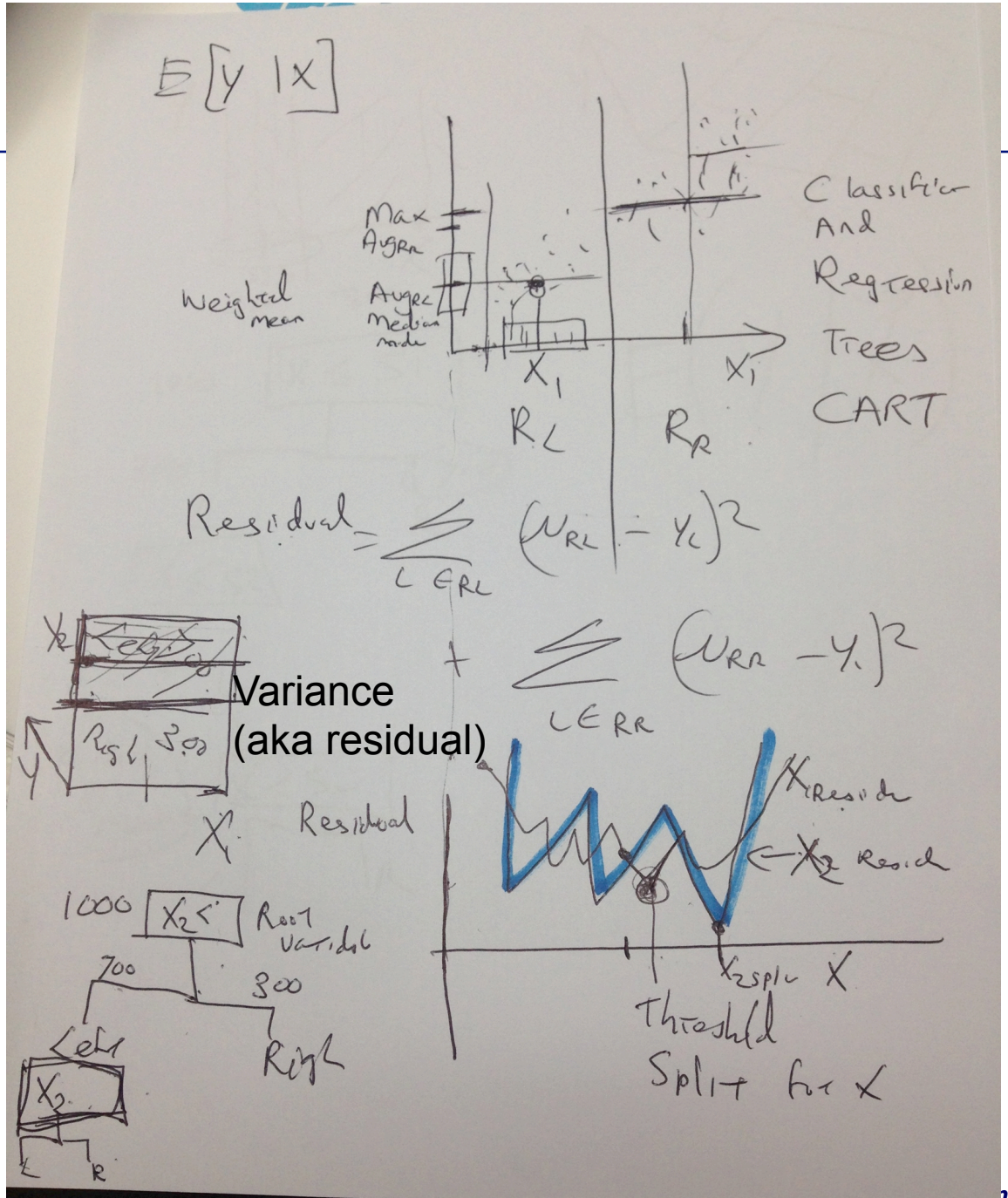
- An *F*-statistic is used to measure the degree of separation of a split for an interval target, such as revenue
- Similar to the sum of squares discussion under multiple regression, the *F*-statistic is based on the ratio of the sum of squares between the groups and the sum of squares within groups, both adjusted for the number of degrees of freedom
- The null hypothesis is that there is no difference in the target mean between the two groups
- As before, the *logworth* of the *p*-value is computed

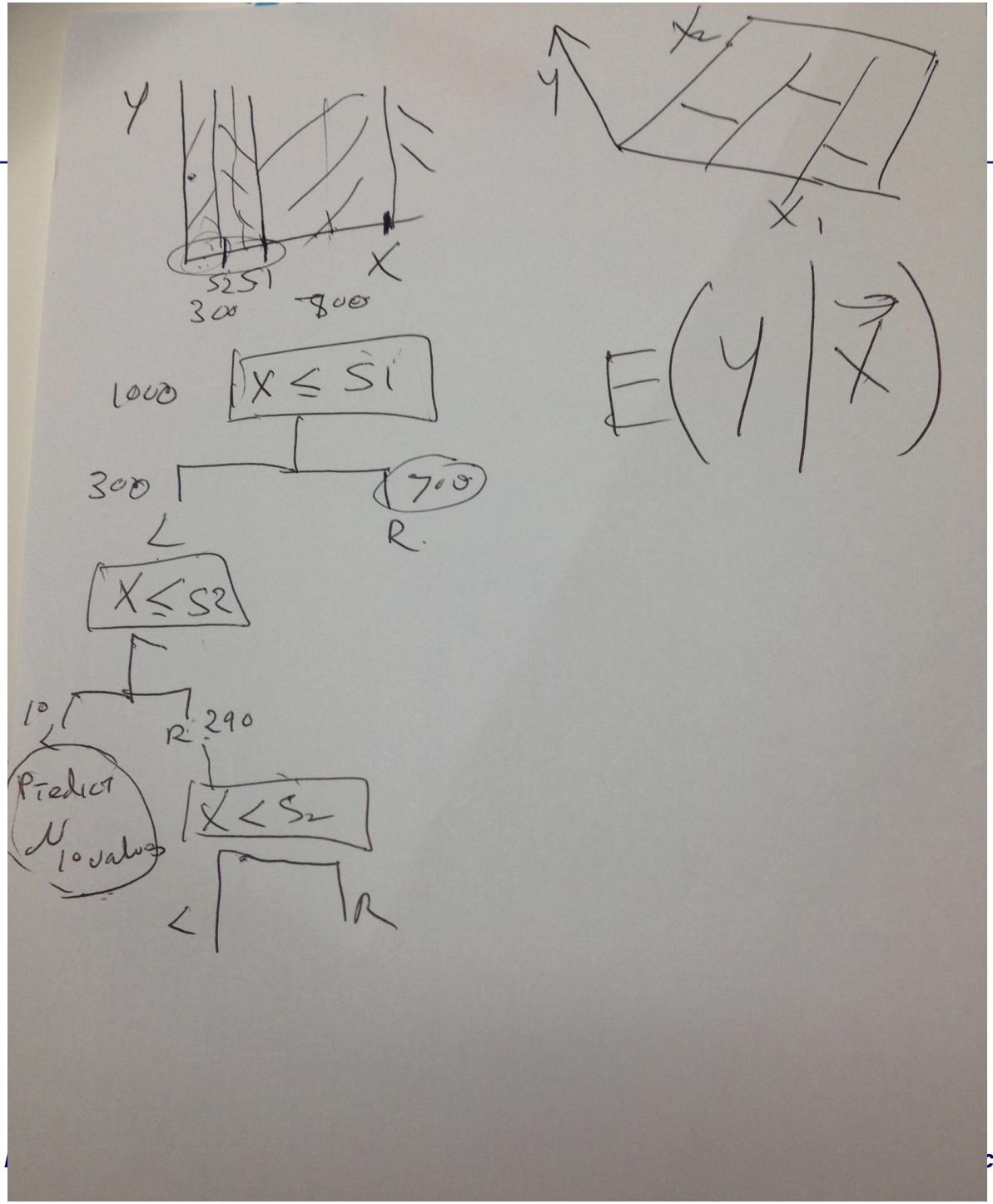
Some Adjustments

- **The more possible splits of an input variable, the less accurate the *p*-value (bigger chance of rejecting the null hypothesis)**
  - If there are *m* splits, the *Bonferroni adjustment* adjusts the *p*-value of the best case by subtracting $\log_{10}(m)$ from the *logworth*
- ***If Time of Kass Adjustment* is set to *before* then the *p*-values of the splits are compared with *Bonferroni adjustment***

Some Adjustments

- **Setting *Split Adjustment* property to *Yes* means that the significance of the *p*-value can be adjusted by the depth of the tree**
  - For example, at the fourth split, a calculate p-value of 0.04 becomes $0.04*2^4 = 0.64$, making the split statistically insignificant
  - This leads to rejecting more splits, limiting the size of the tree
- **Tree growth can also be controlled by setting:**
  - *Leaf Size* property (minimum number of observations in a leaf)
  - *Split Size* property (minimum number of observations to allow a node to be split)
  - *Maximum Depth* property (maximum number of generation of nodes)


E[y|x]
Max
Avgs
Avgs
Median
node
Weighted
mean
X1
X1
RL
RR
Classification
And
Regression
Trees
CART
Residual = Σ (yRL − yi)²
i ∈ RL
+ Σ (yRR − yi)²
i ∈ RR
Variance
(aka residual)
Residual
Threshold
Split for X


y
x
5251
300
800
1000
X ≤ S1
300
700
R
X ≤ S2
10
R: 290
Predict
X < S2
L
R
y
x1
x2
E(y | x)

Decision Trees

Induction Overview (2)

- Score criterion:
  - Regression "loss"
    - Squared error: $L(y, \hat{y}) = (y - \hat{y})^2$ Prediction corresponds to Mean
    - Absolute error: $L(y, \hat{y}) = | y - \hat{y} |$ Prediction corresponds to Median
  - Prediction "risk"
    - Average loss over all predictions – $R(T) = E_{y,\mathbf{x}} L(y, T(\mathbf{x}))$
- Goal: find tree T with lowest prediction risk

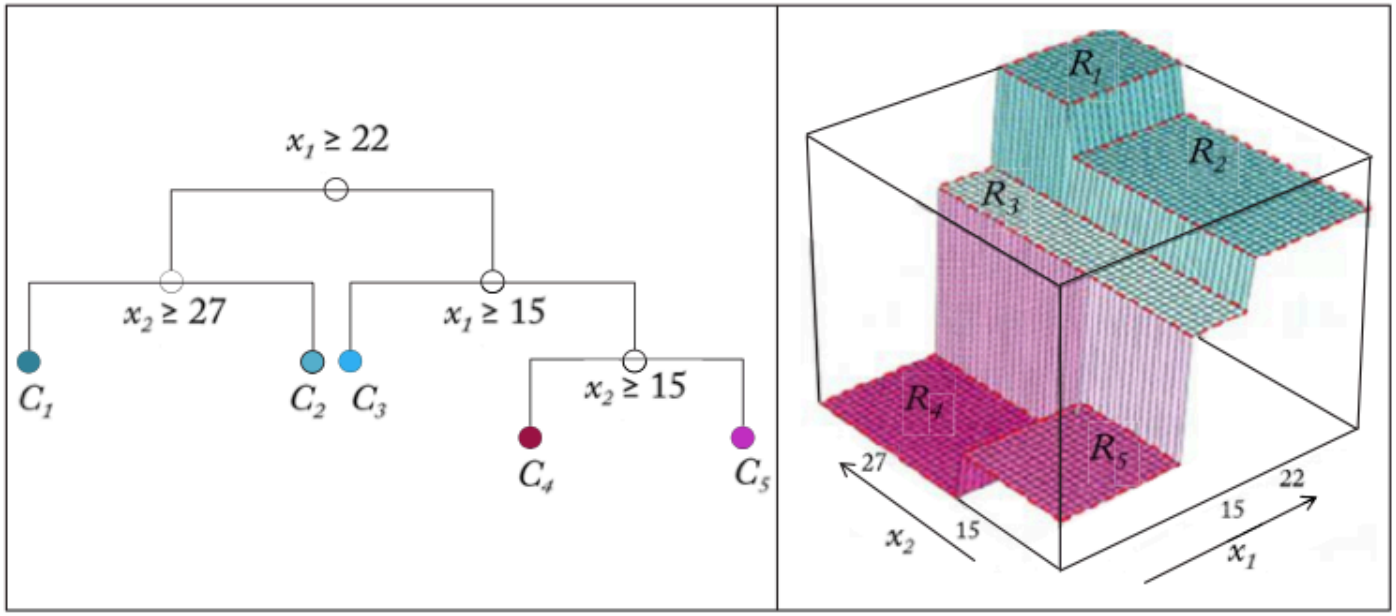
Three Steps in constructing a tree

Algorithm 1 **BuildSubtree**

Require: Node n, Data $D \subseteq D^*$

1: $(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$ (1)
2: **if** $\text{StoppingCriteria}(D_L)$ **then** (2)
3: $n \rightarrow \text{left\_prediction} = \text{FindPrediction}(D_L)$ (3)
4: **else**
5: **BuildSubtree** $(n \rightarrow \text{left}, D_L)$
6: **if** $\text{StoppingCriteria}(D_R)$ **then**
7: $n \rightarrow \text{right\_prediction} = \text{FindPrediction}(D_R)$
8: **else**
9: **BuildSubtree** $(n \rightarrow \text{right}, D_R)$

Regression Tree Learning: learn an m-leaf node tree (m-regions)



m=5 rules
X =input vector
Y target value
yHat = prediction
R_m rules
I_{rm} = indicator function
Only a single region is active for a particular example
T(x) = tree prediction for x

$$\hat{y} = T(\mathbf{x}) = \sum_{m=1}^{M} \hat{c}_m I_{\hat{R}_m}(\mathbf{x})$$

Hyper rectangles

Figure 2.3: Sample regression tree and corresponding surface in input (**x**) space (adapted from (Hastie et al., 2001)).

If we choose to use squared-error loss, then the search problem, finding the tree $T(\mathbf{x})$ with lowest prediction risk, is stated:

$$\left\{ \hat{c}_m, \hat{R}_m \right\}_1^M = \arg\min_{\{c_m, R_m\}_1^M} \sum_{i=1}^{N} [y_i - T(\mathbf{x}_i)]^2$$

$$= \arg\min_{\{c_m, R_m\}_1^M} \sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} c_m I_{R_m}(\mathbf{x}_i) \right]^2$$

learn m-leaf node tree (m-regions) That minimizes this loss function over the m-regions (squared error loss function)

Regression Tree: Loss Functions: L2 (squared error) or L1 (absolute error)

In this section, we look more closely at the algorithm for building decision trees. Figure 2.3 shows an example surface built by a regression tree. It's a piece-wise constant surface: there is a "region" R_m in input space for each terminal node in the tree – i.e., the (hyper) rectangles induced by tree cuts. There is a constant associated with each region, which represents the estimated prediction $\hat{y} = \hat{c}_m$ that the tree is making at each terminal node.

Formally, an M-terminal node tree model is expressed by:

$$\hat{y} = T(\mathbf{x}) = \sum_{m=1}^{M} \hat{c}_m I_{\hat{R}_m}(\mathbf{x})$$

Disjoint

where $I_A(\mathbf{x})$ is 1 if $\mathbf{x} \in A$ and 0 otherwise. Because the regions are disjoint, every possible input $\mathbf{x}$ belongs in a single one, and the tree model can be thought of as the sum of all these regions.

Trees allow for different loss functions fairly easily. The two most used for regression problems are *squared-error* where the optimal constant $\hat{c}_m$ is the mean and the *absolute-error* where the optimal constant is the median of the data points within region R_m (Breiman et al., 1993).

DT Learning: joint optimization of regions and constants?

If we choose to use squared-error loss, then the search problem, finding the tree $T(\mathbf{x})$ with lowest prediction risk, is stated:

$$\left\{ \hat{c}_m, \hat{R}_m \right\}_1^M = \arg\min_{\{c_m, R_m\}_1^M} \sum_{i=1}^{N} [y_i - T(\mathbf{x}_i)]^2$$

$$= \arg\min_{\{c_m, R_m\}_1^M} \sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} c_m I_{R_m}(\mathbf{x}_i) \right]^2$$

To solve, one searches over the space of all possible constants and regions to minimize average loss. Unrestricted optimization with respect to $\{R_m\}_1^M$ is very difficult, so one universal technique is to restrict the shape of the regions (see Figure 2.4).

Joint optimization with respect to $\{\mathbf{R_m}\}_1^{\mathbf{M}}$ and $\{\mathbf{c_m}\}_1^{\mathbf{M}}$, simultaneously, is also extremely difficult, so a greedy iterative procedure is adopted. Start with a global region with all training data and computing the error (variance)

$$\hat{e}(R) = \frac{1}{N} \sum_{\mathbf{x} \in R} \left(y_i - \text{mean}\left(\{y_i\}_1^N \right) \right)^2$$

Then each input variable $\mathbf{x_j}$, and each possible split value $\mathbf{s_j}$ on that particular variable for splitting **R** into $\mathbf{R_L}$ (left region) and $\mathbf{R_r}$ (right region), is considered, and scores $\hat{\mathbf{e}}(\mathbf{R_{Left}})$ and $\hat{\mathbf{e}}(\mathbf{R_{Right}})$ computed.

Regression Tree with continuous inputs

- Assume a single input variable and a single target variable
- Both target and input variables are continuous/ real-valued variables

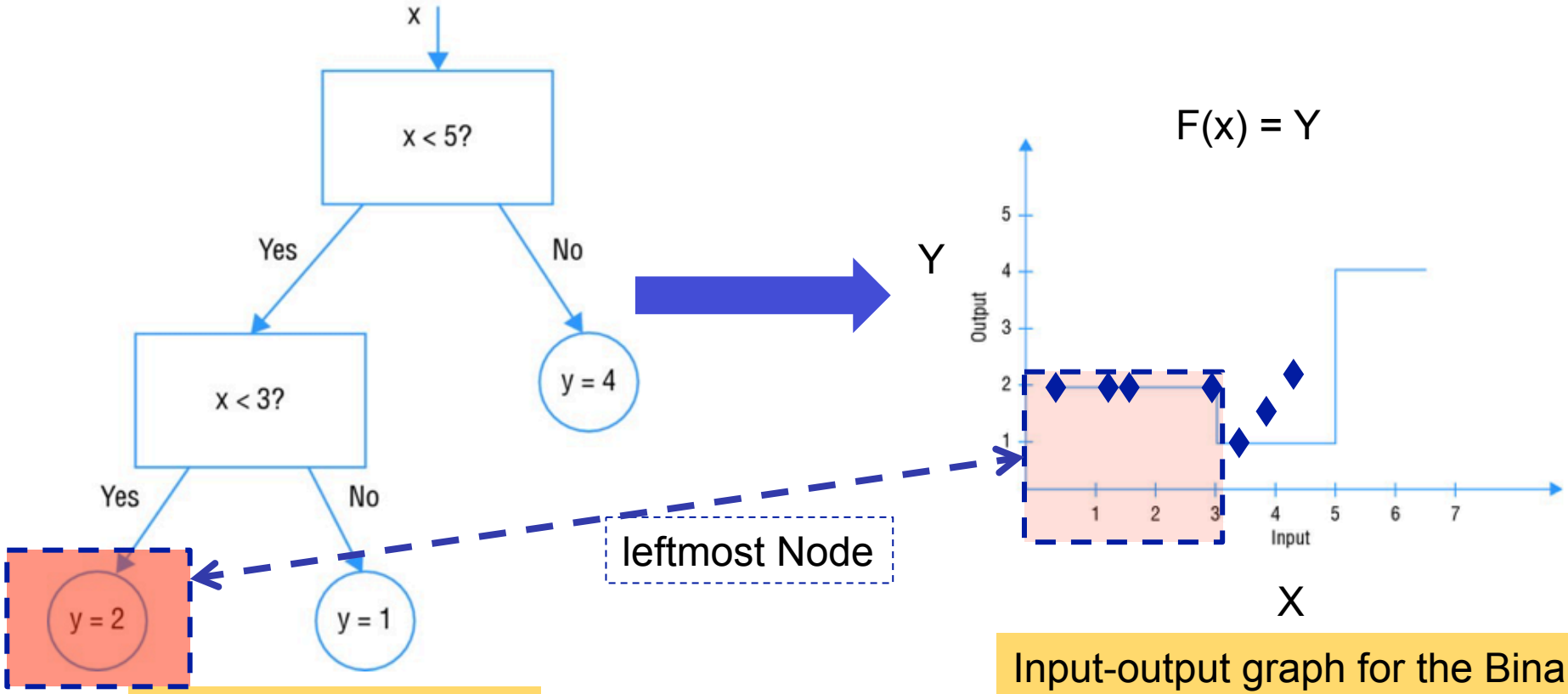
Question on DT

- DTs for regression? For classification?
- Better at regression than classification?
- DTs are really good at both regression and classification

Binary Decision Tree

Slide improvisation

- Classification versus regression



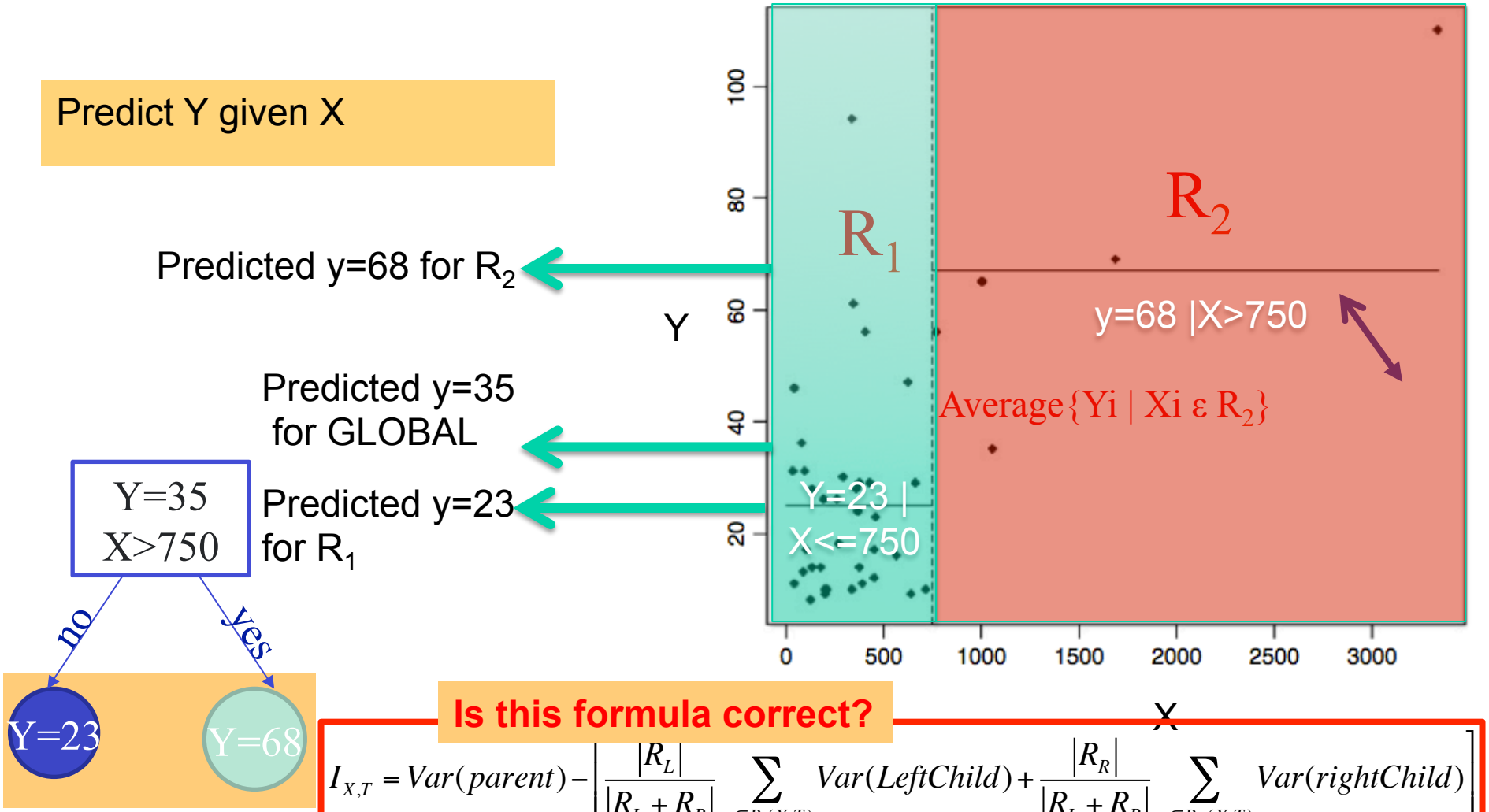
Binary Decision Tree

Input-output graph for the Binary Decision Tree example

Mean value prediction for regression trees with squared error loss

- **Partition the space into *M* regions: R_1, R_2**

Predict Y given X



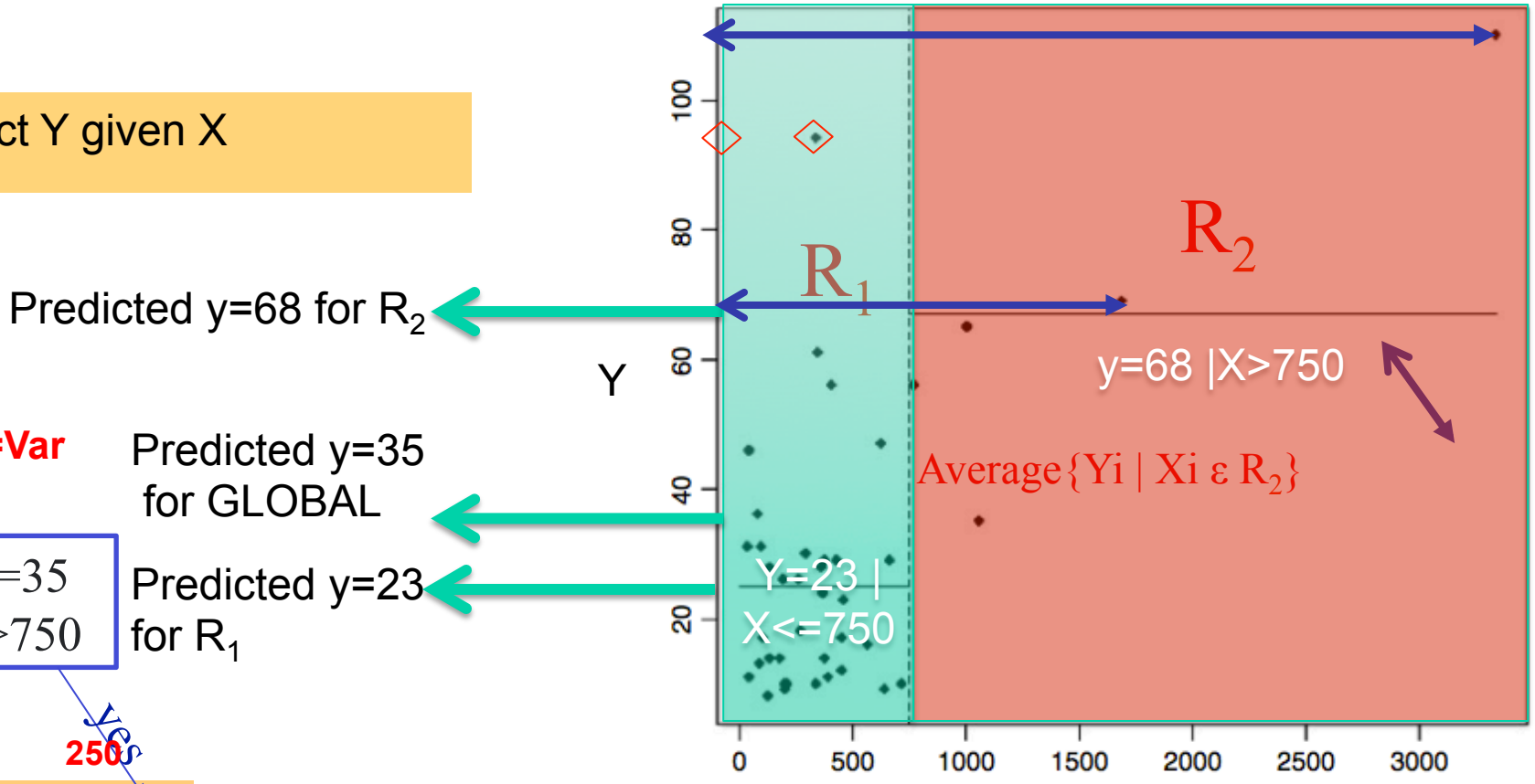
Is this formula correct?

$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} Var(rightChild) \right]$$

Mean value prediction for regression trees with squared error loss

- **Partition the space into *M* regions: R_1, R_2**

Predict Y given X



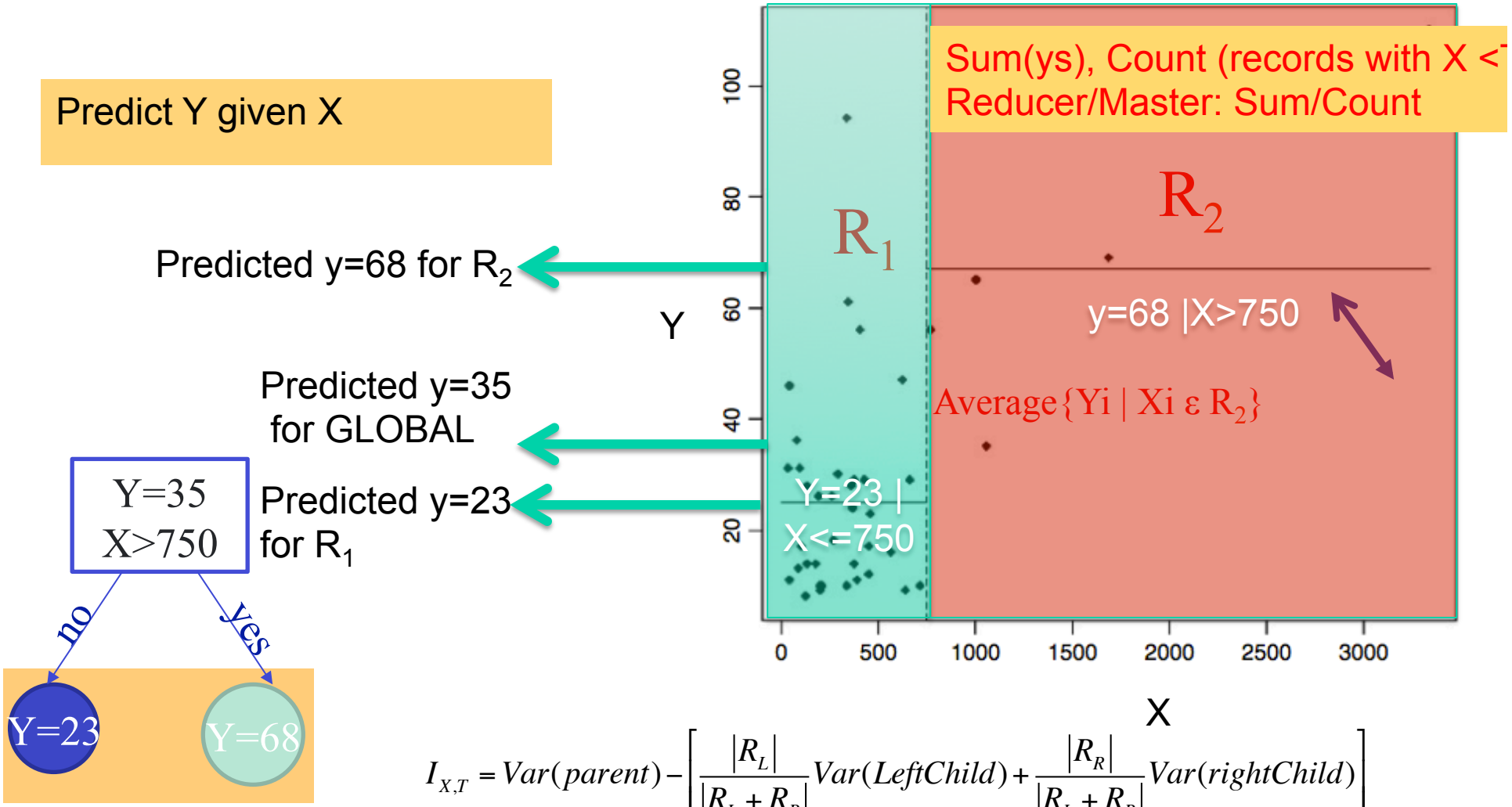
$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} Var(rightChild) \right]$$

- **How can we do DT learning at scale?**
- **Commutative, associative?**

Mean value prediction for regression trees with squared error loss

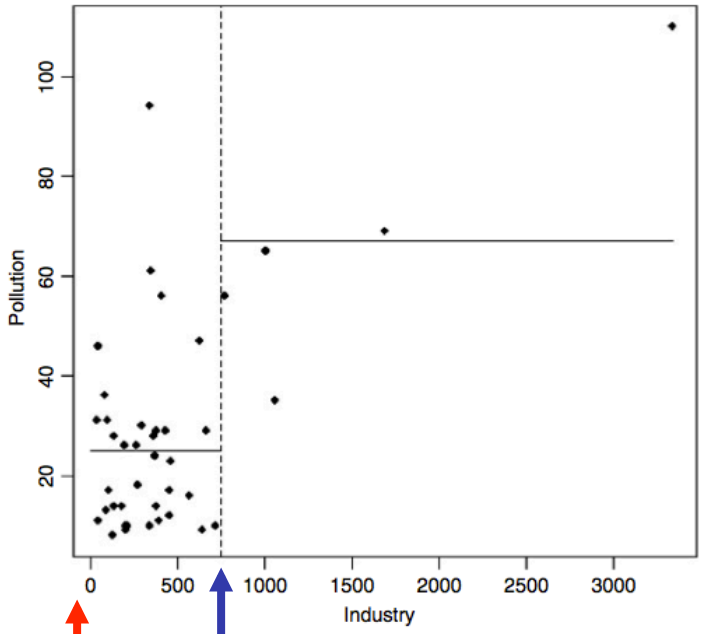
- **Partition the space into *M* regions: R_1, R_2**

Predict Y given X



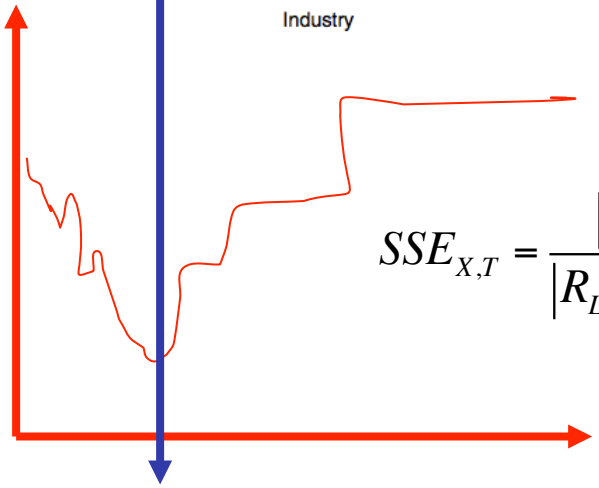
$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} Var(rightChild) \right]$$

Regression Tree: Find optimal first split point



◇ **Find optimal first split point to split global region into left and right regions**

WeightedSSError$_T$
As the split point of
Industry is varied



Optimal Split point

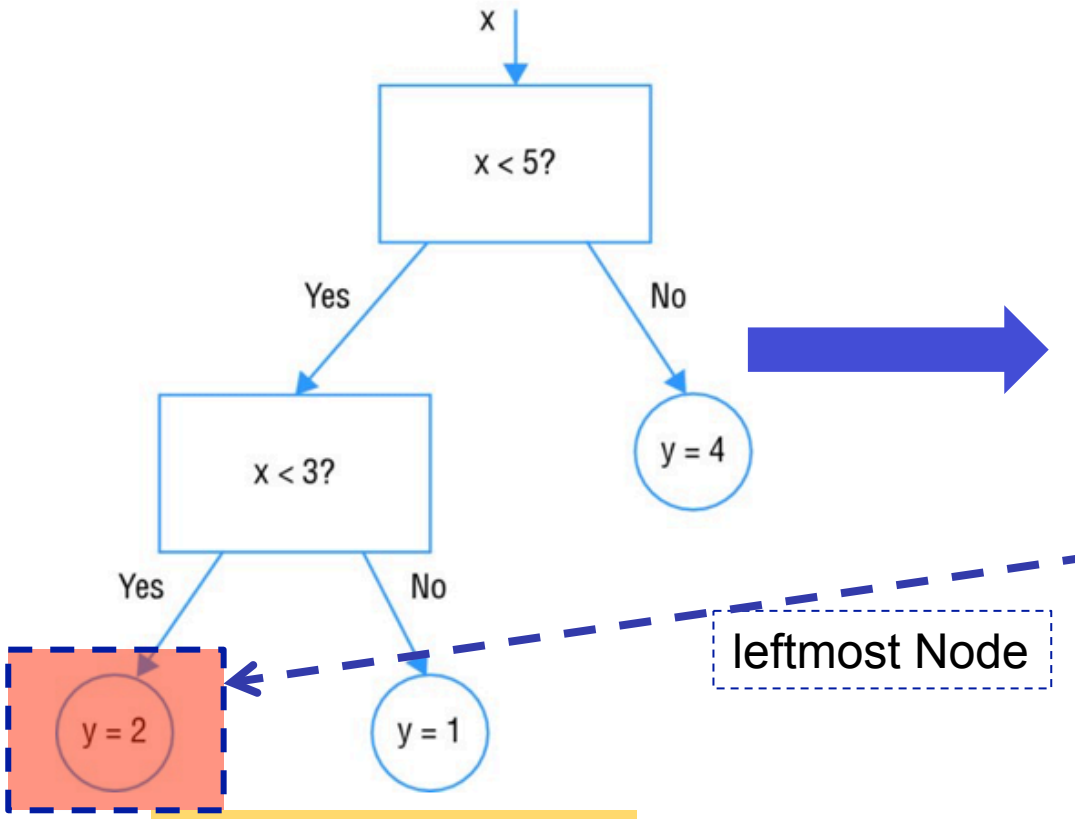
Is this equation correct?

$$SSE_{X,T} = \frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} (y_i - \mu_{R_l})^2 + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} (y_i - \mu_{R_R})^2$$

Binary Decision Tree: Missing Data

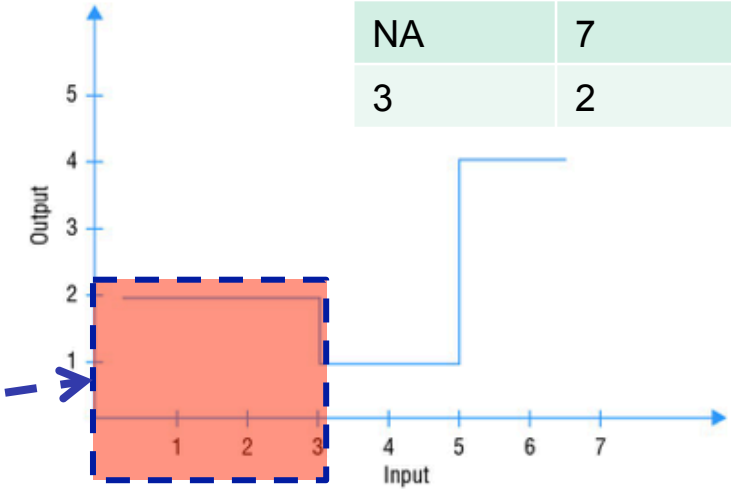
Replace by average
Delete records with missing values

X	Y
1	2
3	2
5	4
6	4
3	1
NA	7
NA	7
3	2



Binary Decision Tree

leftmost Node

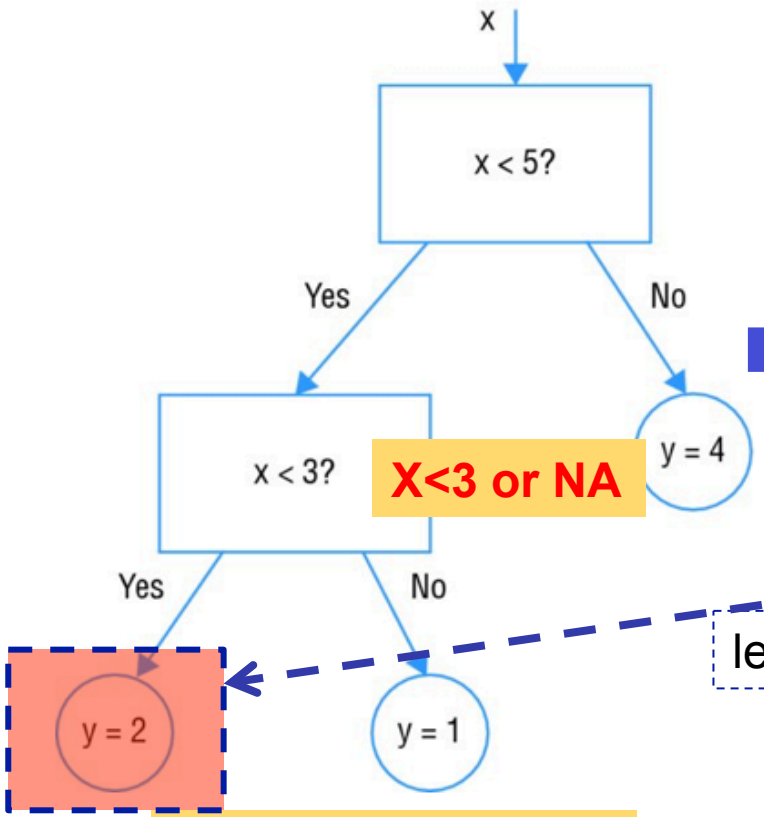


Input-output graph for the Binary Decision Tree example

Binary Decision Tree: Missing Data

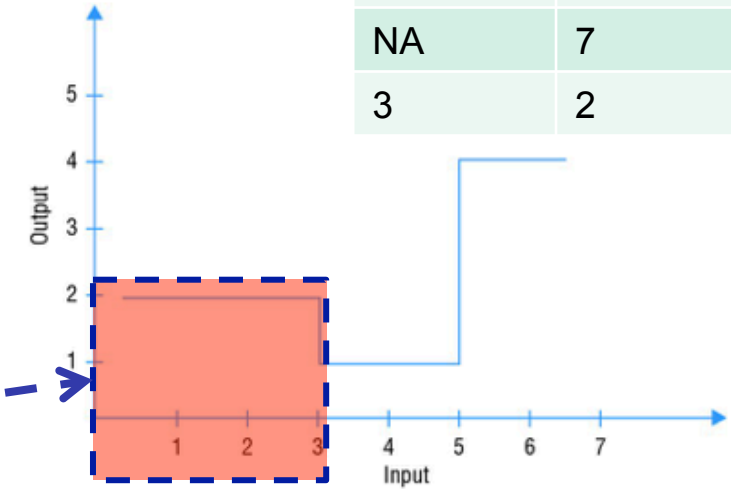
X	Y
1	2
3	2
5	4
6	4
3	1
NA	1.7
NA	7
3	2

X==NA



Binary Decision Tree

leftmost Node



Input-output graph for the Binary Decision Tree example

Classification versus regression Trees

- Tree models where the target variable can take a finite set of values are called classification trees.
- In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

- Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

Input Target

Continuous input variable(Elevation)→ Categorical target

Table 4.8

Dataset for predicting the vegetation in an area with a continuous ELEVATION feature (measured in feet).

ID	STREAM	SLOPE	ELEVATION	VEGETATION
1	false	steep	3,900	chapparal
2	true	moderate	300	riparian
3	true	steep	1,500	riparian
4	false	steep	1,200	chapparal
5	false	flat	4,450	conifer
6	true	steep	5,000	conifer
7	true	steep	3,000	chapparal

Table 4.9

Dataset for predicting the vegetation in an area sorted by the continuous ELEVATION feature.

ID	STREAM	SLOPE	ELEVATION	VEGETATION
2	true	moderate	300	riparian
4	false	steep	1,200	chapparal
3	true	steep	1,500	riparian
7	true	steep	3,000	chapparal
1	false	steep	3,900	chapparal
5	false	flat	4,450	conifer
6	true	steep	5,000	conifer

calculating the information gain when we partition the dataset with ELEVATION using this optimal threshold. Our first task, is to sort the dataset based on the ELEVATION feature. This is shown in Table 4.9[151].

Once the instances have been sorted, we look for adjacent pairs that have different target levels. In Table 4.9[151] we can see that four pairs of adjacent instances have a transition between the target levels, instances $\mathbf{d}_2$ and $\mathbf{d}_4$, $\mathbf{d}_4$ and $\mathbf{d}_3$, $\mathbf{d}_3$ and $\mathbf{d}_7$, and $\mathbf{d}_1$ and $\mathbf{d}_5$. The boundary value between each of these pairs is simply the average of their ELEVATION values:

- the boundary between $\mathbf{d}_2$ and $\mathbf{d}_4$ is $\frac{300 + 1{,}200}{2} = 750$
- the boundary between $\mathbf{d}_4$ and $\mathbf{d}_3$ is $\frac{1{,}200 + 1{,}500}{2} = 1{,}350$
- the boundary between $\mathbf{d}_3$ and $\mathbf{d}_7$ is $\frac{1{,}500 + 3{,}000}{2} = 2{,}250$
- the boundary between $\mathbf{d}_1$ and $\mathbf{d}_5$ is $\frac{3{,}900 + 4{,}450}{2} = 4{,}175$

Partition sets (Part.), entropy, remainder (Rem.), and information gain (Info. Gain) for the candidate ELEVATION thresholds: ⩾750, ⩾1,350, ⩾2,250 and ⩾4,175.

Split by Threshold	Part.	Instances	Partition Entropy	Rem.	Info. Gain
⩾750	$\mathcal{D}_1$	$\mathbf{d}_2$	0.0	1.2507	0.3060
	$\mathcal{D}_2$	$\mathbf{d}_4, \mathbf{d}_3, \mathbf{d}_7, \mathbf{d}_1, \mathbf{d}_5, \mathbf{d}_6$	1.4591		
⩾1,350	$\mathcal{D}_3$	$\mathbf{d}_2, \mathbf{d}_4$	1.0	1.3728	0.1839
	$\mathcal{D}_4$	$\mathbf{d}_3, \mathbf{d}_7, \mathbf{d}_1, \mathbf{d}_5, \mathbf{d}_6$	1.5219		
⩾2,250	$\mathcal{D}_5$	$\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_3$	0.9183	0.9650	0.5917
	$\mathcal{D}_6$	$\mathbf{d}_7, \mathbf{d}_1, \mathbf{d}_5, \mathbf{d}_6$	1.0		
⩾4,175	$\mathcal{D}_7$	$\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_3, \mathbf{d}_7, \mathbf{d}_1$	0.9710	0.6935	0.8631
	$\mathcal{D}_8$	$\mathbf{d}_5, \mathbf{d}_6$	0.0		

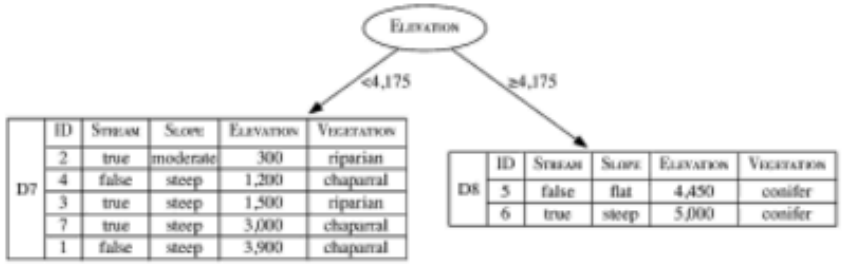


Figure 4.13

The vegetation classification decision tree after the dataset has been split using ELEVATION ⩾ 4,175.

This results in four candidate thresholds: ⩾750, ⩾1,350, ⩾2,250, and ⩾4,175. Table 4.10[152] shows the computation of information gain for a split using each of these thresholds. The threshold ⩾4,175 has the highest information gain of any of the candidate thresholds (0.8631 bits), and this information gain is also higher than the information gain for either of the other two descriptive features. So, we should use ELEVATION ⩾ 4,175 as the test at the root node of the tree, as shown in Figure 4.13[152].

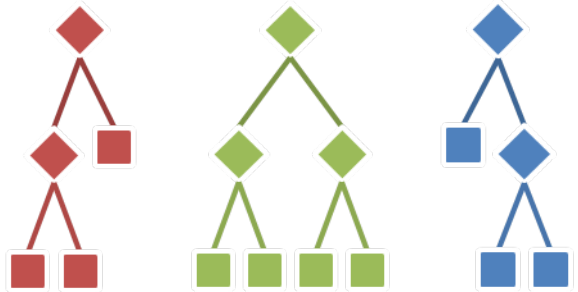
Unlike categorical features, continuous features can be used at multiple points along a path in a decision tree, although the threshold applied to the feature at each of these tests will be different. This is important as it allows multiple splits within a range of a continuous feature to be considered on a path. Consequently, as we build the rest of the tree, we may reuse the ELEVATION

Live Session Outline

- Gradient descent versus non-gradient descent approaches
- Decision Trees
- Classification DTs on discrete variables
- Regression DTs over continuous variables [Optional]
- Ensembles (of decision Trees) [Optional]
- Practical decision trees (single core/Spark)

- **In practice we generally use ensembles of decision trees**

Scaling Up Decision Tree Ensembles

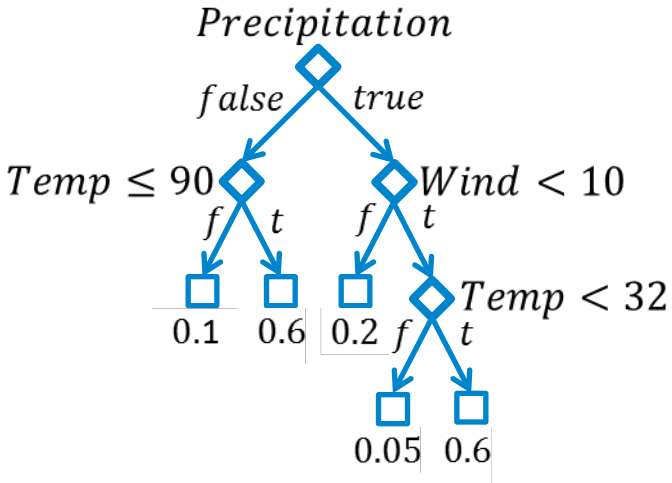


Misha Bilenko (Microsoft)
with Ron Bekkerman (LinkedIn) and John Langford (Yahoo!)

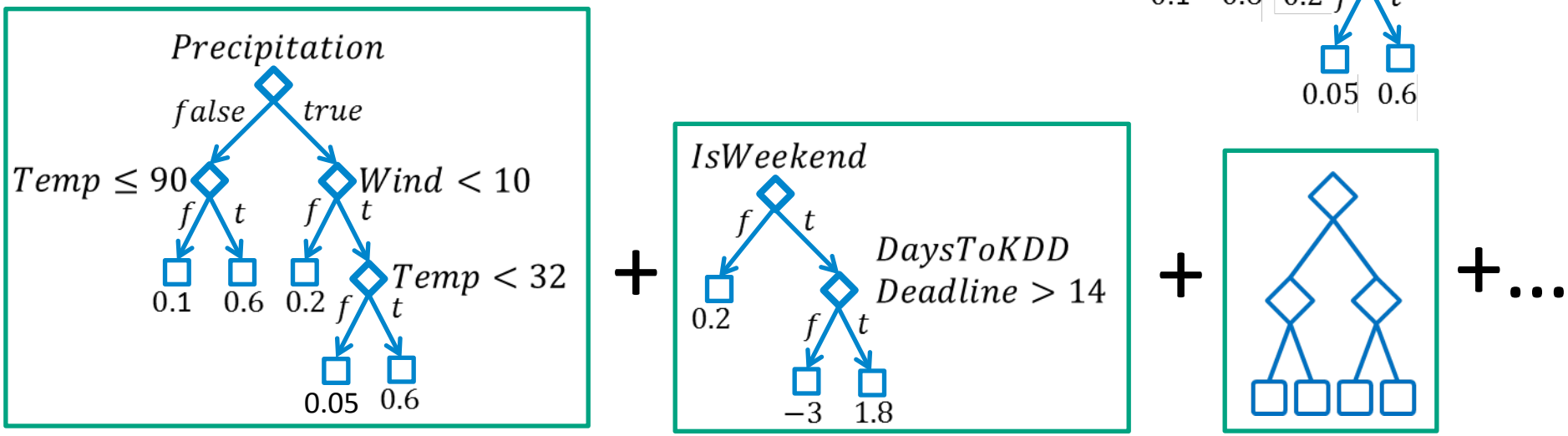
http://hunch.net/~large_scale_survey

Tree Ensembles: Basics

- **Rule-based prediction is natural and powerful (non-linear)**
  - *Play outside: if no rain and not hot, or if snowing but not windy.*
- **Trees hierarchically encode rule-based prediction**
  - Nodes test features to branch
  - Leaves produce predictions
  - Regression trees: numeric outputs



- **Ensembles combine tree predictions**



Boosting Fits an Additive Model

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

where $b(x; \gamma_m) = G_m(x) \in \{-1, 1\}$ (for Adaboost) is like a set of elementary "basis functions"

This model is fit by minimizing a loss function L averaged over the training data set:

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right)$$

Source http://www.stat.ucl.ac.be/

Forward Stagewise Additive Modeling

- **An approximate solution to the minimization problem is obtained via forward stagewise additive modeling (greedy algorithm)**

1. Initialize $f_0(x) = 0$
2. For $m = 1$ to M
   - Compute
     $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$
   - Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Source http://www.stat.ucl.ac.be/

Gradient Boosted Decision Trees

The next seminal work in the theory of Boosting came with the generalization of AdaBoost to any differential loss function. Friedman's Gradient Boosting algorithm (Friedman, J., 2001), with the default values for the algorithm parameters, is summarized in Table 4.8.

Table 4.8: Gradient Boosting as an ISLE-based algorithm.

- General $L(y,\hat{y})$ and y
- $\upsilon = 0.1$
- $\eta = N/2$
- $T_m(\mathbf{x})$: any "weak" learner
- $c_o = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$
- $\{c_m\}_1^M$: "shrunk" sequential partial regression coefficients

$F_0(\mathbf{x}) = c_0$

For $m = 1$ to M {

$$(c_m, \mathbf{p}_m) = \arg\min_{c,\mathbf{p}} \sum_{i \in S_m(\eta)} L\left(y_i, F_{m-1}(\mathbf{x}_i) + c \cdot T(\mathbf{x}_i;\mathbf{p})\right)$$

$T_m(\mathbf{x}) = T(\mathbf{x};\mathbf{p}_m)$

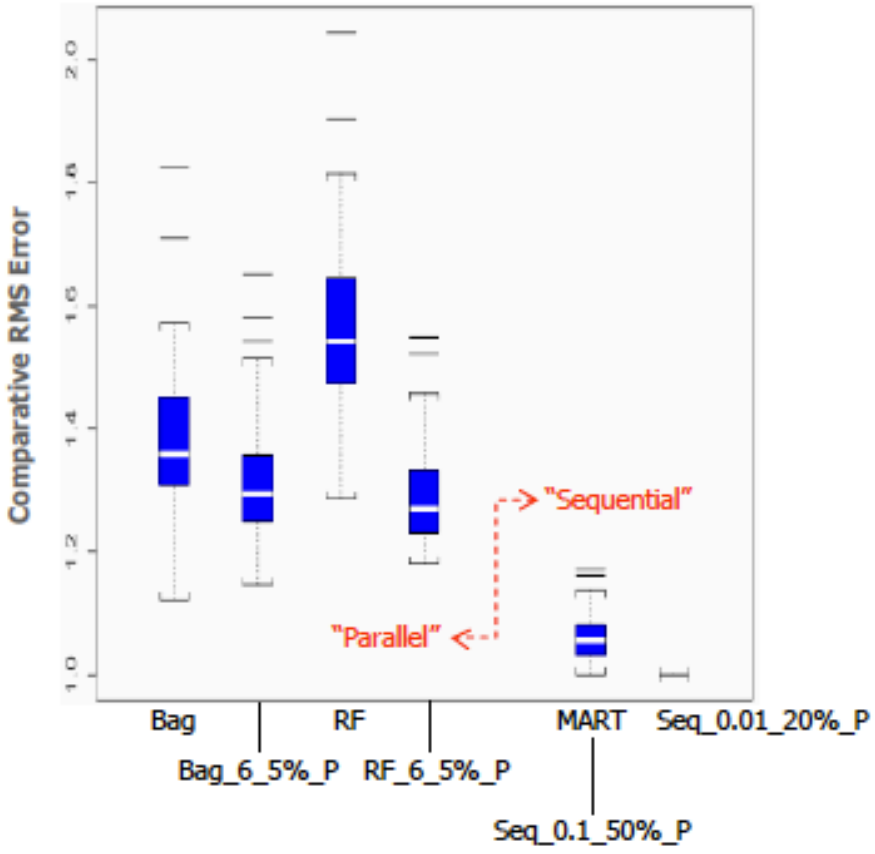
$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \upsilon \cdot c_m \cdot T_m(\mathbf{x})$

}

write $\{(\upsilon \cdot c_m), T_m(\mathbf{x})\}_1^M$

Ensemble Methods

Parallel vs. Sequential Ensembles



- Simulation study with 100 different target functions (Popescu, 2005)
- xxx_6_5%_P : 6 terminal nodes trees
  5% samples without replacement
  Post-processing – i.e., using estimated "optimal" quadrature coefficients
- Seq_η_ν%_P : "Sequential" ensemble
  6 terminal nodes trees
  η: "memory" factor
  ν% samples without replacement
  Post-processing

- Sequential ISLE tend to perform better than parallel ones
  - Consistent with results observed in classical Monte Carlo integration

Tree Ensemble Zoo

- **Different models can define different types of:**
  - Combiner function: voting vs. weighting
  - Leaf prediction models: constant vs. regression
  - Split conditions: single vs. multiple features

- **Examples (small biased sample, some are not tree-specific)**
  - Boosting: AdaBoost [FS97], LogitBoost [F+00], GBM/MART [F01b], BrownBoost [F01a], Transform Regression [SUML11-Ch9]
  - Random Forests [B01]: Random Subspaces [H98], Bagging [B98], Additive Groves [S+07], BagBoo [P+10]
  - Beyond regression and binary classification: RankBoost [F+03], abc-mart [L09], GBRank [Z+08], LambdaMART [SUML11-Ch8]

XGBoost: Learning Ensembles (in Parallel)

dmlc
XGBoost eXtreme Gradient Boosting

build passing | docs latest | license Apache 2.0 | CRAN 0.4-3 | pypi package 0.4a30 | gitter join chat

Documentation | Resources | Installation | Release Notes | RoadMap

XGBoost is an optimized distributed gradient boosting library designed to be highly ***efficient***, ***flexible*** and ***portable***. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting(also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment(Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

What's New

- XGBoost4J: Portable Distributed XGboost in Spark, Flink and Dataflow, see JVM-Package
- Story and Lessons Behind the Evolution of XGBoost
- Tutorial: Distributed XGBoost on AWS with YARN
- XGBoost brick Release

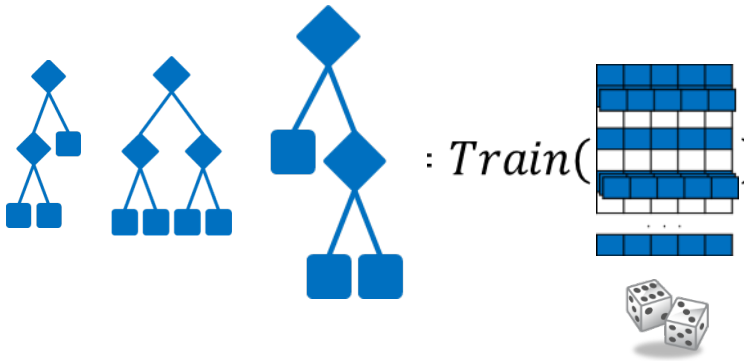
Ask a Question

Tree Ensembles Are Rightfully Popular

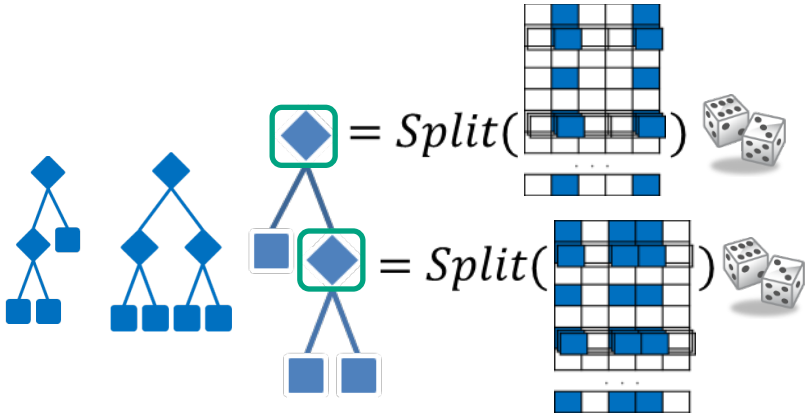
- **State-of-the-art accuracy: web, vision, CRM, bio, …**
- **Efficient at prediction time**
  - Multithread evaluation of individual trees; optimize/short-circuit
- **Principled: extensively studied in statistics and learning theory**
- **Practical**
  - Naturally handle mixed, missing, (un)transformed data
  - Feature selection embedded in algorithm
  - Well-understood parameter sweeps
  - Scalable to extremely large datasets: rest of this section

Naturally Parallel Tree Ensembles

- **No interaction when learning individual trees**
  - **Bagging**: each tree trained on a bootstrap sample of data
  - **Random forests**: bootstrap plus subsample features at each split
  - For large datasets, local data replaces bootstrap -> **embarrassingly parallel**



Bagging tree construction

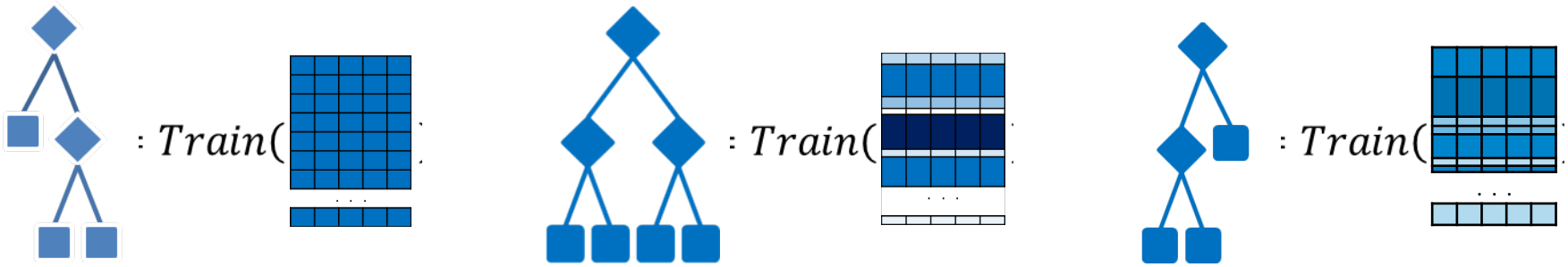


Random forest tree construction

Boosting: Iterative Tree Construction

"Best off-the-shelf classifier in the world" – Breiman

- Reweight examples for each subsequent tree to focus on **errors**



- Numerically: gradient descent in function space
  - Each subsequent tree approximates a step in $-\frac{\partial L}{\partial f}$ direction

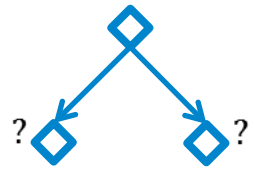
Target for mth model

  - Recompute *target labels* $y^{(m)} = -\left[\frac{\partial L(y,f(x))}{\partial f(x)}\right]_{f(x)=f^{(m-1)}(x)}$
  - Logistic loss: $L(y, f(x)) = \log(1 + \exp(-yf(x))$ $y^{(m)} = \frac{y}{1+\exp(yf(x))}$
  - Squared loss: $L(y, f(x)) = \frac{1}{2}(y - f(x))^2$ $y^{(m)} = y - f(x)$

Efficient Tree Construction

- **Boosting is iterative: scaling up = parallelizing tree construction**

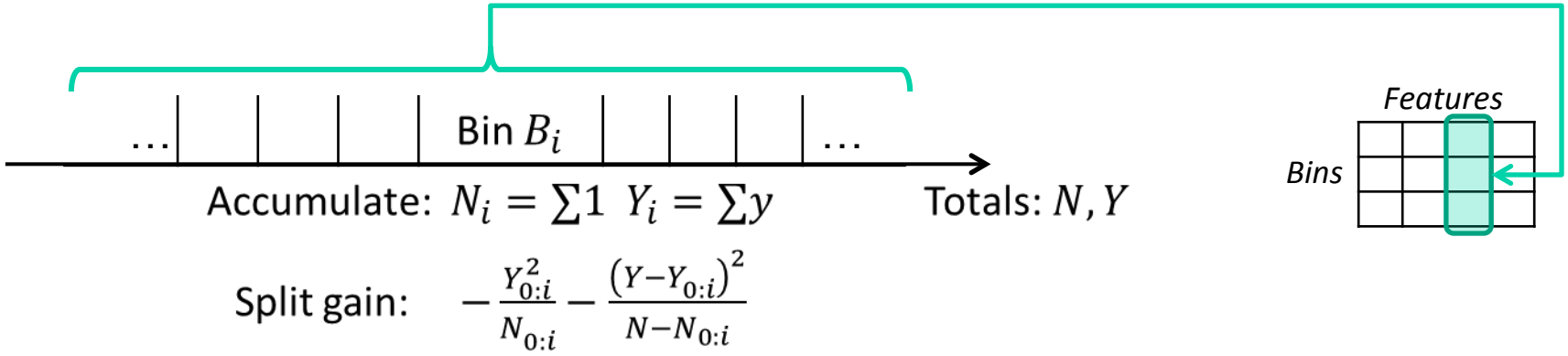
- **For every node: pick best feature to split**
  - For every feature: pick best split-point
    - For every potential split-point: compute gain
      - For every example in current node, add its gain contribution for given split



- **Key efficiency: limiting+ordering the set of considered split points**
  - Continuous features: discretize into bins, splits = bin boundaries
  - Allows computing split values in a single pass over data

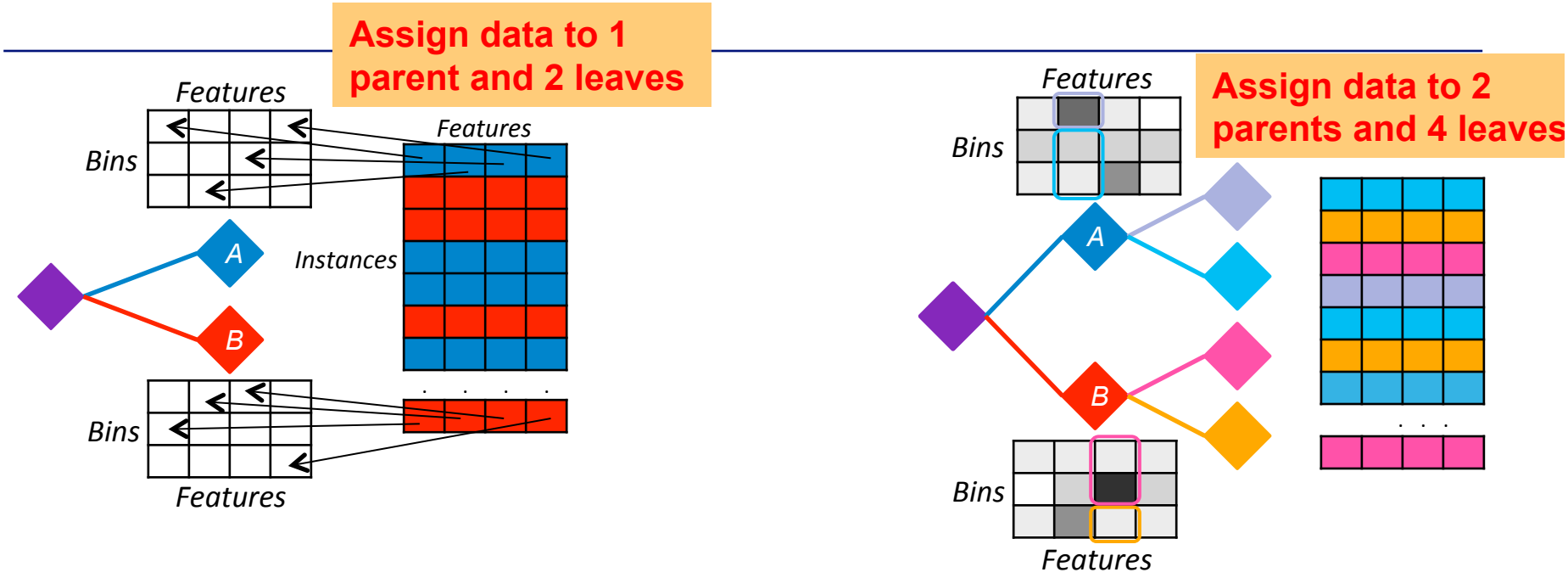
Binned Split Evaluation

- Each feature's range is split into k bins. Per-bin statistics are aggregated in a single pass

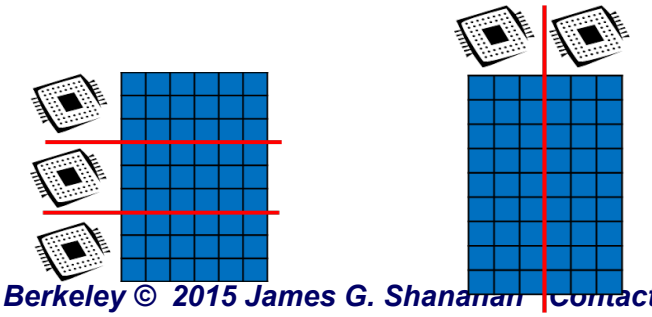


- For each tree node, a two-stage procedure
  - (1) Pass through dataset aggregating node-feature-bin statistics
  - (2) Select split among all (feature,bin) options

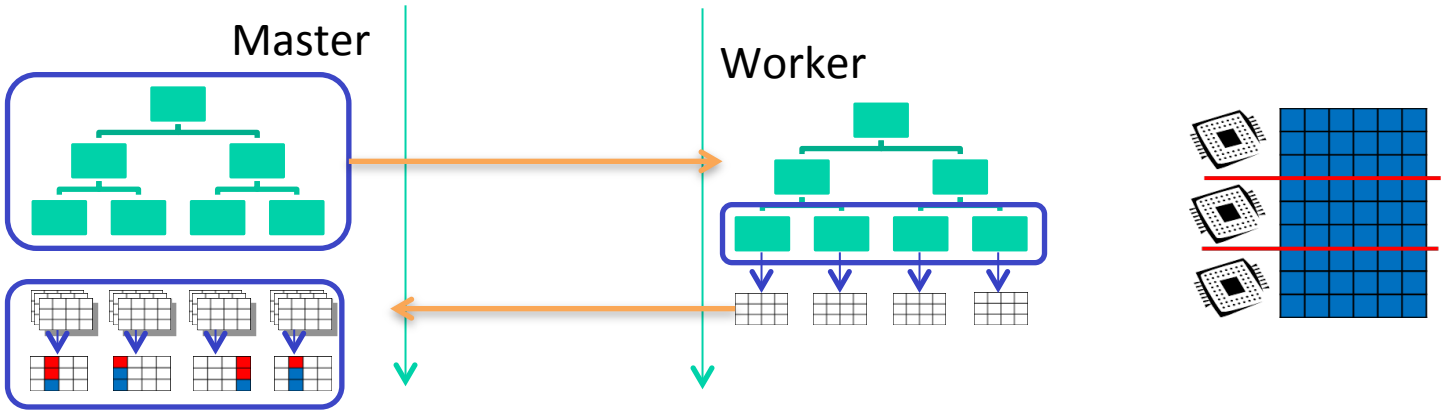
Tree Construction Visualized



- **Observation 1: a single pass is sufficient per tree level**
- **Observation 2: data pass can iterate by-instance or by-feature**
  - Supports horizontally or vertically partitioned data



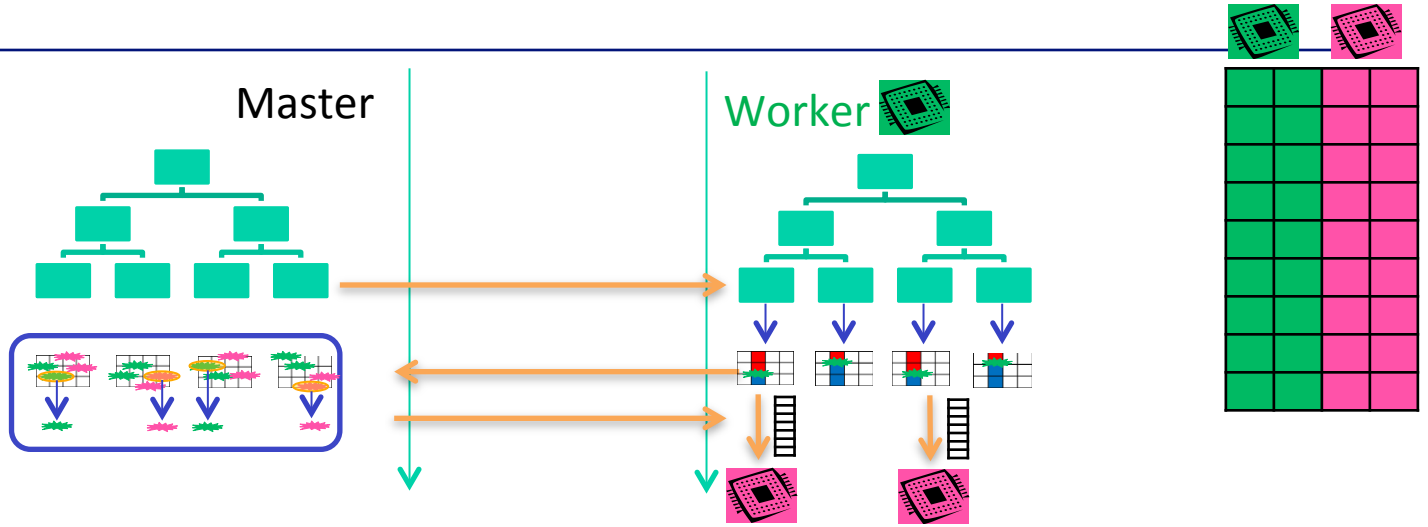
Data-Distributed Tree Construction



- **Master**
  1. Send workers current model and set of nodes to expand
  2. Wait to receive local split histograms from workers
  3. Aggregate local split histograms, select best split for every node

- **Worker**
  2a. Pass through local data, aggregating split histograms
  2b. Send completed local histograms to master

Feature-Distributed Tree Construction



- **Workers maintain per-instance index of current residuals and previous splits**
- **Master**
  1. Request workers to expand a set of nodes
  2. Wait to receive best per-feature splits from workers
  3. Select best feature-split for every node
  4. Request best splits' workers to broadcast per-instance assignments and residuals

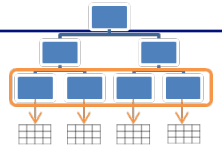
- **Worker**
  2a. Pass through all instances for local features, aggregating split histograms for each node
  2b. Select local features' best splits for each node, send to master

PLANET [SUML11-Ch2]

(Parallel Learner for Assembling Numerous Ensemble Trees)

- **Platform: MapReduce (Google), RPC (!)**
- **Data Partitioning: Horizontal (by-instance)**
- **Binning: during initialization**
  - Single-pass approximate quantiles via [Manku et al. '98]
- **Modulate between distributed and single-machine tree construction**
  - *MapReduceQueue*: distributed tree construction
  - *InMemoryQueue*: single-worker tree construction (when node small enough)

PLANET: ExpandNodes Mapper



- Go through examples, aggregating summaries $(\sum y, \sum 1)$ for every:
  - Node: total statistics for the node (applies for all features)
  - Split: for every feature-bin

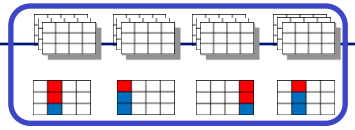
$Mapper.Map(\langle x, y\rangle, model\ M, \text{nodes } N)$

1: $n = \text{TraverseTree(M, } \mathbf{x})$
2: **If** $n \in N$ **then**
3: $\quad \text{agg\_tup}_n \leftarrow y$
4: $\quad$ **For each** $X \in \mathcal{X}$ **do**
5: $\quad\quad v = \text{Value on } X \text{ in } \mathbf{x}$
6: $\quad$ **For each** Split point s of X s.t. $s < v$ **do**
7: $\quad\quad T_{n,X}[s] \leftarrow y$

$Mapper.Finalize()$

1: **For each** $n \in N$ **do**
2: $\quad \text{Output to all reducers}(n, \text{agg\_tup}_n)$
3: $\quad$ **For each** $X \in \mathcal{X}$ **do**
4: $\quad$ **For each** Split point s of X **do**
5: $\quad\quad \text{Output}((n, X, s), T_{n,X}[s])$

PLANET: ExpandNodes Reducer



- **Receives (presorted)**
  - (a) Totals for every node
  - (b) Feature-split histogram for every node
- **Adds up histograms, finds best split among all possible ones.**
- **Emits best split found for given node**

Input: Key k,Value Set V

1: **If** $k == n$ **then**
2: *// Aggregate agg_tup$_n$'s from mappers by pre-sorting.*
3: $\text{agg\_tup}_n = \text{Aggregate}(V)$
4: **Else** // $k == n, X, s$
5: *// Split on ordered feature*
6: $\text{agg\_tup}_{left} = \text{Aggregate}(V)$
7: $\text{agg\_tup}_{right} = \text{agg\_tup}_n$ - agg_tup_{left}
8: $\text{UpdateBestSplit}(S[n], X, s, \text{agg\_tup}_{left}, \text{agg\_tup}_{right})$

PLANET: Master (Controller)

- **Master (Controller)**
  - Creates and sends jobs to *MRQueue* or *InMemoryQueue*
  - Aggregates best splits for each node, updates model
- **Engineering is non-trivial, impactful**
  - MapReduce per-worker setup and tear-down costs are high
    - Forward-schedule: pre-allocate Mappers and Reducers, standby
    - Controller uses RPC to send jobs
  - Categorical attributes
    - Specially handled (different MR keys; still linear using the "Breiman trick")
    - Evaluation speedup: fingerprint, store signature with each node

Ensemble Learning in MapReduce
Simple modification

predictions (z). For a given training record $(\mathbf{x}, y)$, the residual prediction for tree k is $z = y - F_{k-1}(\mathbf{x})$ for a regression problem, and $z = y - \frac{1}{1+exp(-F_{k-1}(\mathbf{x}))}$ for a classification problem. The boosting process is initialized by setting F_0 as some aggregate defined over the Y values in the training dataset. Abstracting out the details, we need three main features in our framework to build boosted models.

- Building multiple trees: Extending the Controller to build multiple trees is straightforward. Since the Controller manages tree induction by reducing the process to repeated node expansion, the only change necessary for constructing a boosted model is to push the root node for tree k onto the MR after tree $k-1$ is completed.

- Residual computation: Training trees on residuals is simple since the current model is sent to every Map Reduce job in full. If the mapper decides to use a training record as input to a node, it can compute the current model's prediction, and hence the residual.

- Sampling: Each tree is built on a sample of D^*. Dappers compute a hash of a training record's id and the tree id. Records hashing into a particular range are used for constructing the tree. This hash-based sampling guarantees that the same sample will be used for all nodes in a tree, but different samples of D^* will be used for different trees.

Build multiple trees
Broadcast ensemble to Task nodes for the spliting map-reduce job

Compute residual on the fly in the map-reduce job for spliting

Hash record id and tree id: records hashing in a particular range are used for building the tree

Live Session Outline

- Gradient descent versus non-gradient descent approaches
- Decision Trees
- Classification DTs on discrete variables
- Regression DTs over continuous variables [Optional]
- Ensembles (of decision Trees) [Optional]
- Practical decision trees (single core/Spark)

Practical guide to DTs on single core machines

- **Practical guide to DTs on single core machines**
  - https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/
- **DTs and Ensembles**
  - Spark
  - XGboost

Practical guide to DTs on single core machines

A Complete Tutorial on Tree Based Modeling from Scratch (in R & Python)

https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/

1. What is a Decision Tree? How does it work
2. Regression Trees vs Classification Trees
3. How does a tree decide where to split?
4. What are the key parameters of model building and how can we avoid over-fitting in decision trees?
5. Are tree based models better than linear models?
6. Working with Decision Trees in R and Python
7. What are the ensemble methods of trees based model?
8. What is Bagging? How does it work?
9. What is Random Forest ? How does it work?
10. What is Boosting ? How does it work?
11. Which is more powerful: GBM or Xgboost?
12. Working with GBM in R and Python
13. Working with Xgboost in R and Python
14. Where to Practice ?

DTs and Ensembles in Spark

- https://spark.apache.org/docs/latest/ml-classification-regression.html
  - Decision trees
    - Inputs and Outputs
      - Input Columns
      - Output Columns
  - Tree Ensembles
    - Random Forests
      - Inputs and Outputs
        - Input Columns
        - Output Columns (Predictions)
    - Gradient-Boosted Trees (GBTs)
      - Inputs and Outputs
        - Input Columns
        - Output Columns (Predictions)

Decision trees

Decision trees and their ensembles are popular methods for the machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions. Tree ensemble algorithms such as random forests and boosting are among the top performers for classification and regression tasks.

The `spark.ml` implementation supports decision trees for binary and multiclass classification and for regression, using both continuous and categorical features. The implementation partitions data by rows, allowing distributed training with millions or even billions of instances.

Users can find more information about the decision tree algorithm in the MLlib Decision Tree guide. The main differences between this API and the original MLlib Decision Tree API are:

- support for ML Pipelines
- separation of Decision Trees for classification vs. regression
- use of DataFrame metadata to distinguish continuous and categorical features

The Pipelines API for Decision Trees offers a bit more functionality than the original API.
In particular, for classification, users can get the predicted probability of each class (a.k.a. class conditional probabilities); for regression, users can get the biased sample variance of prediction.

Ensembles of trees (Random Forests and Gradient-Boosted Trees) are described below in the Tree ensembles section.

Inputs and Outputs

We list the input and output (prediction) column types here. All output columns are optional; to exclude an output column, set its corresponding Param to an empty string.

Refer to the `DecisionTree` Python docs and `DecisionTreeModel` Python docs for more details on the API.

```python
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.
data = MLUtils.loadLibSVMFile(sc, 'data/mllib/sample_libsvm_data.txt')
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a DecisionTree model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
model = DecisionTree.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
                                     impurity='gini', maxDepth=5, maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification tree model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "target/tmp/myDecisionTreeClassificationModel")
sameModel = DecisionTreeModel.load(sc, "target/tmp/myDecisionTreeClassificationModel")
```

https://spark.apache.org/docs/latest/mllib-decision-tree.html

Find full example code at "examples/src/main/python/mllib/decision_tree_classification_example.py" in the Spark repo.

XGBoost: Learning Ensembles (in Parallel)

dmlc
XGBoost eXtreme Gradient Boosting

build passing | docs latest | license Apache 2.0 | CRAN 0.4-3 | pypi package 0.4a30 | gitter join chat

Documentation | Resources | Installation | Release Notes | RoadMap

XGBoost is an optimized distributed gradient boosting library designed to be highly ***efficient***, ***flexible*** and ***portable***. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting(also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment(Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

What's New

- XGBoost4J: Portable Distributed XGboost in Spark, Flink and Dataflow, see JVM-Package
- Story and Lessons Behind the Evolution of XGBoost
- Tutorial: Distributed XGBoost on AWS with YARN
- XGBoost brick Release

Ask a Question

XGBoost: Learning Ensembles (in Parallel)

- https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/

13. Working with XGBoost in R and Python

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm. It's feature to implement parallel computing makes it at least **10 times faster** than existing gradient boosting implementations. It supports various objective functions, including regression, classification and ranking.

R Tutorial: For R users, this is a complete tutorial on XGboost which explains the parameters along with codes in R. Check Tutorial.

Python Tutorial: For Python users, this is a comprehensive tutorial on XGBoost, good to get you started. Check Tutorial.

- **End of slides**