Decision Trees for Classification and Regression, PLUS Ensembles

[1]Church and Duncan Group,

[2]Information School, UC Berkeley

[3]School of Informatics, Computing and Engineering, Indiana University

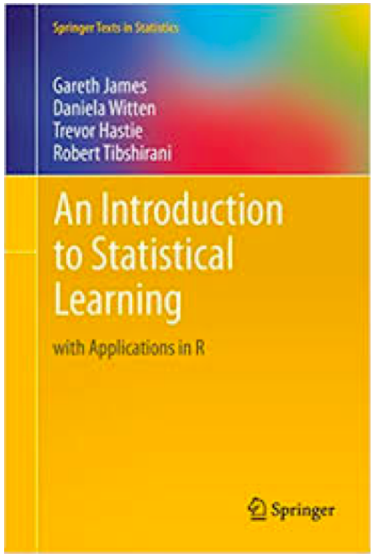
EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

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)**

- **SKLearn**
  - http://scikit-learn.org/stable/modules/tree.html

Reading Material



An Introduction to Statistical Learning with Applications in R

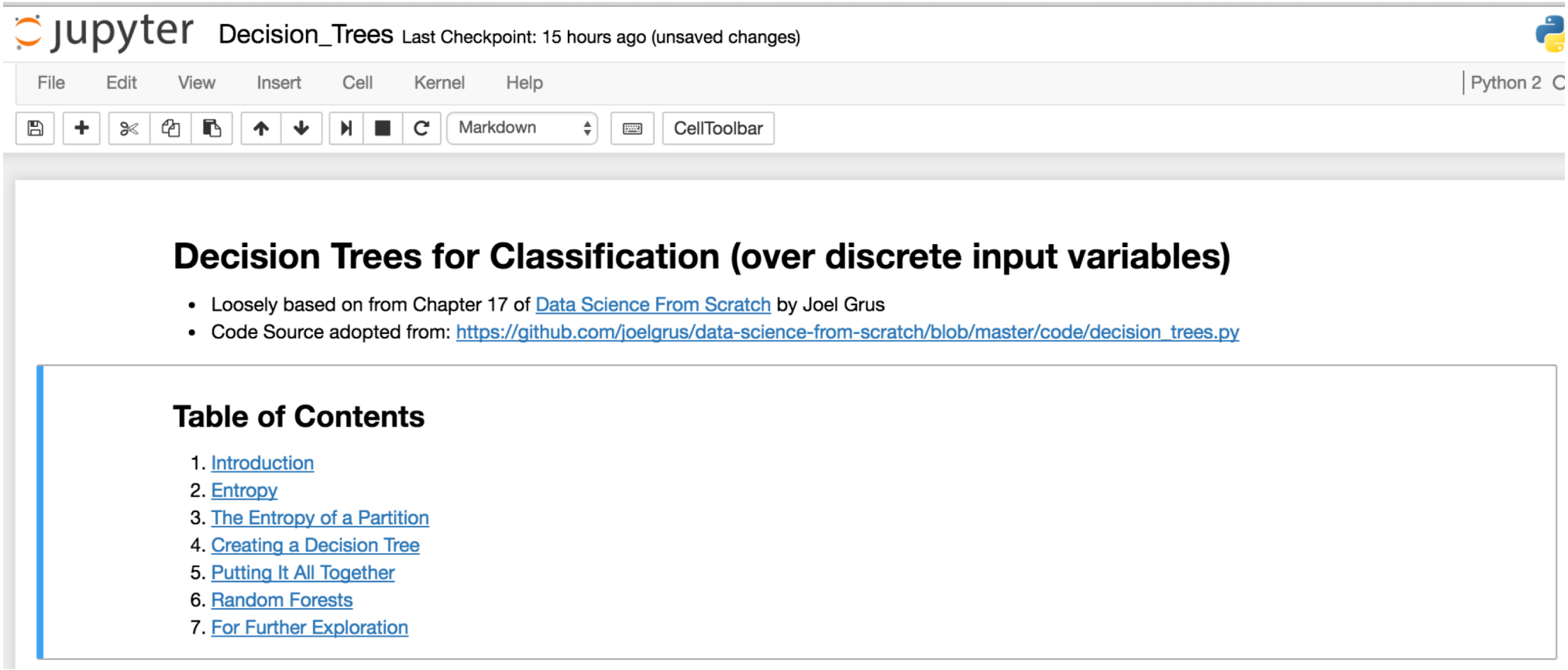
https://www.dropbox.com/s/xftbiscmkmulhhu/ISLR-BOOK-Tibshirani-Sixth%20Printing.pdf?dl=0

Slides for Chapter 8 and corresponding Video

- **Video**
  - https://www.youtube.com/playlist?list=PL5-da3qGB5IB23TLuA8ZgVGC8hV8ZAdGh
- **Slides**
  - https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/trees.pdf

Companion Notebook

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



Live Session Outline

- **Introduction**
- **Gradient descent versus non-gradient descent approaches**
- **Decision trees (DT)**
- **Learning decision trees**
  - Classification DTs on discrete variables (X, Y)
  - Regression DTs over continuous variables (X,y)
- **Ensembles of decision Trees**
- **Practical decision trees**
- **Summary**

Live Session Outline

- **Introduction**
- **Gradient descent versus non-gradient descent approaches**
- **Decision trees (DT)**
- **Learning decision trees**
  - Classification DTs on discrete variables (X, Y)
  - Regression DTs over continuous variables (X,y)
- **Ensembles of decision Trees**
- **Practical decision trees**
- **Summary**

Gradient Descent versus NonGradient descent

- **Gradient descent**
  - FOC, SOC, find global optimal in a principled way
  - Convexity, Constrained optimization → unconstrained opt.
  - Linear regression, logistic regression, Perceptrons, SVMs etc.
- **Non-gradient descent**
  - more local in nature;
  - Different types of search algorithms. E.g., hill climbing, heuristic in nature
  - Decision trees [Focus of this lecture]
    - Classification
    - Regression
  - Ensembles
  - Coordinate descent

Coordinate descent: both differentiable and derivative-free contexts

- Coordinate descent is an optimization algorithm that successively minimizes along coordinate directions to find the minimum of a function.
- At each iteration, the algorithm determines a coordinate or coordinate block via a coordinate selection rule, then exactly or inexactly minimizes over the corresponding coordinate hyperplane while fixing all other coordinates or coordinate blocks.
- A line search along the coordinate direction can be performed at the current iterate to determine the appropriate step size.
- Coordinate descent is applicable in both differentiable and derivative-free contexts.

Coordinate descent

- Coordinate descent can be 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.

Coordinate descent: differentiable case

Differentiable case [edit]

In the case of a continuously differentiable function F, a coordinate descent algorithm can be sketched as:[1]

- Choose an initial parameter vector $\mathbf{x}$.
- Until convergence is reached, or for some fixed number of iterations:
  - Choose an index i from 1 to n.
  - Choose a step size α.
  - Update x_i to $x_i - \alpha \frac{\partial F}{\partial x_i}(\mathbf{x})$.

The step size can be chosen in various ways, e.g., by solving for the exact minimizer of $f(x_i) = F(\mathbf{x})$ (i.e., F with all variables but x_i fixed), or by traditional line search criteria.[1]

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

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

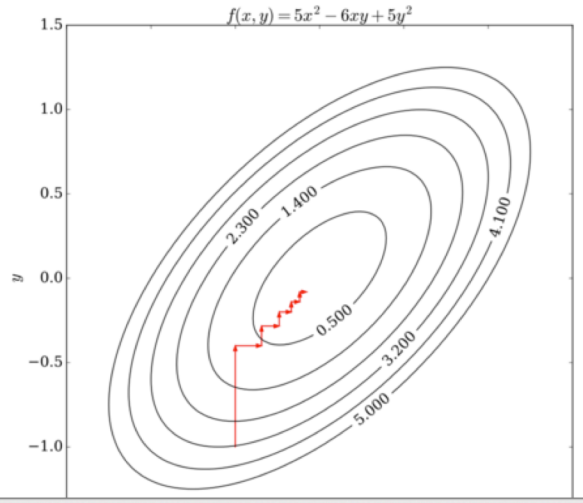
Thus, one begins with an initial guess $\mathbf{x}^0$ for a local minimum of F, and get a sequence $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots$ iteratively.

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.



Application of CD

- Moreover, there has been increased interest in the use of coordinate descent with the advent of large-scale problems in machine learning, where coordinate descent has been shown competitive to other methods when applied to such problems as training linear support vector machines[9] (see LIBLINEAR) and non-negative matrix factorization.[10]
- They are attractive for problems where computing gradients is infeasible, perhaps because the data required to do so are distributed across computer networks

Live Session Outline

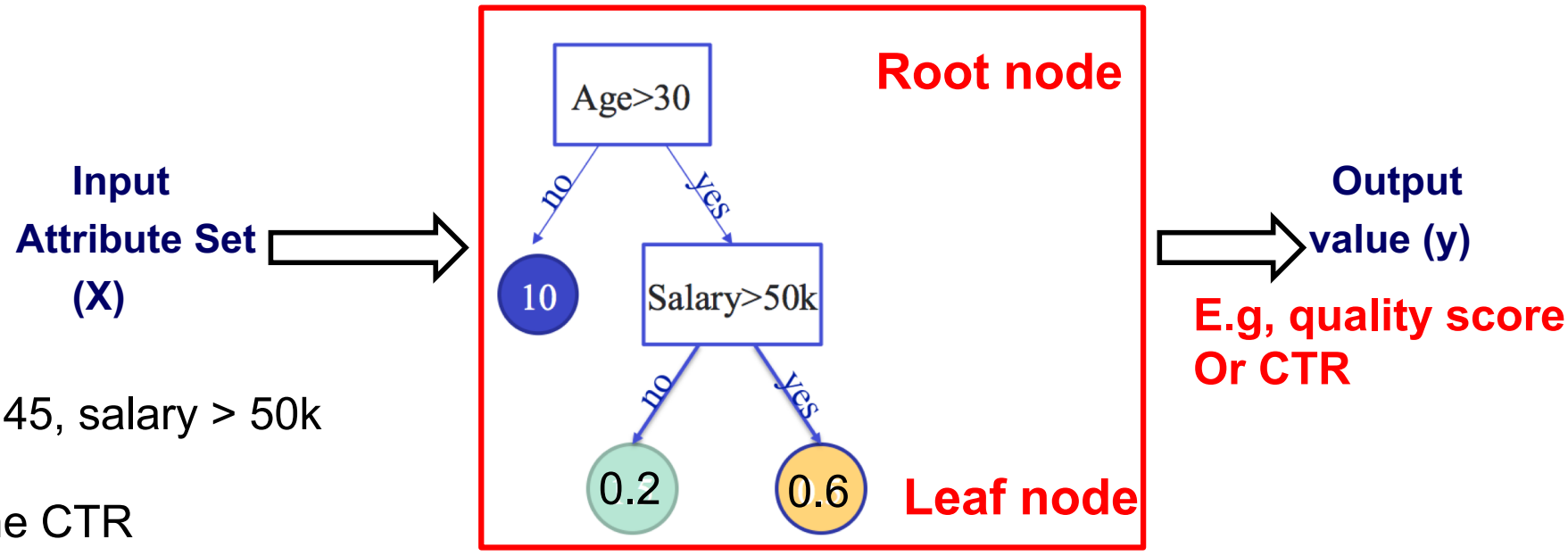
- **Introduction**
- **Gradient descent versus non-gradient descent approaches**
- **Decision trees (DT)**
- **Learning decision trees**
  - Classification DTs on discrete variables (X, Y)
  - Regression DTs over continuous variables (X,y)
- **Ensembles of decision Trees**
- **Practical decision trees**
- **Summary**

Decision Tree Approach

- **Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression.**
- **The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.**
- **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 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.**



Person is 45, salary > 50k

What is the CTR

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

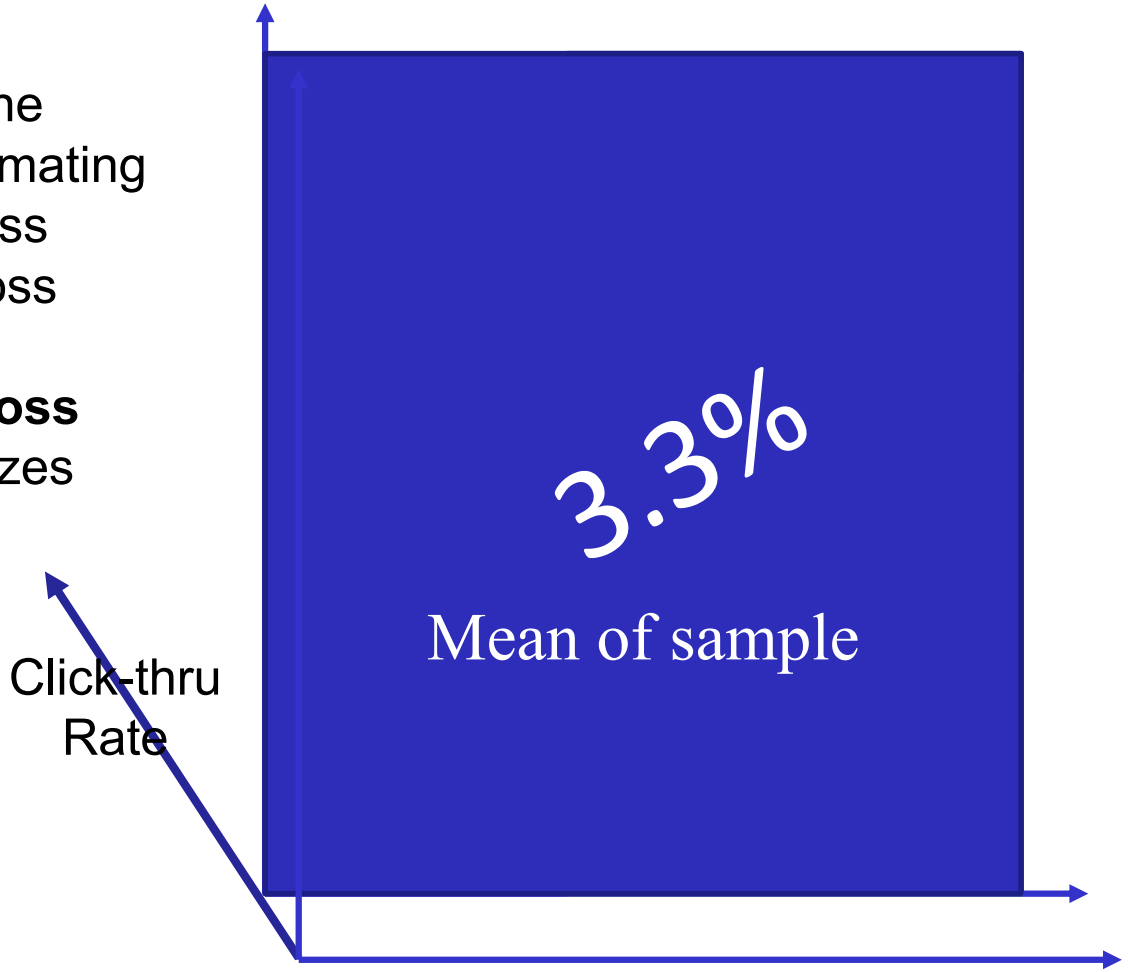
Decision Tree-based Segmentation

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

HeatMap

-

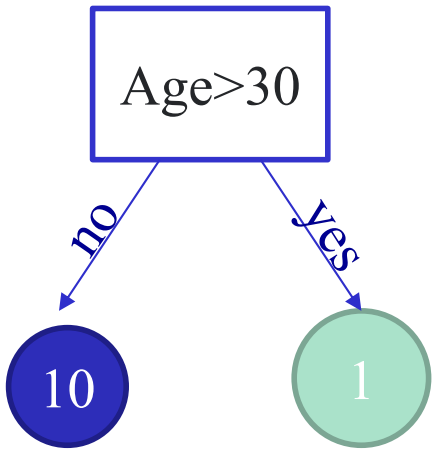
MEAN under SSE Loss Function
Under typical statistical assumptions, the mean or average is the statistic for estimating location that minimizes the expected loss experienced under the squared-error loss function,
MEDIAN under absolute-difference loss
the median is the estimator that minimizes expected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

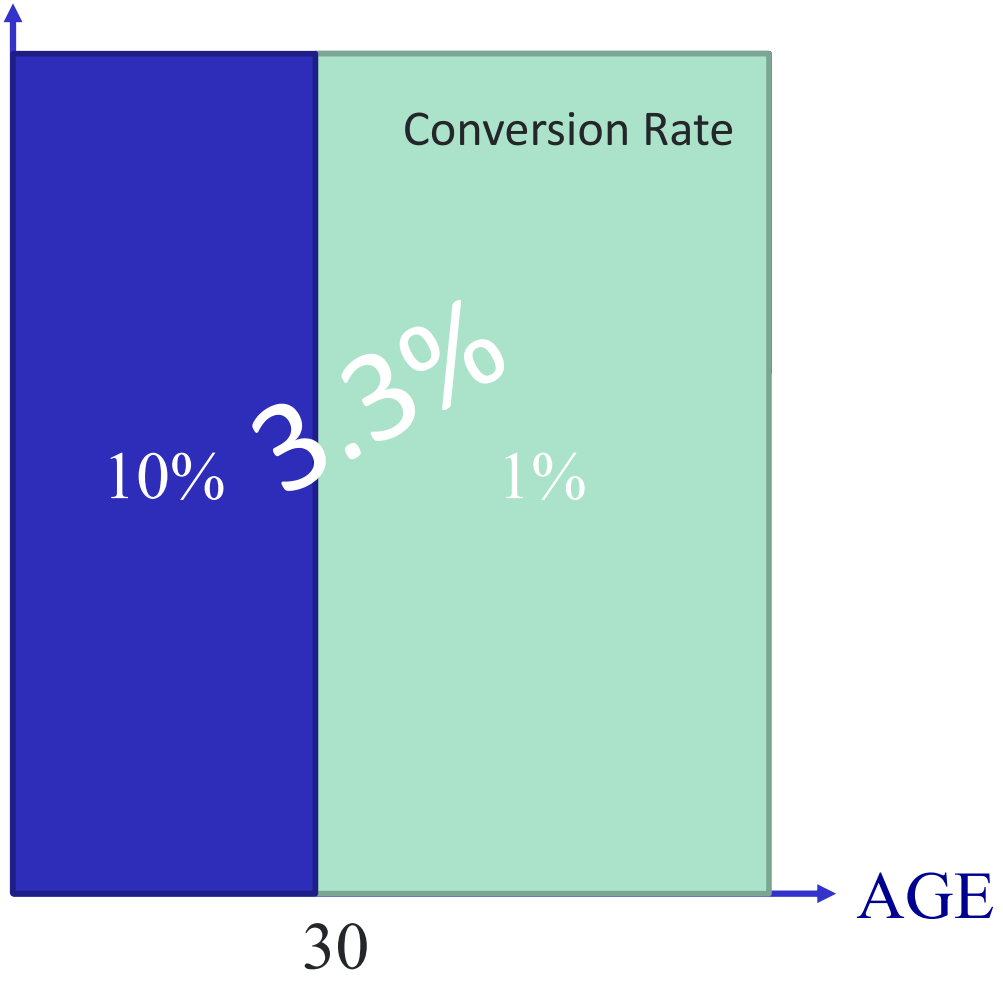


Decision Tree-based Segmentation

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

HeatMap



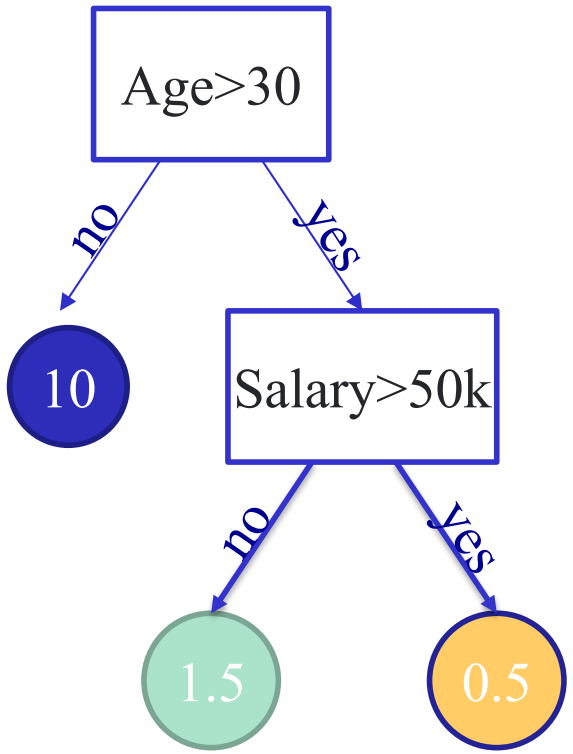


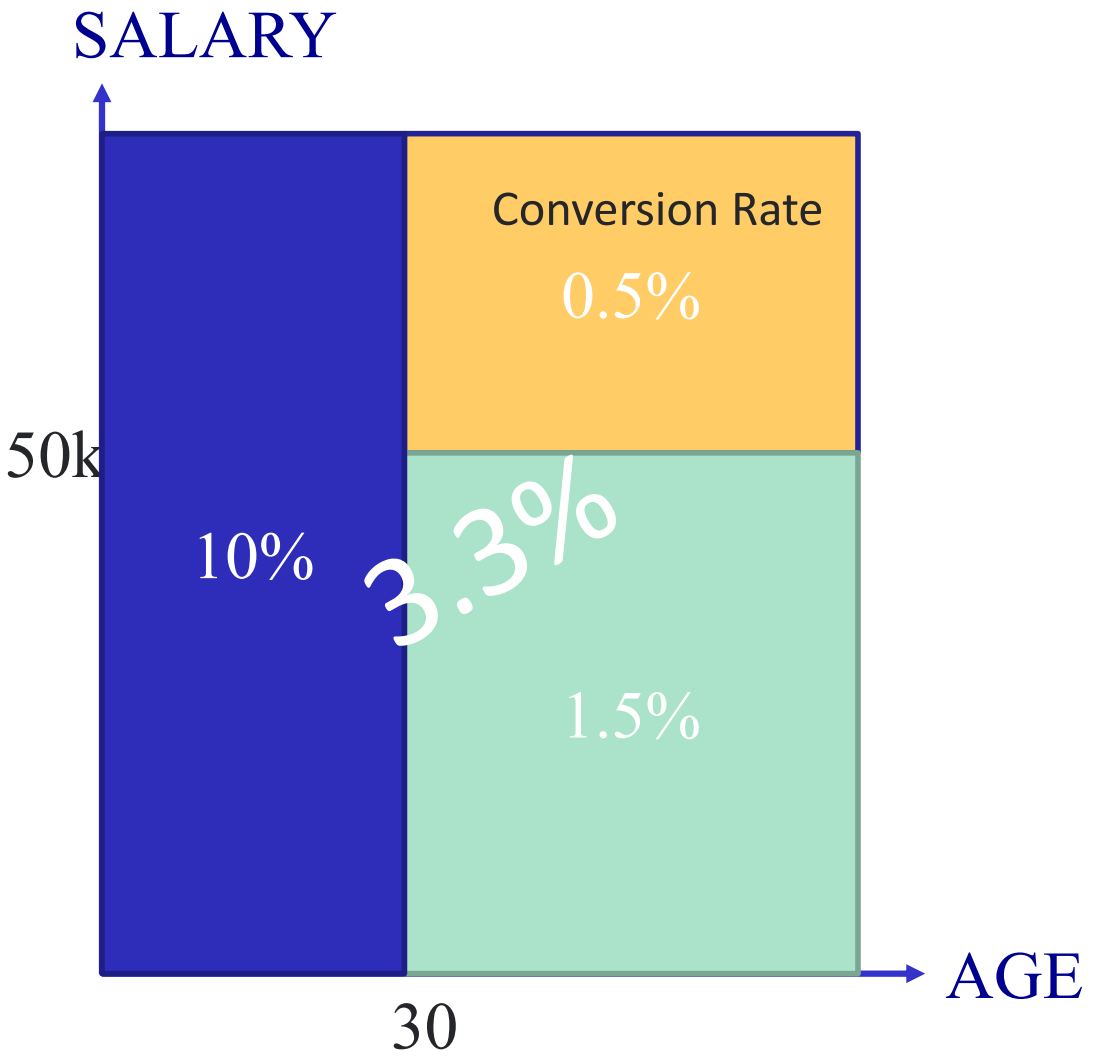
If Age > 30
Then predict CR = 1
Else predict CR =10

Decision Tree-based Segmentation (3D)

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

HeatMap



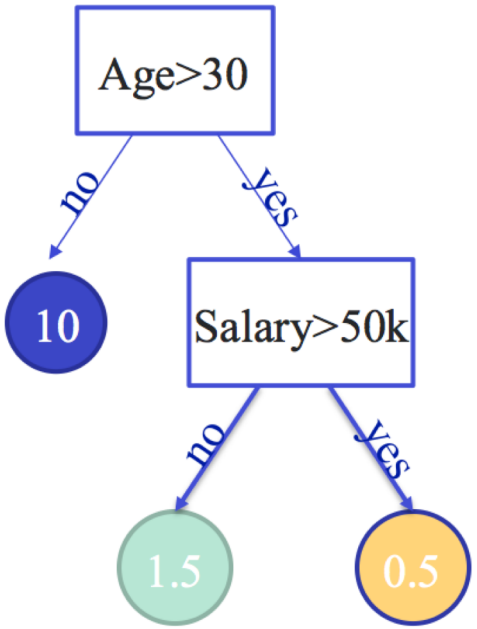


Decision tree learning

- **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)**

Person is 45, salary > 50k

What is the CTR



Decision Trees are very versatile

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

View Decision tree as a fancy way to do GROUP BYs in SQL

- **View Decision tree as a fancy way to do GROUP BYs in SQL or generating a Pivot table**
- **For categorical input variable and output variable the mapping is quite natural.**
- **For real-valued variables a little more explanation is required.**
- **Challenges in prediction: Data sparsity**
  - 1/1, 2/1,000

Tables in SQL

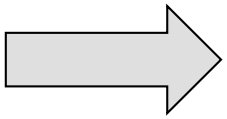
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Simple Aggregations

Purchase

Product	Date	Price	Quantity
Bagel	10/21	1	20
Banana	10/3	0.5	10
Banana	10/10	1	10
Bagel	10/25	1.50	20

```
SELECT  Sum(price * quantity)
FROM    Purchase
WHERE   product = 'bagel'
```



50 (= 20+30)

Grouping and Aggregation

Purchase(product, date, price, quantity)

Find total sales after 10/1/2005 per product.

```
SELECT      product, Sum(price*quantity) AS TotalSales
FROM        Purchase
WHERE       date > '10/1/2005'
GROUP BY    product
```

Let's see what this means…

Grouping and Aggregation

1. Compute the FROM and WHERE clauses.

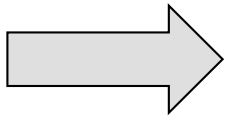
2. Group by the attributes in the GROUPBY

3. Compute the SELECT clause: grouped attributes and aggregates.

Group By

Purchase Table

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10



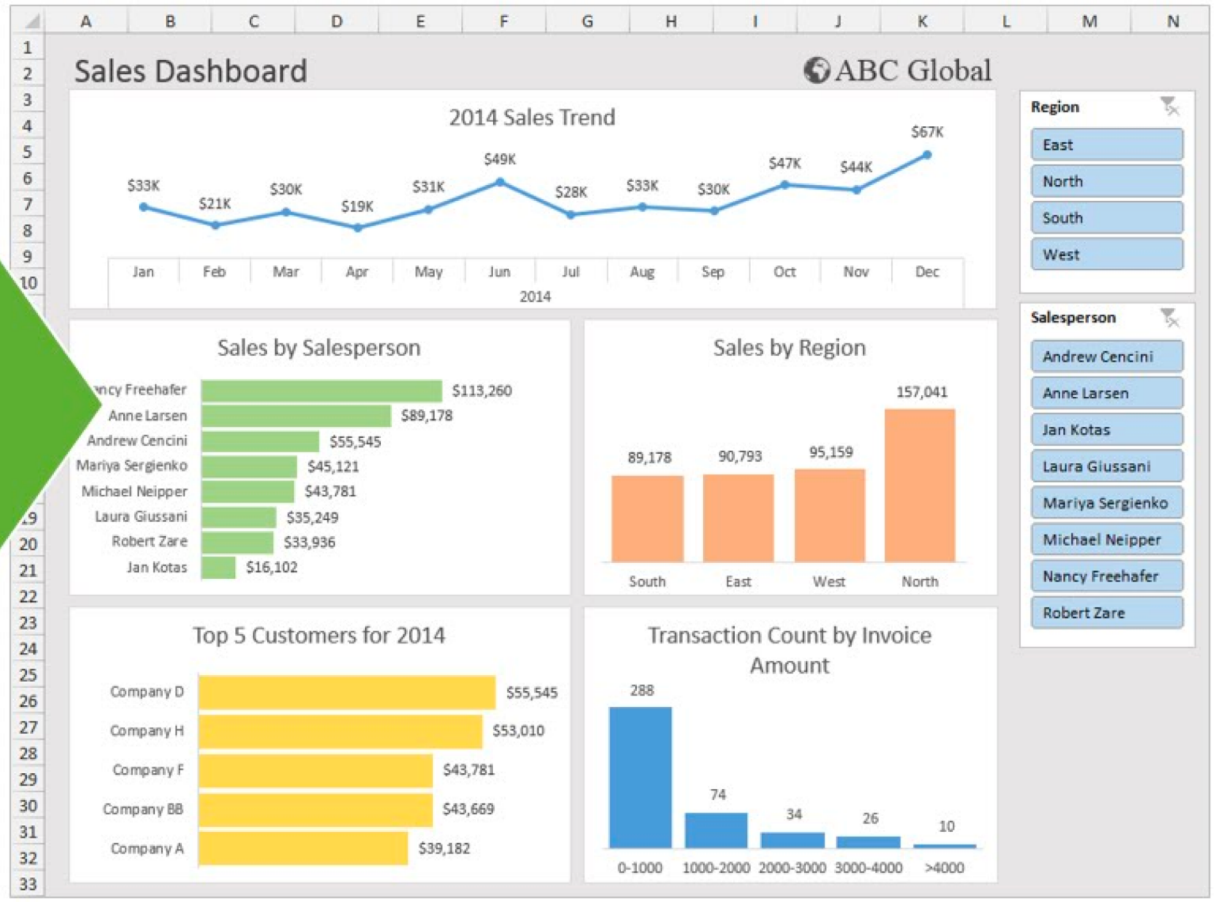
Product	TotalSales
Bagel	50
Banana	15

```
SELECT      product, Sum(price*quantity) AS TotalSales
FROM        Purchase
WHERE       date > '10/1/2005'
GROUP BY    product
```

Pivot Table

Turn your data into a beautiful report or dashboard with **Pivot Tables**!

Order	Order Da	Customer	Customer Nan	Address	City
1015	01/27/14	27	Company AA	789 27th Street	Las V
1015	01/27/14	27	Company AA	789 27th Street	Las V
1016	01/04/14	4	Company D	123 4th Street	New
1016	01/04/14	4	Company D	123 4th Street	New
1016	01/04/14	4	Company D	123 4th Street	New
1017	01/12/14	12	Company L	123 12th Street	Las V
1017	01/12/14	12	Company L	123 12th Street	Las V
1018	01/08/14	8	Company H	123 8th Street	Portl
1019	01/04/14	4	Company D	123 4th Street	
1020	01/29/14	29	Company CC	789 29th Stre	
1021	01/03/14	3	Company C	123 3rd Street	
1022	01/06/14	6	Company F	123 6th Street	
1023	01/28/14	28	Company BB	789 28th Stre	
1024	01/08/14	8	Company H	123 8th Street	
1025	01/10/14	10	Company J	123 10th Stre	
1026	01/07/14	7	Company G	123 7th Street	
1027	01/10/14	10	Company J	123 10th Stre	
1027	01/10/14	10	Company J	123 10th Stre	
1027	01/10/14	10	Company J	123 10th Street	Chica
1028	01/11/14	11	Company K	123 11th Street	Mian
1028	01/11/14	11	Company K	123 11th Street	Mian
1029	01/01/14	1	Company A	123 1st Street	Seatt
1029	01/01/14	1	Company A	123 1st Street	Seatt
1029	01/01/14	1	Company A	123 1st Street	Seatt
1030	01/28/14	28	Company BB	789 28th Street	Mem
1030	01/28/14	28	Company BB	789 28th Street	Mem
1031	01/09/14	9	Company I	123 9th Street	Salt L
1031	01/09/14	9	Company I	123 9th Street	Salt L
1032	01/06/14	6	Company F	123 6th Street	Milw
1033	02/08/14	8	Company H	123 8th Street	Portl
1034	02/03/14	3	Company C	123 3rd Street	Los A
1034	02/03/14	3	Company C	123 3rd Street	Los A



Facebook interview question on SQL

Interview Question

Data Scientist Interview Menlo Park, CA

"You have a table with appID, eventID, and timestamp . eventID is either 'click' or 'impression'. Calculate the click through rate. Now do it in for each app."

```
select
  clicks/impressions CTR,
  sum(case when eventID = 'click' then 1 else 0 end) clicks,
  sum(case when eventID = 'click' then 1 else 0 end) impressions,
from log_table
```

Facebook interview question on SQL

Interview Question

Data Scientist Interview Menlo Park, CA

"You have a table with appID, eventID, and timestamp . eventID is either 'click' or 'impression'. Calculate the click through rate. Now do it in for each app."

```
select
  appID,
  clicks/impressions CTR,
  sum(case when eventID = 'click' then 1 else 0 end) clicks,
  sum(case when eventID = 'click' then 1 else 0 end) impressions,
from log_table
group by appID
```

CTR Table output

AppID	Sum Impressions	Sum of Clicks	Sum of calculated CT
App1	60988	29732	48.75%
App2	34294	415	1.21%
App3	19644	164	0.83%
	16502	4	0.02%
….	15669	10185	65.00%
	13941	26	0.19%
	11698	124	1.06%
….	….	….	
App10	1	1	100%
App11	1	0	0%
….			

Trust a click thru rate of 100% with one impression
VERSUS
A click thru rate of 1% with 1,000 impressions

Expand GroupBy → Sparsity; coalesce

category,	Sum Impressions	Sum of Clicks	Sum of calculated CT
sports	60988	29732	48.75%
action	34294	415	1.21%
kids	19644	164	0.83%
	16502	4	0.02%
….	15669	10185	65.00%
	13941	26	0.19%
	11698	124	1.06%
….	….	….	
Cat10	1	1	100%
Cat50	1	0	0%
….			

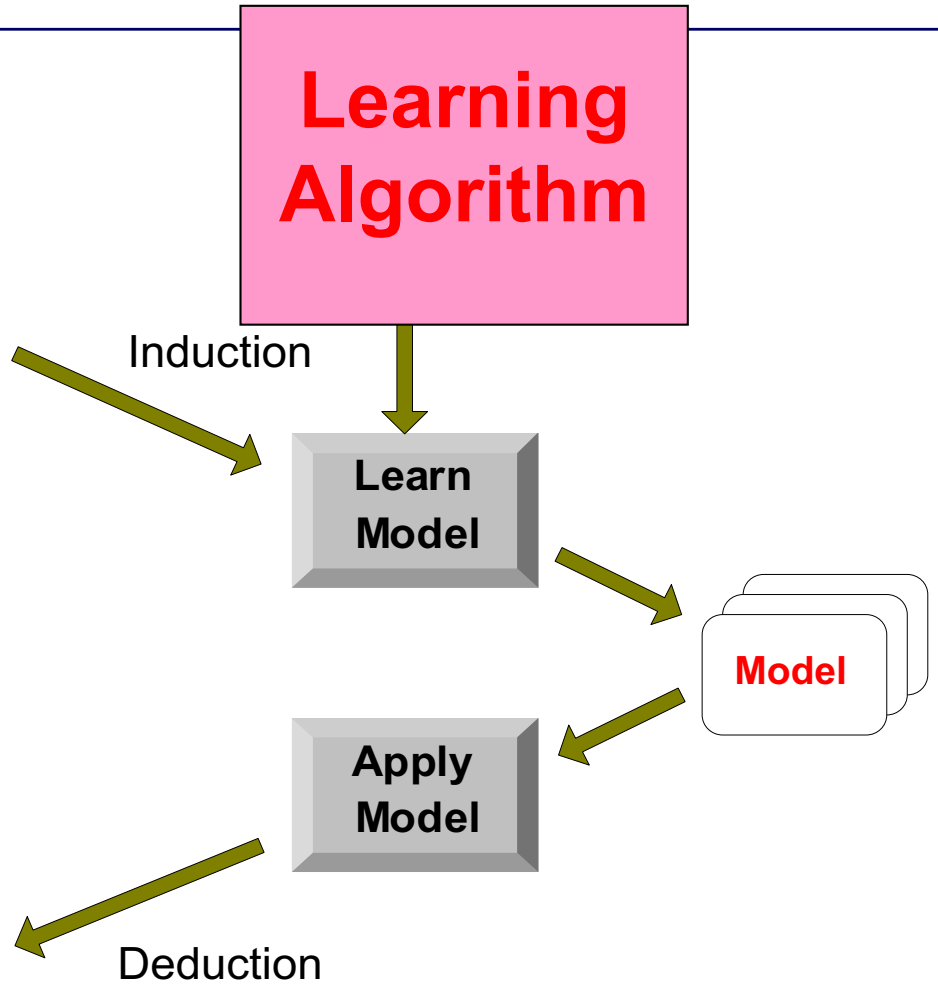
Illustration of the Classification Task:

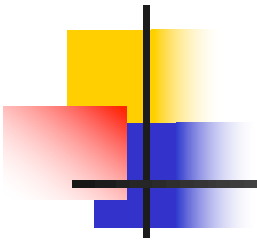
Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set





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*

Training Examples

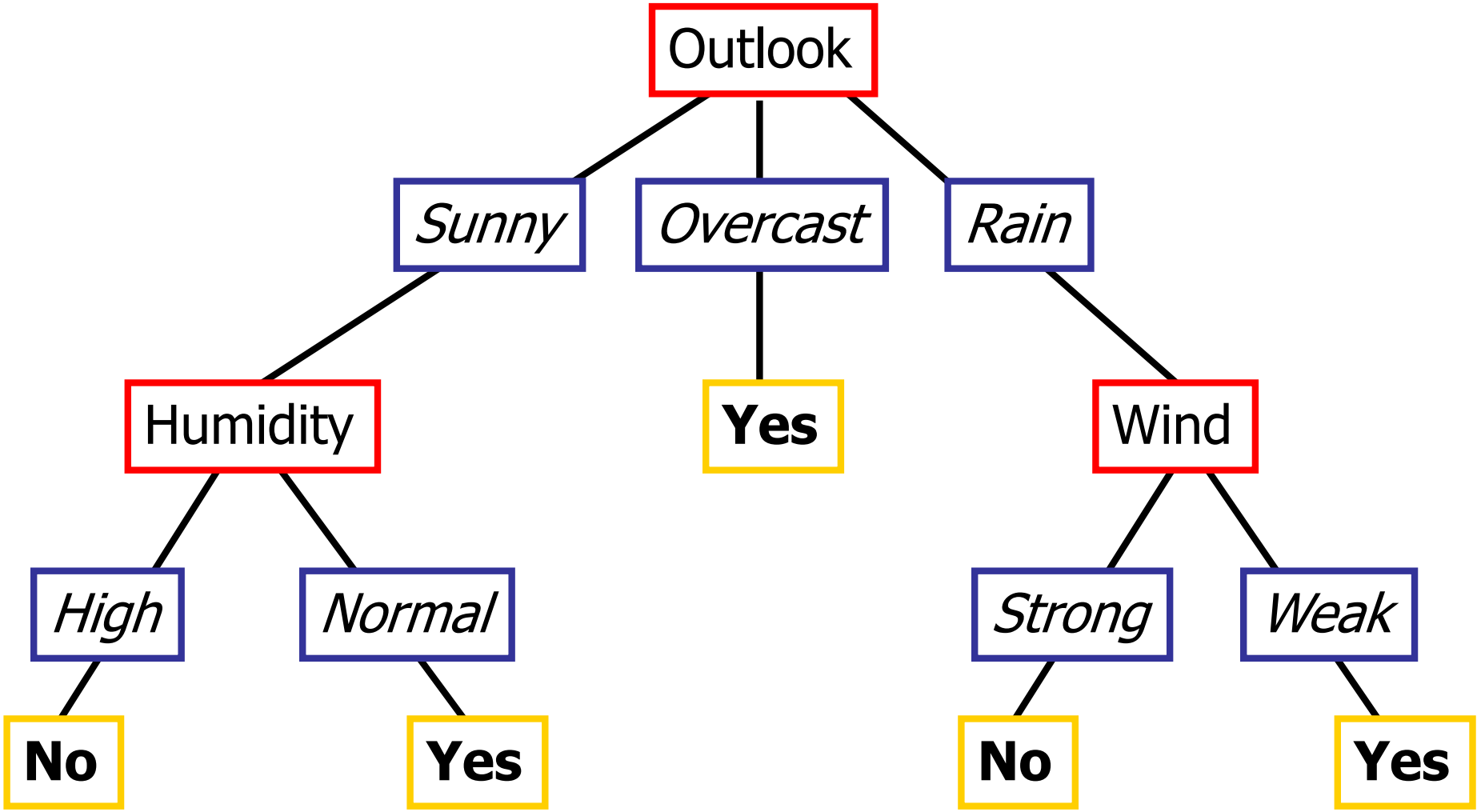
Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes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

Decision Trees via SQL Group By

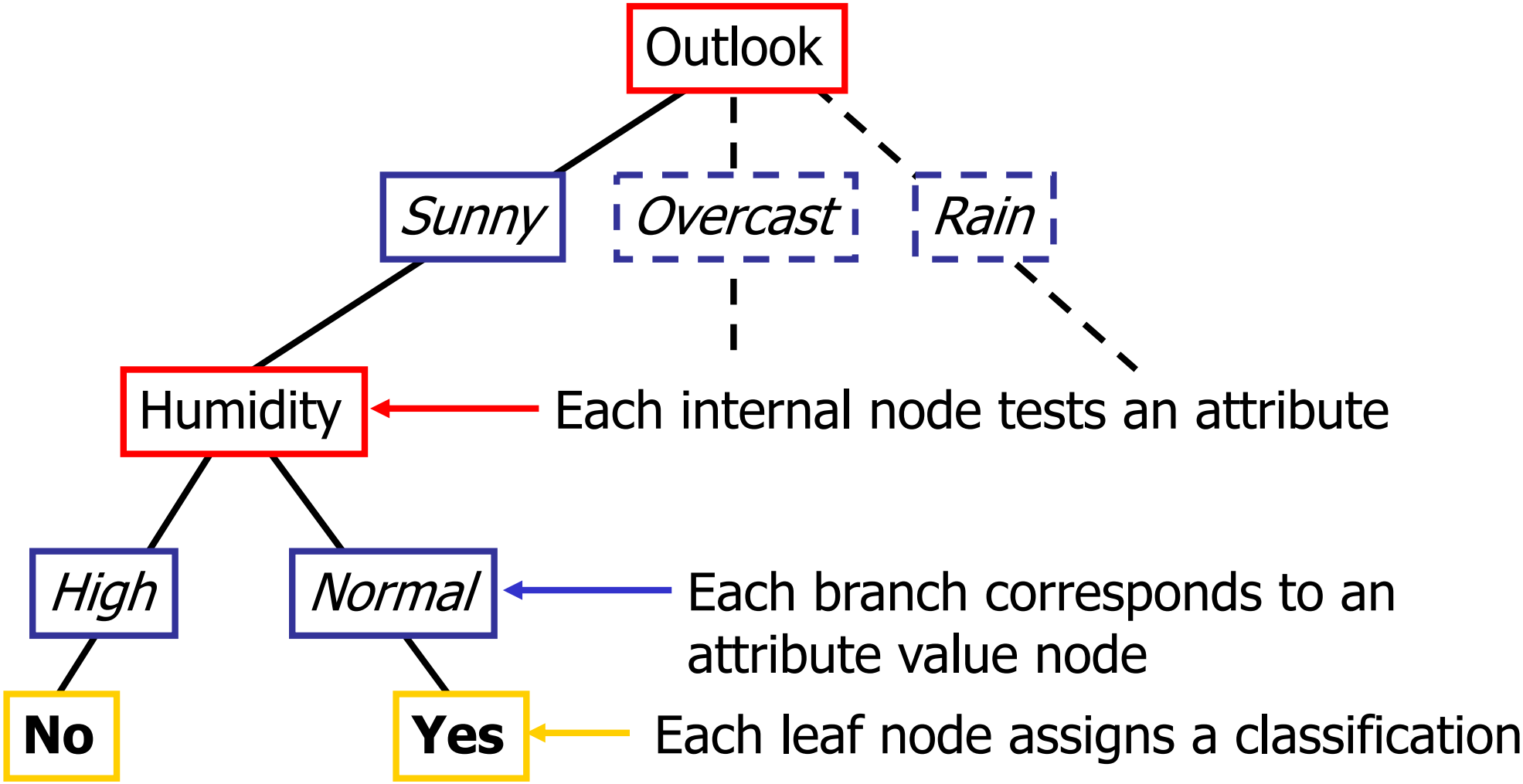
- **Attributes and their values:**
  - Outlook: *Sunny, Overcast, Rain*
  - Humidity: *High, Normal*
  - Wind: *Strong, Weak*
  - Temperature: *Hot, Mild, Cool*

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

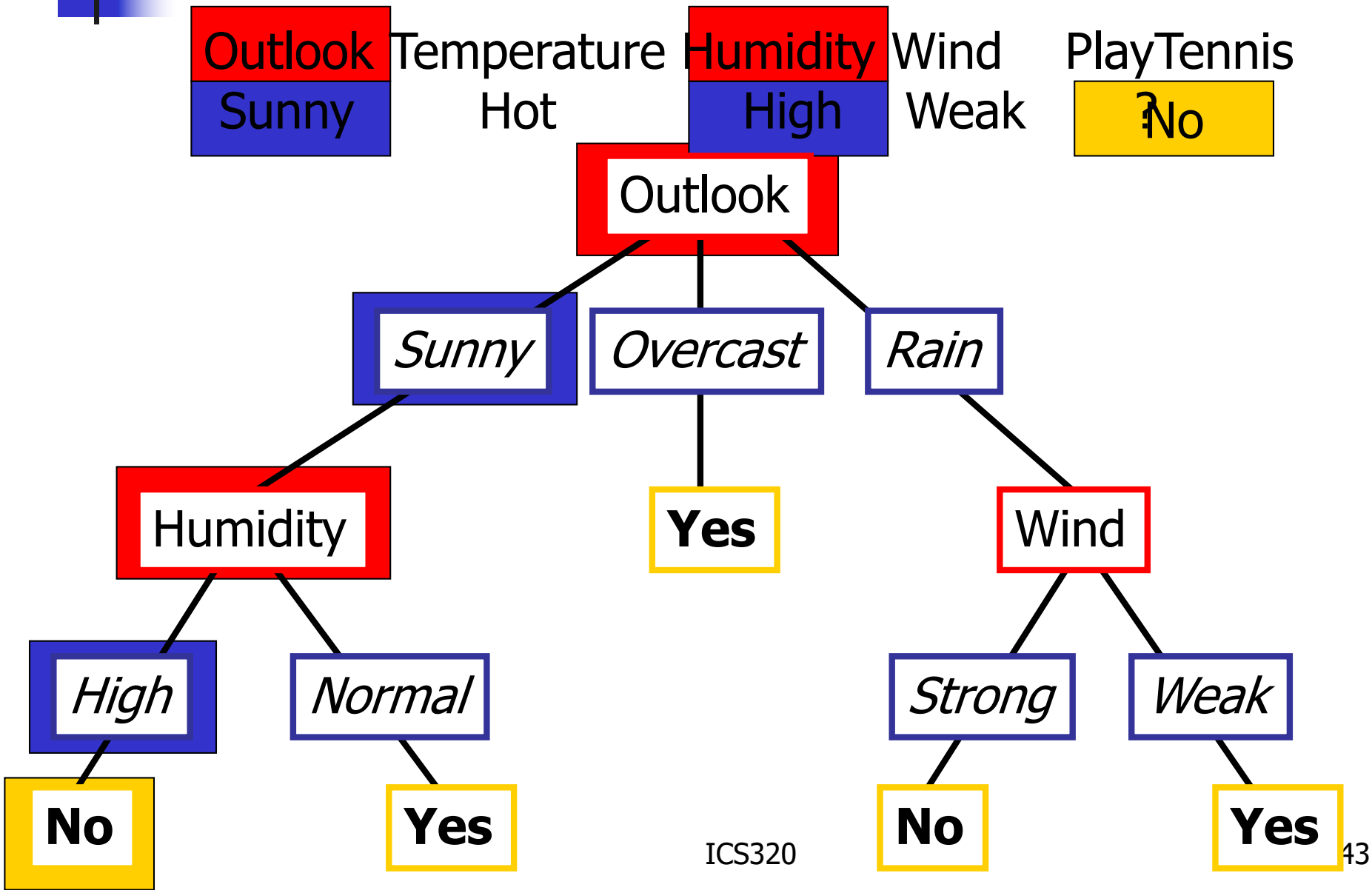
Decision Tree for PlayTennis



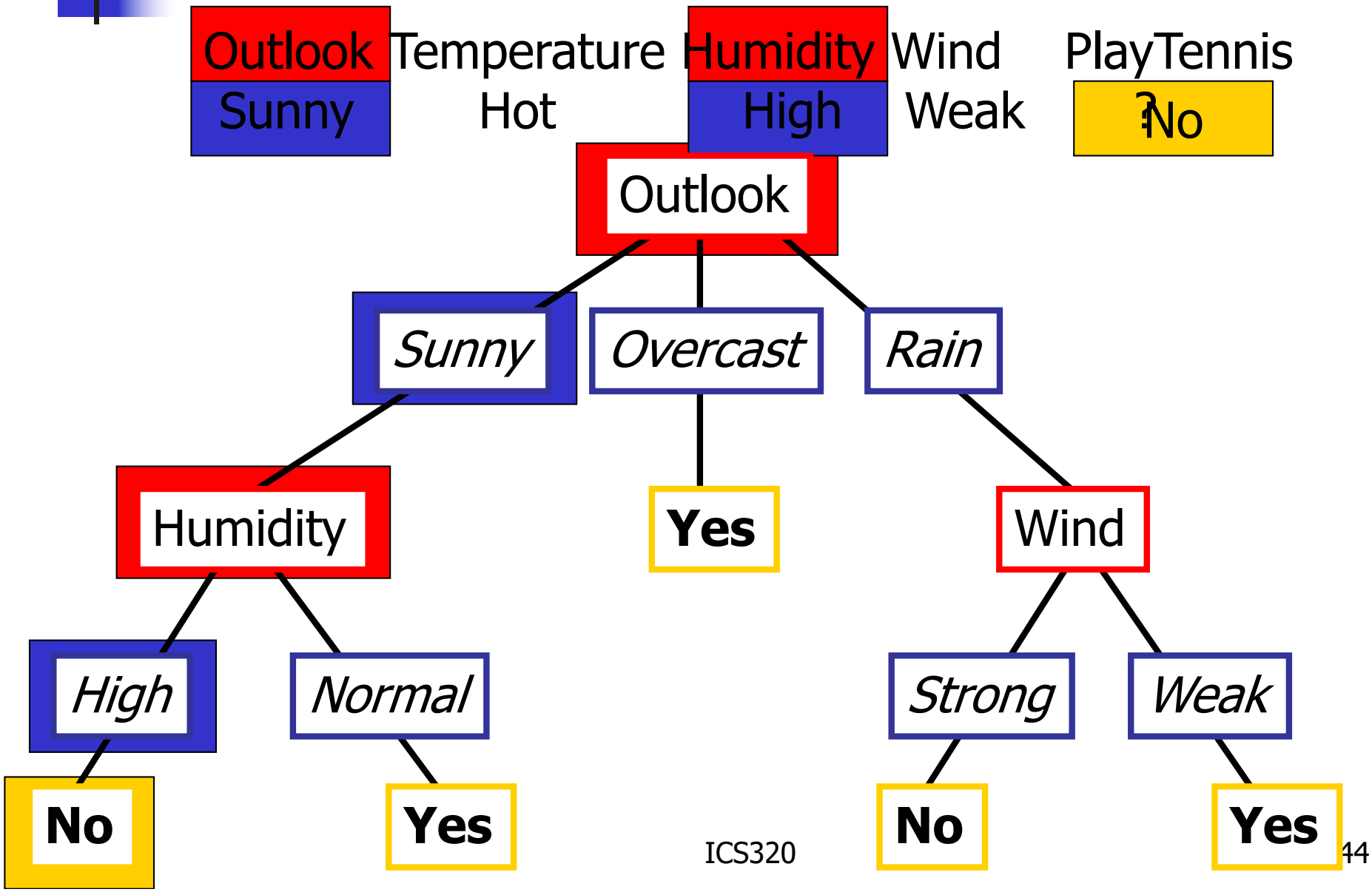
Decision Tree for PlayTennis



Decision Tree for PlayTennis

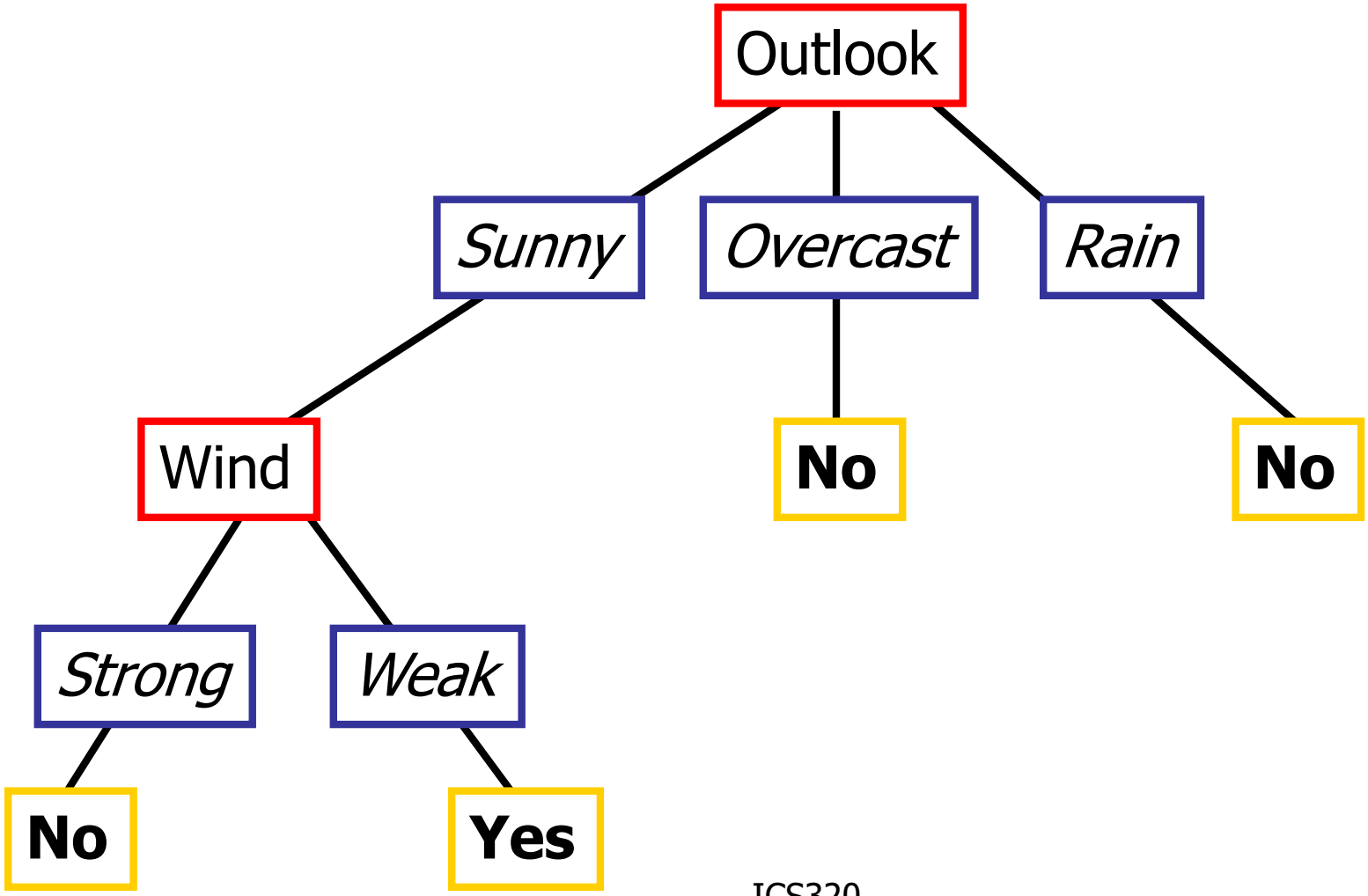


Decision Tree for PlayTennis



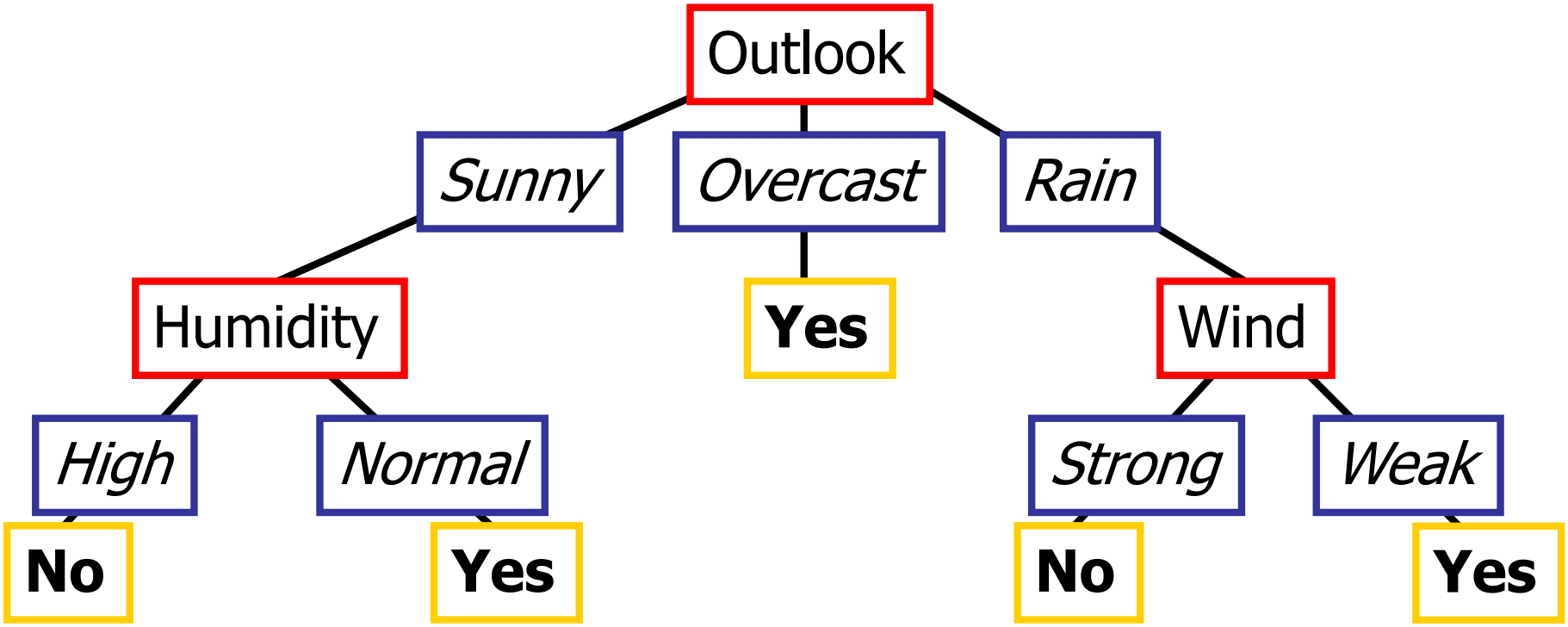
Decision Tree for Conjunction

Outlook=Sunny $\wedge$ Wind=Weak



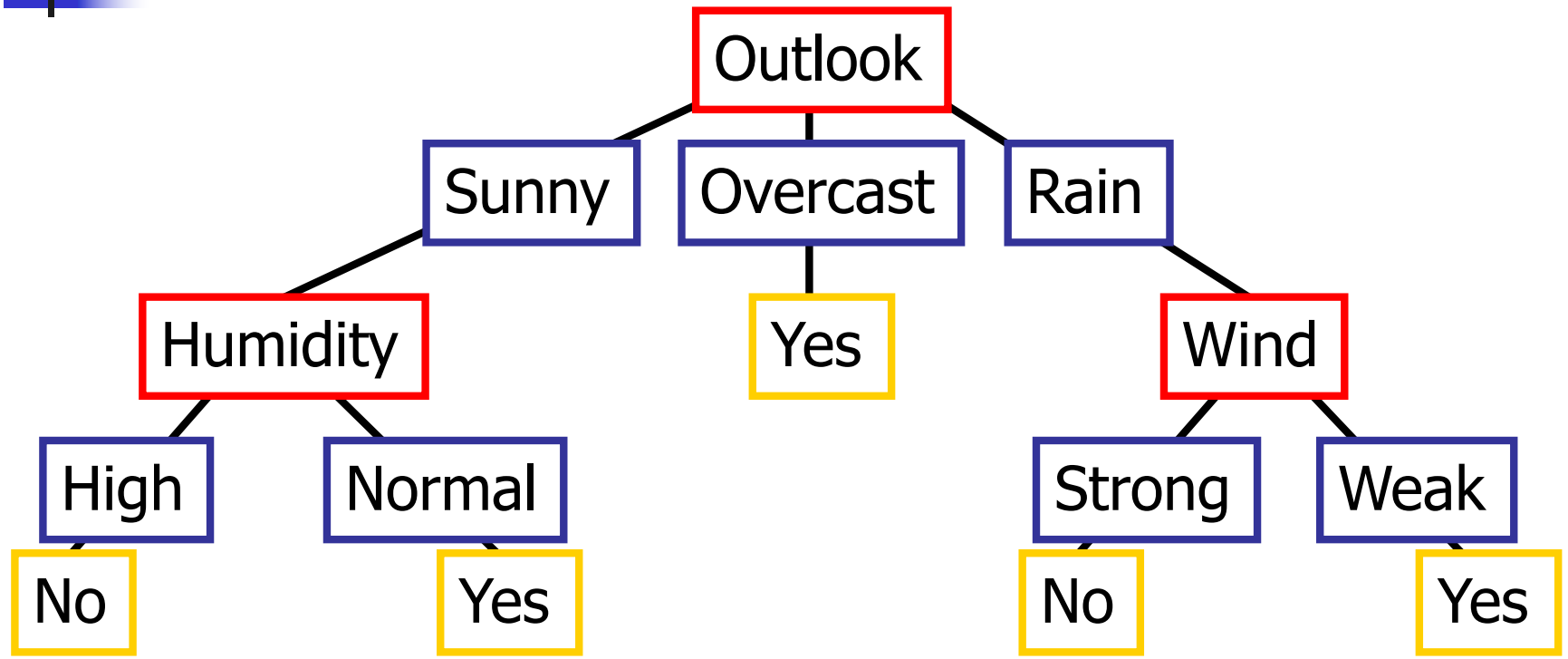
Decision Tree

- decision trees represent disjunctions of conjunctions

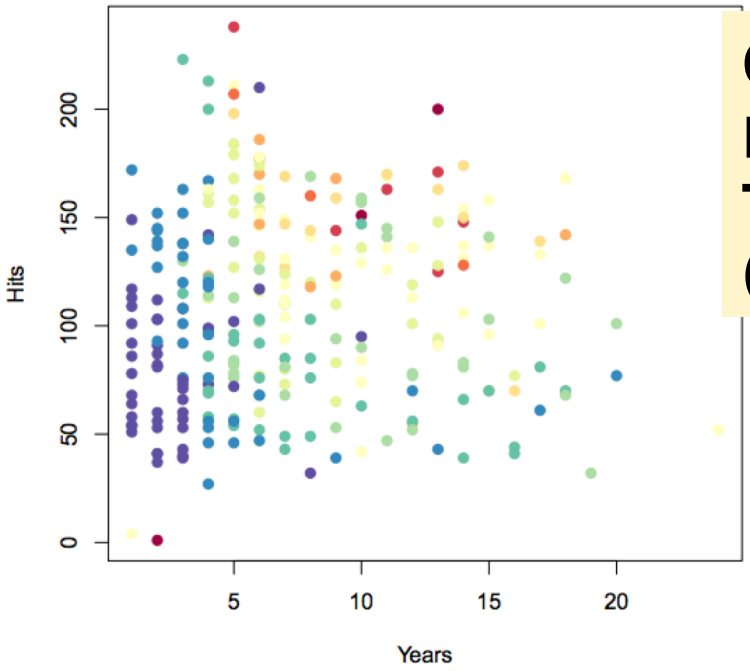


(Outlook=Sunny ∧ Humidity=Normal)
∨ (Outlook=Overcast)
∨ (Outlook=Rain ∧ Wind=Weak)

Converting a Tree to Rules



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

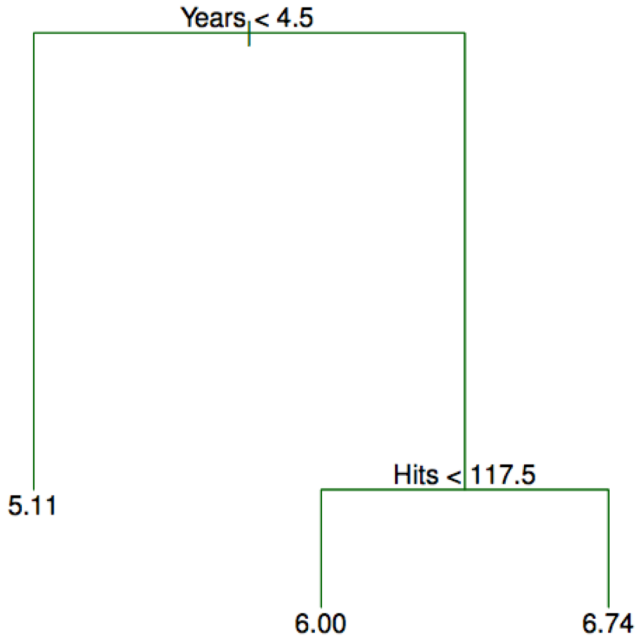


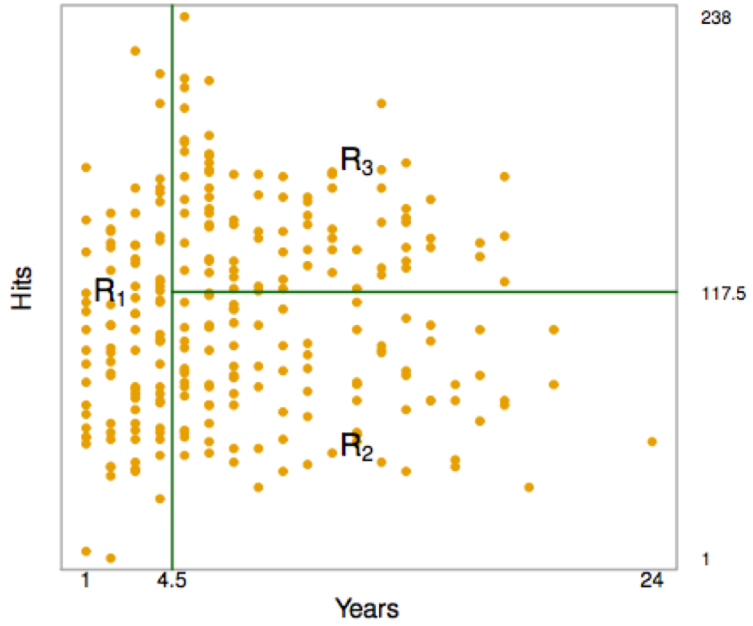
GOAL: Predict salary from years, hits

Baseball salary data:

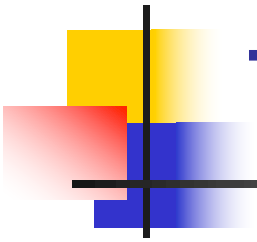
To visualize, salary is color-coded from low (blue, green) to high (yellow,red)

Overall, the tree stratifies or segments the players into three regions of predictor space: $R_1 = \{X \mid \texttt{Years} < 4.5\}$, $R_2 = \{X \mid \texttt{Years} >= 4.5, \texttt{Hits} < 117.5\}$, and $R_3 = \{X \mid \texttt{Years} >= 4.5, \texttt{Hits} >= 117.5\}$.





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)

Live Session Outline

- Introduction
- Gradient descent versus non-gradient descent approaches
- Decision trees (DT)
- Learning decision trees
  - Classification DTs on discrete variables (X, Y)
  - Regression DTs over continuous variables (X,y)
- Ensembles of decision Trees
- Practical decision trees
- Summary

Algorithms: ID3, CART, c4.5, c5

Grow tree by determining best (variable, split point) combination using InfoGain/Gini

	Real-Value Input versus Discete	Real-valued Output	Variable spliting	DT Learning Algorithm
	Real	Real	MSE (aka variance)	ID3, CART, c4.5, c5
	Real	Categorical	InfoGain	ID3, CART, c4.5, c5
	Categorical	Real	MSE/Variance	ID3, CART, c4.5, c5
Easiest	Categorical	Categorical	InfoGain	ID3, CART, c4.5, c5

Live Session Outline

- Introduction
- Gradient descent versus non-gradient descent approaches
- Decision trees (DT)
- Learning decision trees
  - Classification DTs on discrete variables (X, Y)
  - Regression DTs over continuous variables (X,y)
- Ensembles of decision Trees
- Practical decision trees
- Summary

Decision Trees for Classification

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

What is ID3?
Works for Discrete Inputs and outputs

- **ID3 (Iterative Dichotomiser 3) is one of the most celebrated algorithms in ML**
- **It learns decision trees from training data**
- **Works only with ordinal or nominal attributes!**
- **Each attribute occurs at most once**
- **Developed by J. Ross Quinlan in the late 1970's**
- **Commercial versions include**
  - Quinlan's own C5 (UNIX), See5 (Windows)
    - Also public-domain version, C4.5
- **CART (Breiman et al.) is a similar algorithm independently developed**
  - available in Clementine

ID3 as a search algorithm

ID3 searches a tree that fits the training examples

- ***top-down***
  - **from the root , which is the most general tree, towards more specific trees**
- ***hill-climbing (=greedy) search***
  - **its next tree is always an immediate successor to the last one**
- ***without backtracking***
  - **once an attribute is selected, ID3 never changes its mind**
- ***using information gain as a heuristic***
  - **to select the best expansion**
- ***A leaf is assigned the majority class amongst the training examples in it***

Variable Selection

- The central idea is that some attributes tell us more about class than others
- Also at any point in the tree there is an attribute which can best add to our information *locally* at that point
- Roughly speaking, an attribute is *informative* if each of its values tends to be associated with one class
- The *informativeness* of a node is measured using a heuristic evaluation function
  - This assesses the contribution made by each child node of the expansion

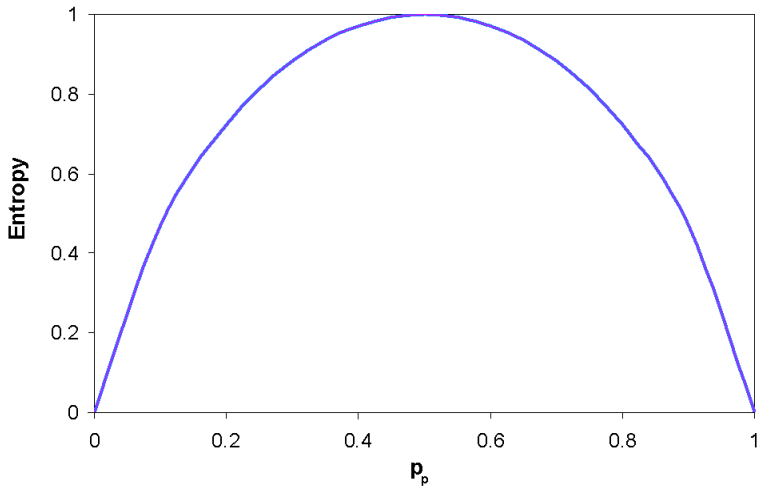
Entropy of a set

- **Entropy or information impurity in a node v:**

$$I = -\sum_{c} p(c) \log_2 p(c)$$

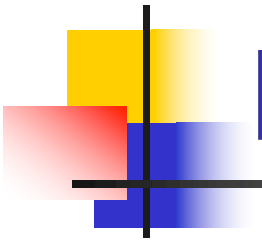
p(c) is the fraction of examples (in node v) that are in class c

- **For a training set containing *p* positive examples and *n* negative examples:**



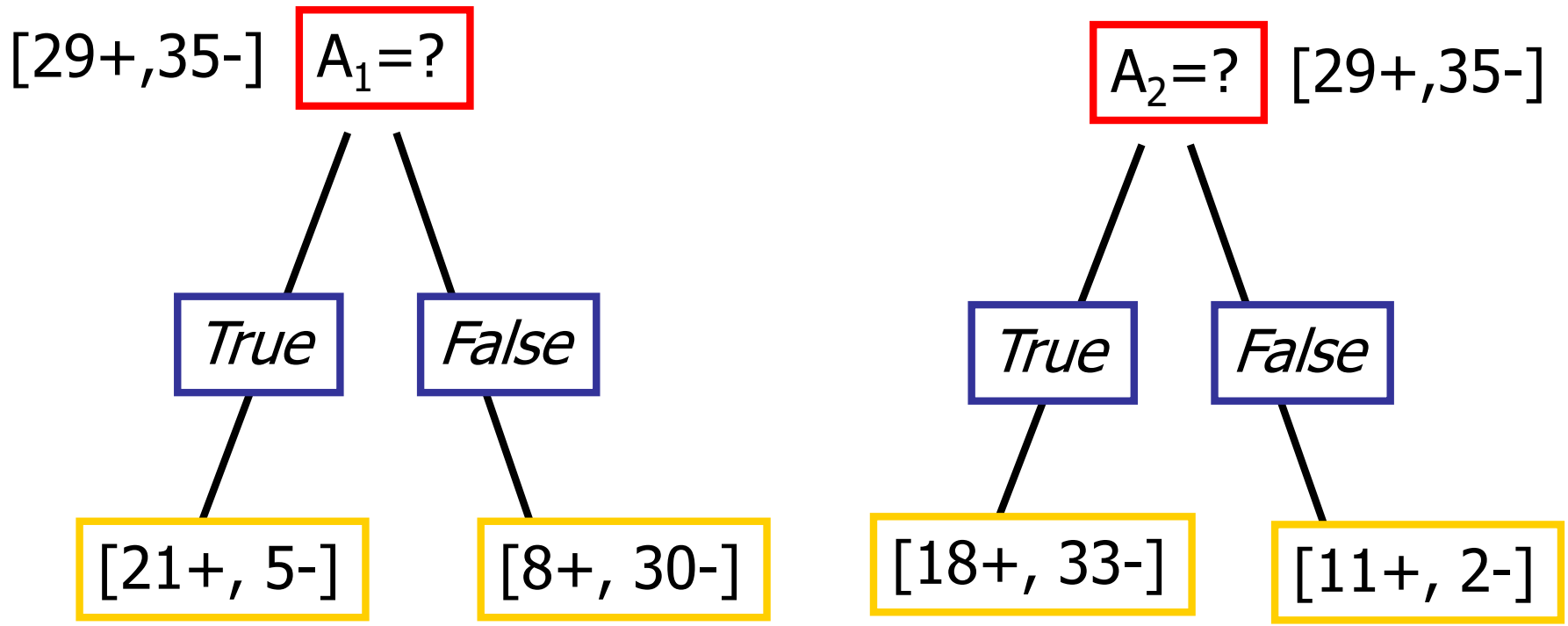
$$I(v) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Top-Down Induction of Decision Trees ID3

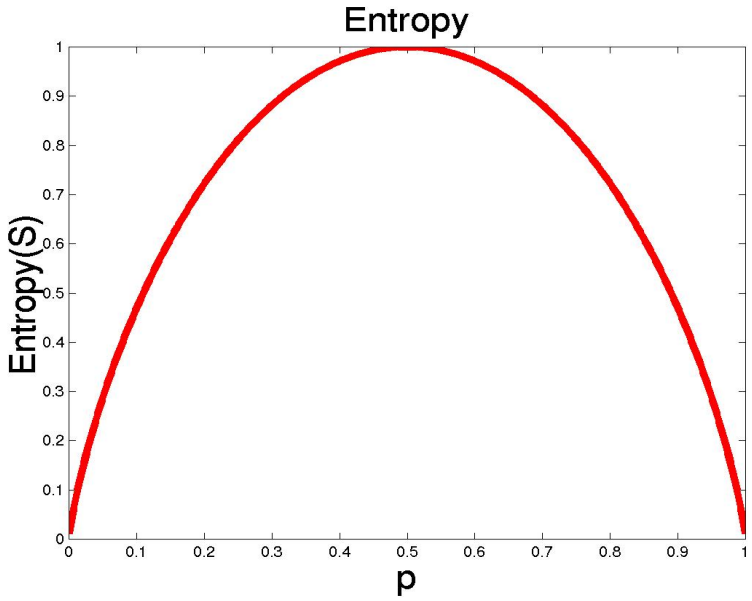


1. A ← the "best" decision attribute for next *node*
2. Assign A as decision attribute for *node*
3. For each value of A create new descendant
4. Sort training examples to leaf node according to the attribute value of the branch
5. If all training examples are perfectly classified (same value of target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

Which Attribute is "best"?

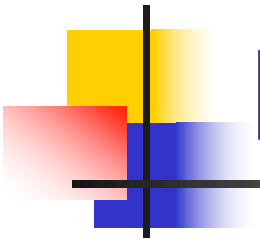


Entropy



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

Info Gain == KL Divergence (mutual information)

Information gain in decision trees

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(December 2009) (Learn how and when to remove this template message)*

In information theory and machine learning, **information gain** is a synonym for *Kullback–Leibler divergence*. However, in the context of decision trees, the term is sometimes used synonymously with mutual information, which is the expected value of the Kullback–Leibler divergence of the univariate probability distribution of one variable from the conditional distribution of this variable given the other one.

In particular, the information gain about a random variable X obtained from an observation that a random variable A takes the value $A=a$ is the Kullback-Leibler divergence $D_{KL}(p(x \mid a) \| p(x \mid I))$ of the prior distribution $p(x \mid I)$ for x from the posterior distribution $p(x \mid a)$ for x given a.

The expected value of the information gain is the mutual information $I(X; A)$ of X and A – i.e. the reduction in the entropy of X achieved by learning the state of the random variable A.

In machine learning, this concept can be used to define a preferred sequence of attributes to investigate to most rapidly narrow down the state of X. Such a sequence (which depends on the outcome of the investigation of previous attributes at each stage) is called a decision tree. Usually an attribute with high mutual information should be preferred to other attributes.

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

General definition [edit]

In general terms, the expected information gain is the change in information entropy H from a prior state to a state that takes some information as given:

$$IG(T,a) = H(T) - H(T|a)$$

Formal definition [edit]

Let T denote a set of training examples, each of the form $(\mathbf{x}, y) = (x_1, x_2, x_3, \ldots, x_k, y)$ where $x_a \in vals(a)$ is the value of the ath attribute of example $\mathbf{x}$ and y is the corresponding class label. The information gain for an attribute a is defined in terms of entropy $H()$ as follows:

$$IG(T,a) = H(T) - \sum_{v \in vals(a)} \frac{|\{\mathbf{x} \in T | x_a = v\}|}{|T|} \cdot H(\{\mathbf{x} \in T | x_a = v\})$$

The mutual information is equal to the total entropy for an attribute if for each of the attribute values a unique classification can be made for the result attribute. In this case, the relative entropies subtracted from the total entropy are 0.

Drawbacks [edit]

Although information gain is usually a good measure for deciding the relevance of an attribute, it is not perfect. A notable problem occurs when information gain is applied to attributes that can take on a large number of distinct values. For example, suppose that one is building a decision tree for some data describing the customers of a business. Information gain is often used to decide which of the attributes are the most relevant, so they can be tested near the root of the tree. One of the input attributes might be the customer's credit card number. This attribute has a high mutual information, because it uniquely identifies each customer, but we do *not* want to include it in the decision tree: deciding how to treat a customer based on their credit card number is unlikely to generalize to customers we haven't seen before (overfitting).

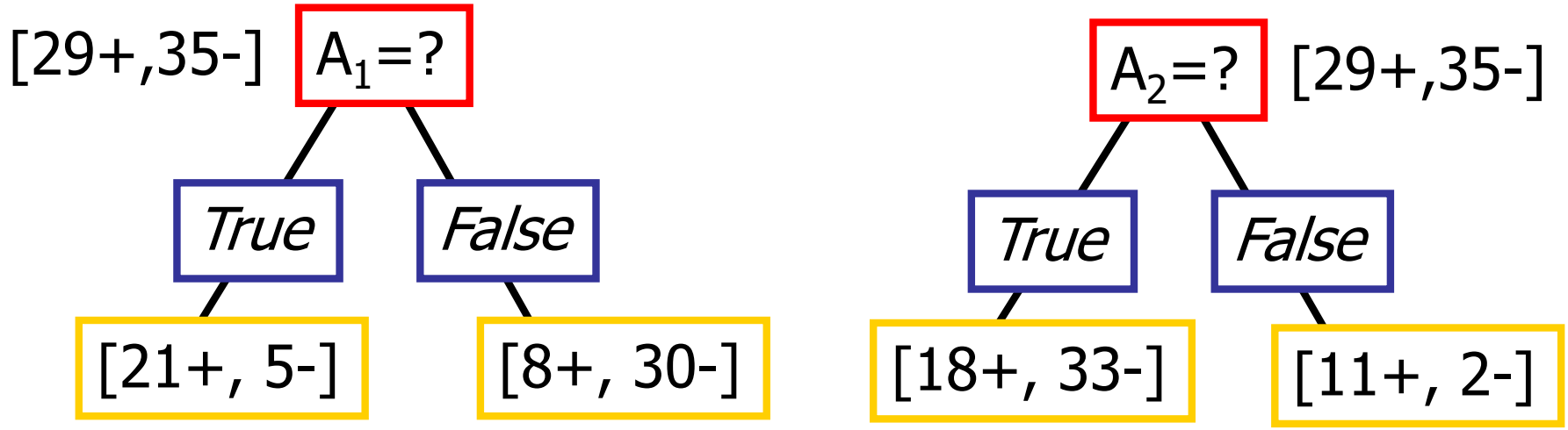
Information gain ratio is sometimes used instead. This biases the decision tree against considering attributes with a large number of distinct values. However, attributes with very low information values then appeared to receive an unfair advantage.

Information Gain

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

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

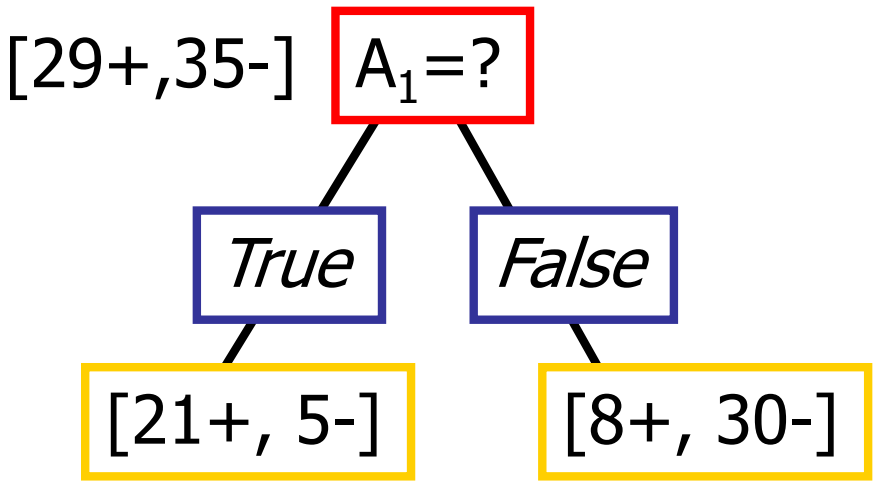
$$Entropy([29+,35-]) = -29/64 \log_2 29/64 - 35/64 \log_2 35/64 = 0.99$$

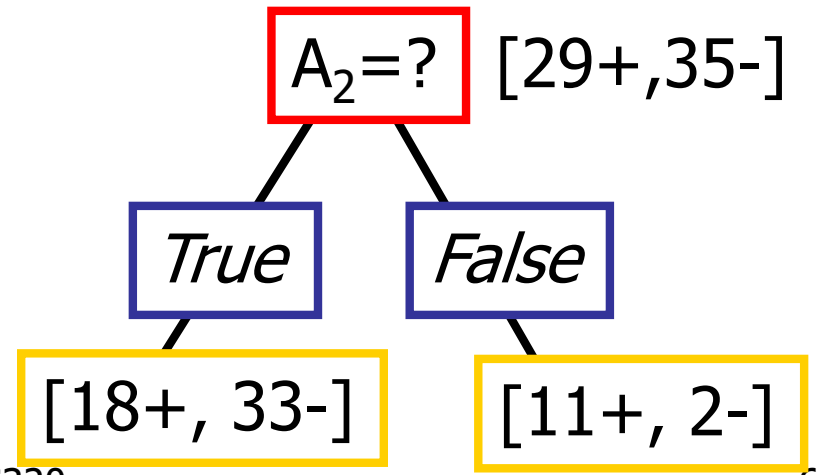


Information Gain

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

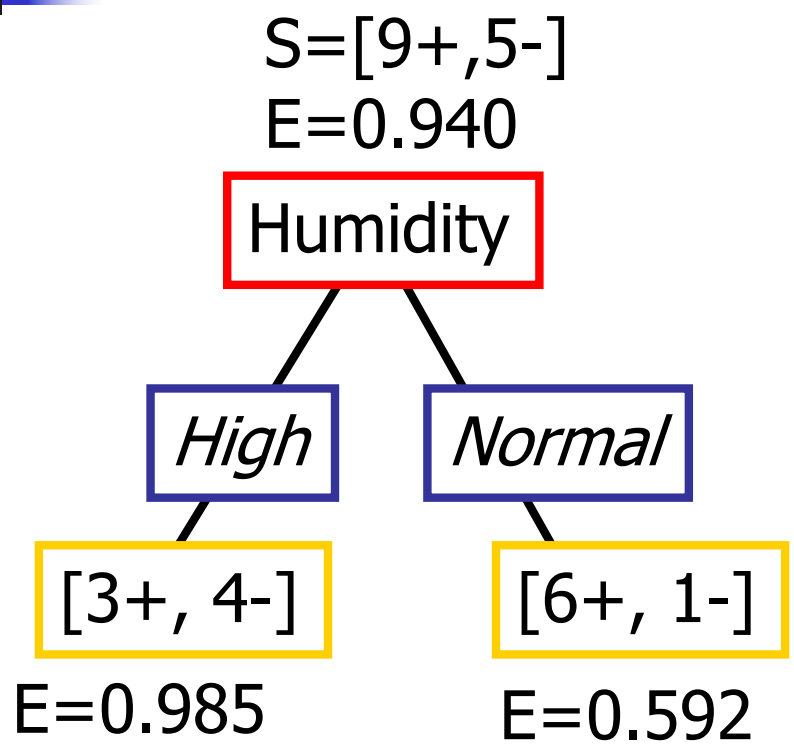


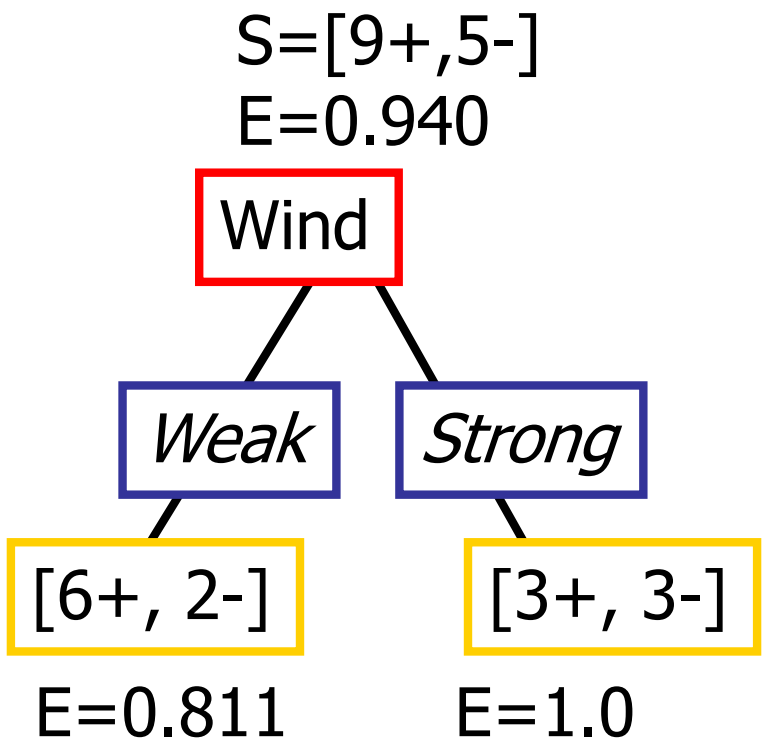


Training Examples

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



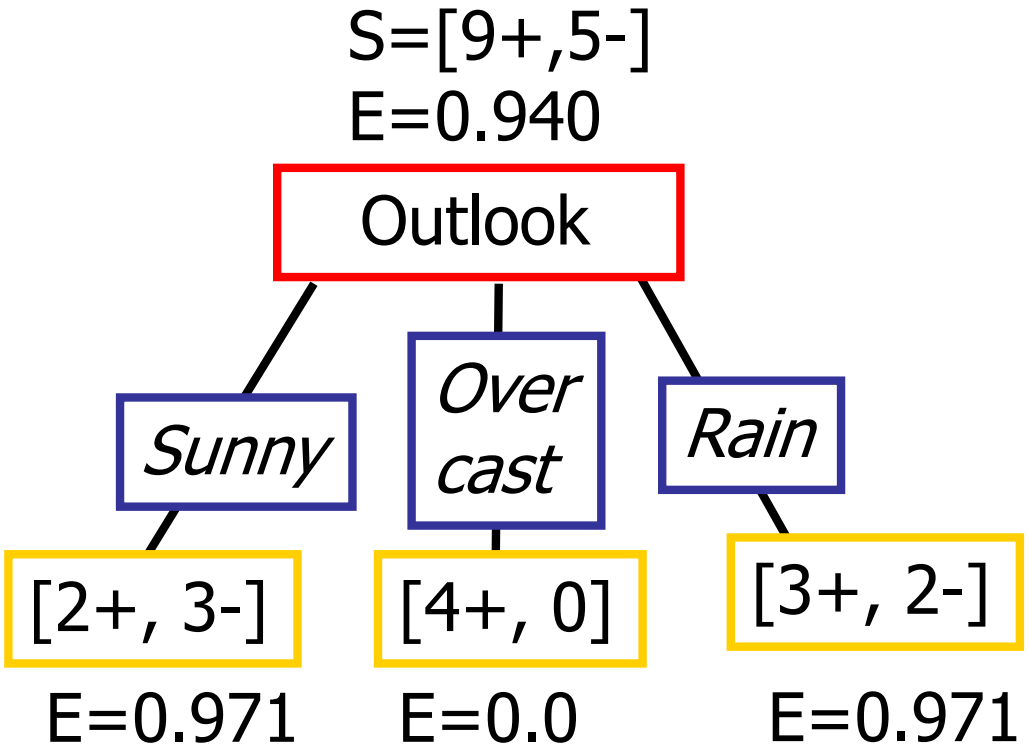


Gain(S,Humidity)
=0.940-(7/14)*0.985
– (7/14)*0.592
=0.151

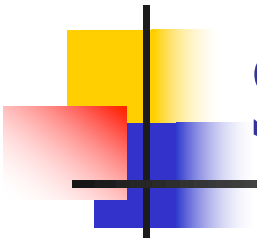
Gain(S,Wind)
=0.940-(8/14)*0.811
– (6/14)*1.0
=0.048

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:

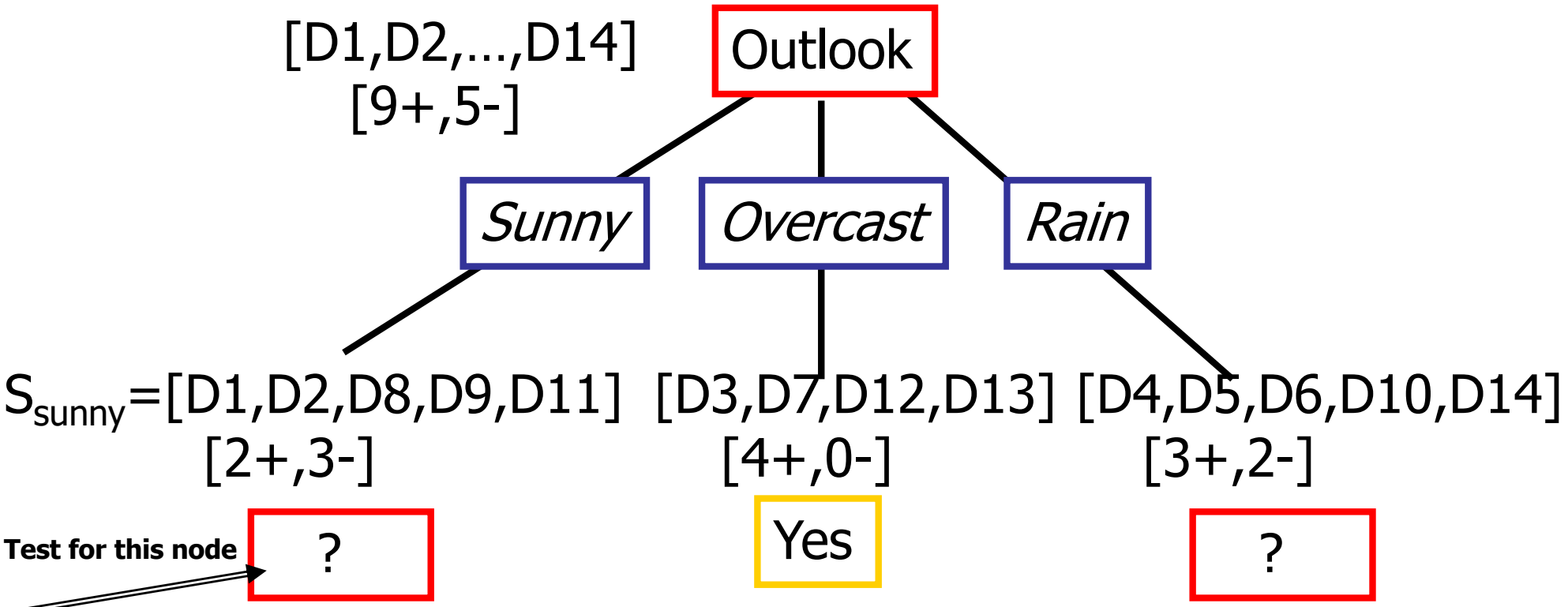
- Gain(S,Outlook) =0.247
- Gain(S,Humidity) =0.151
- Gain(S,Wind) =0.048
- Gain(S,Temperature) =0.029

where S denotes the collection of training examples

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

ID3 Algorithm

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

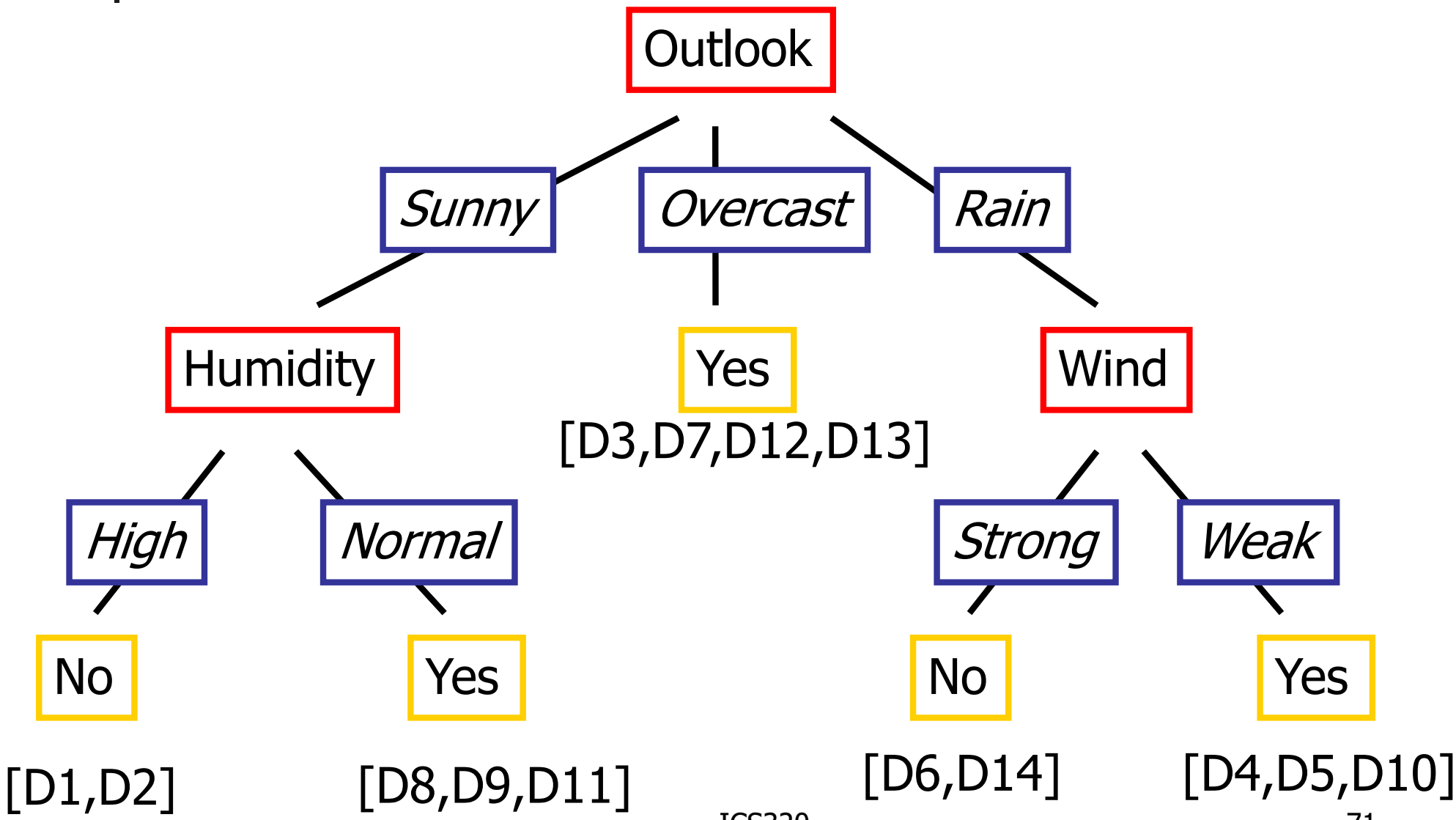


$\text{Gain}(S_{\text{sunny}}, \text{Humidity}) = 0.970 - (3/5)0.0 - 2/5(0.0) = 0.970$

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

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

ID3 Algorithm



IG is biased towards high cardinality features so use IG Ratio

Information gain calculation [edit]

Let $Attr$ be the set of all attributes and Ex the set of all training examples, $value(x, a)$ with $x \in Ex$ defines the value of a specific example x for attribute $a \in Attr$, H specifies the entropy. The $values(a)$ function denotes set of all possible values of attribute $a \in Attr$. The information gain for an attribute $a \in Attr$ is defined as follows:

$$IG(Ex, a) = H(Ex) - \sum_{v \in values(a)} \left(\frac{|\{x \in Ex | value(x, a) = v\}|}{|Ex|} \cdot H(\{x \in Ex | value(x, a) = v\}) \right)$$

The information gain is equal to the total entropy for an attribute if for each of the attribute values a unique classification can be made for the result attribute. In this case the relative entropies subtracted from the total entropy are 0.

Intrinsic value calculation [edit]

The intrinsic value for a test is defined as follows:

$$IV(Ex, a) = - \sum_{v \in values(a)} \frac{|\{x \in Ex | value(x, a) = v\}|}{|Ex|} \cdot \log_2 \left(\frac{|\{x \in Ex | value(x, a) = v\}|}{|Ex|} \right)$$

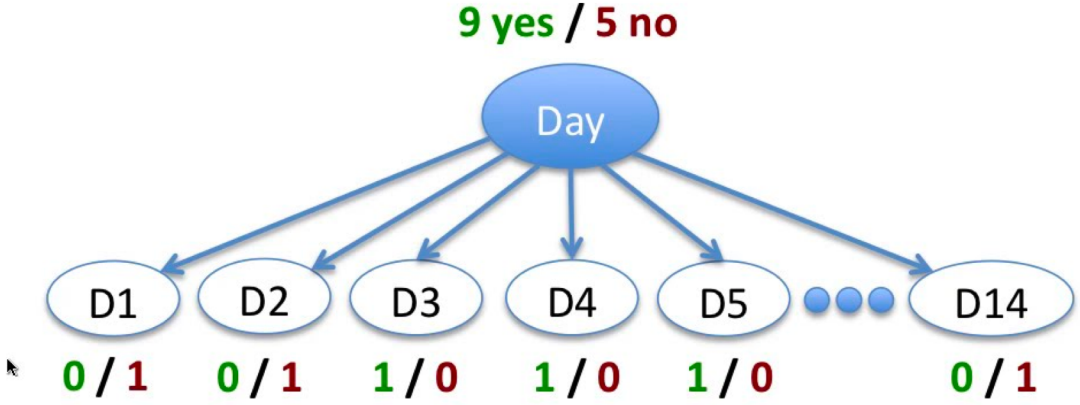
Entropy of new attribute a
Lots of splits means high entropy

Information gain ratio calculation [edit]

The information gain ratio is just the ratio between the information gain and the intrinsic value: $IGR(Ex, a) = IG/IV$

Problems with Information Gain

- Biased towards attributes with many values
- Won't work for new data: D15 Rain High Weak
- Use GainRatio:



$$SplitEntropy(S,A) = -\sum_{V \in Values(A)} \frac{|S_V|}{|S|} \log \frac{|S_V|}{|S|}$$

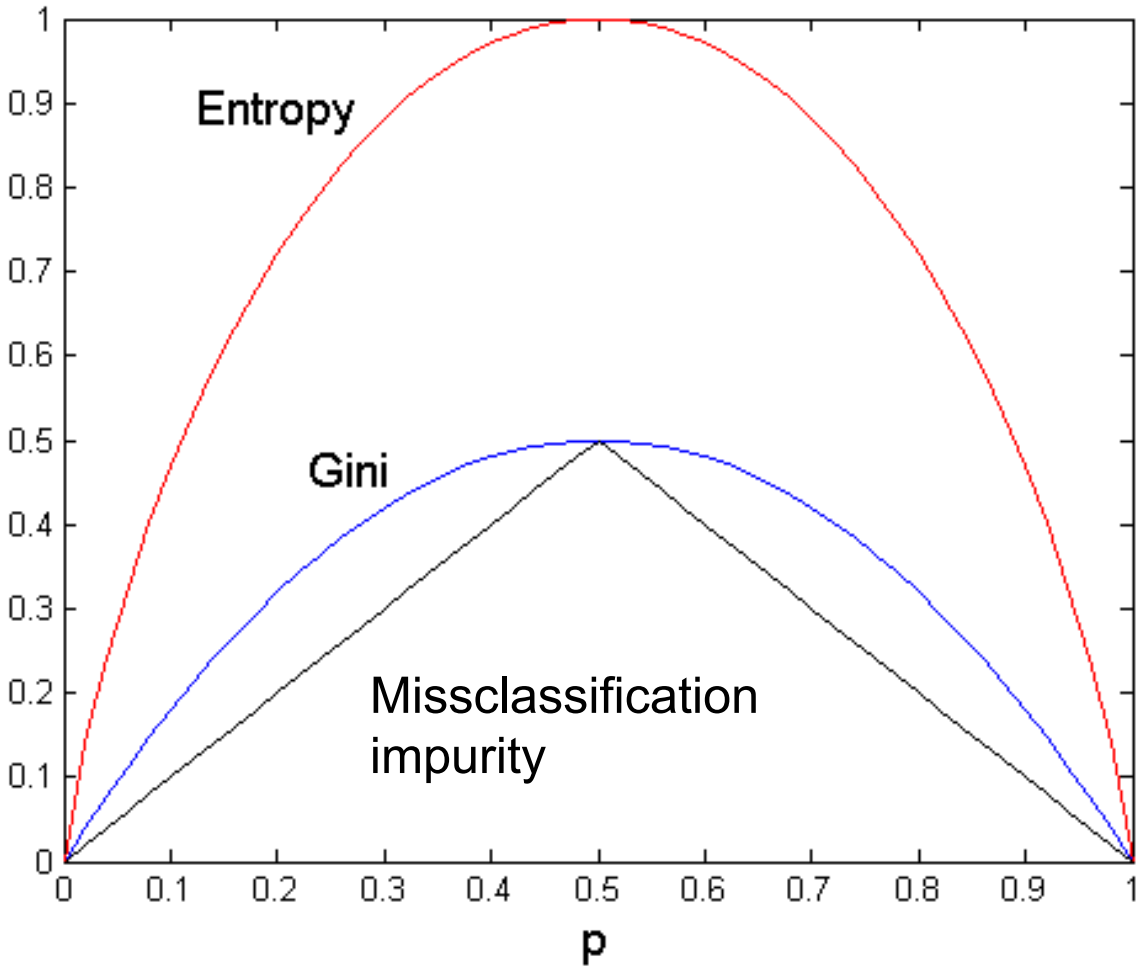
A … candidate attribute
V … possible values of A
S … set of examples {X}
S_v … subset where $X_A = V$

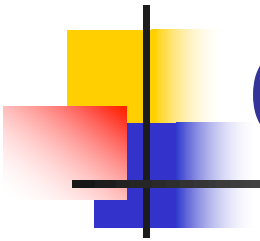
$$GainRatio(S,A) = \frac{Gain(S,A)}{SplitEntropy(S,A)}$$

penalizes attributes with many values

A is High entropy here

Impurities for Binary Classification





Occam's Razor

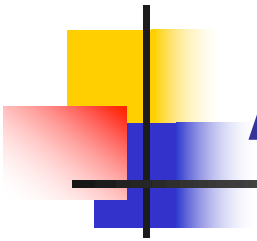
Why prefer short hypotheses?

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



Avoid Overfitting

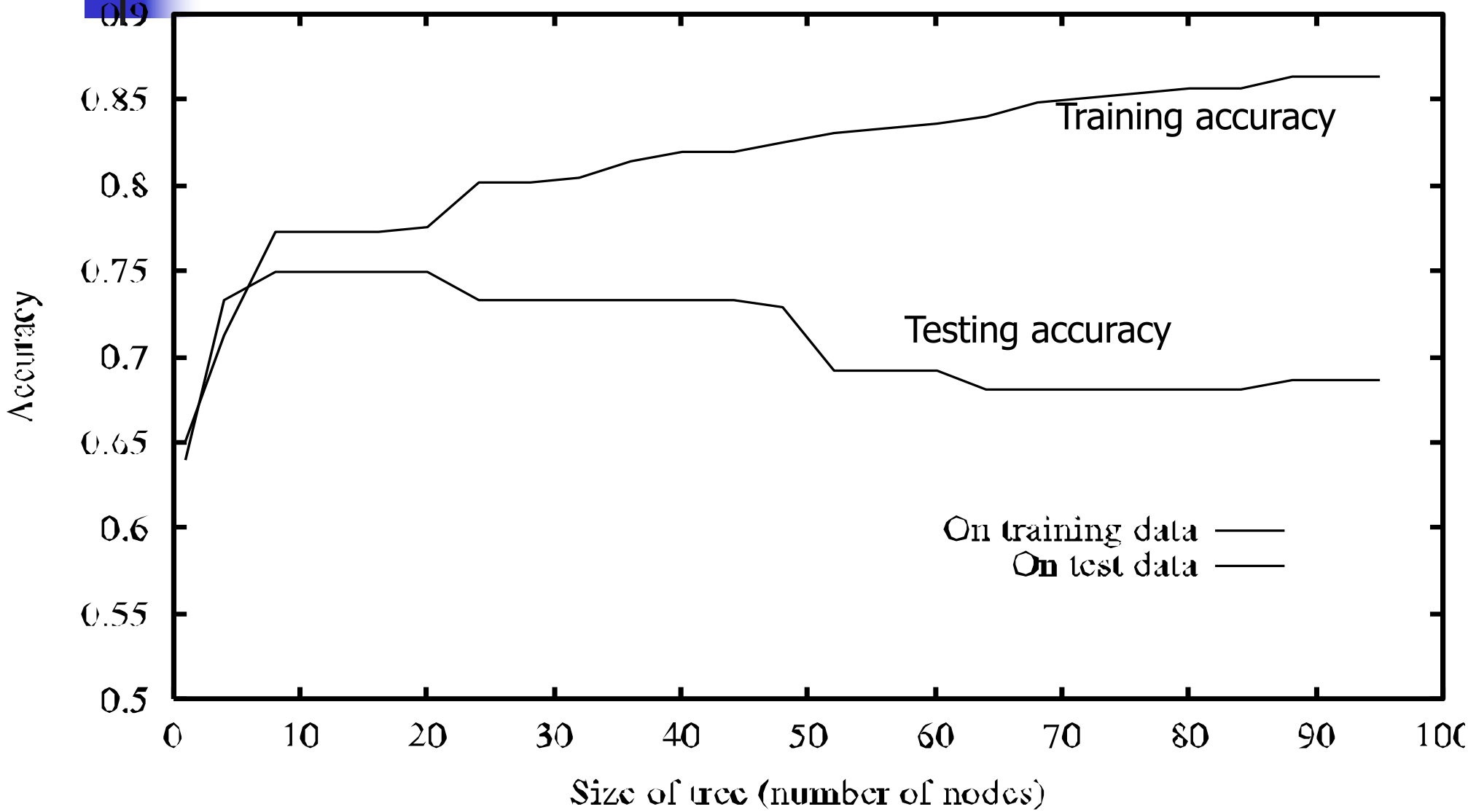
How can we avoid overfitting?

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

  Minimize:

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

Overfitting in Decision Tree Learning



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]

Continuous Input Attributes

- **Different ways of handling**

  Discretization to form an ordinal categorical attribute

  - Static – discretize once at the beginning
  - Dynamic – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.

- **For continuous input variables, splitting is done by calculating the information gain for each possible split and selecting the greatest (ID3 is known as a greedy algorithm).**

  Binary Decision: $(A < v)$ or $(A \geq v)$

  - consider all possible splits and finds the best cut
  - can be more compute intensive

The bias of ID3

- **All ML algorithms exhibit a bias**
  - a way of reducing the complexity of learning by
    - limiting search (*search bias*) or
    - favouring particular properties in the knowledge structure (*preference bias*)
- **ID3's bias is for *small trees***
    - trees which reach pure profiles quickly, i.e. are shallow
    - This is a *preference* bias
- **It is better to learn a small tree because:**
  - it corresponds to a *lesser* number of *more general* rules, i.e. with fewer conditions
    - and is therefore easier to comprehend
  - there is much less fragmentation of examples and so rules have much better support
    - and thus more likely to perform well
- **ID3 cannot guarantee to produce the smallest tree**
  - but it *tends* to

Live Session Outline

- Introduction
- Gradient descent versus non-gradient descent approaches
- Decision trees (DT)
- Learning decision trees
  - Classification DTs on discrete variables (X, Y)
  - Regression DTs over continuous variables (X,y)
- Ensembles of decision Trees
- Practical decision trees
- Summary

CART

- **After ID3, CART is probably the most widely-used decision tree learner**
- **Main differences are:**
  - Makes binary splits only
  - Works with continues, nominal, ordinal attributes
  - Can be used for classification and regression

Splitting variables

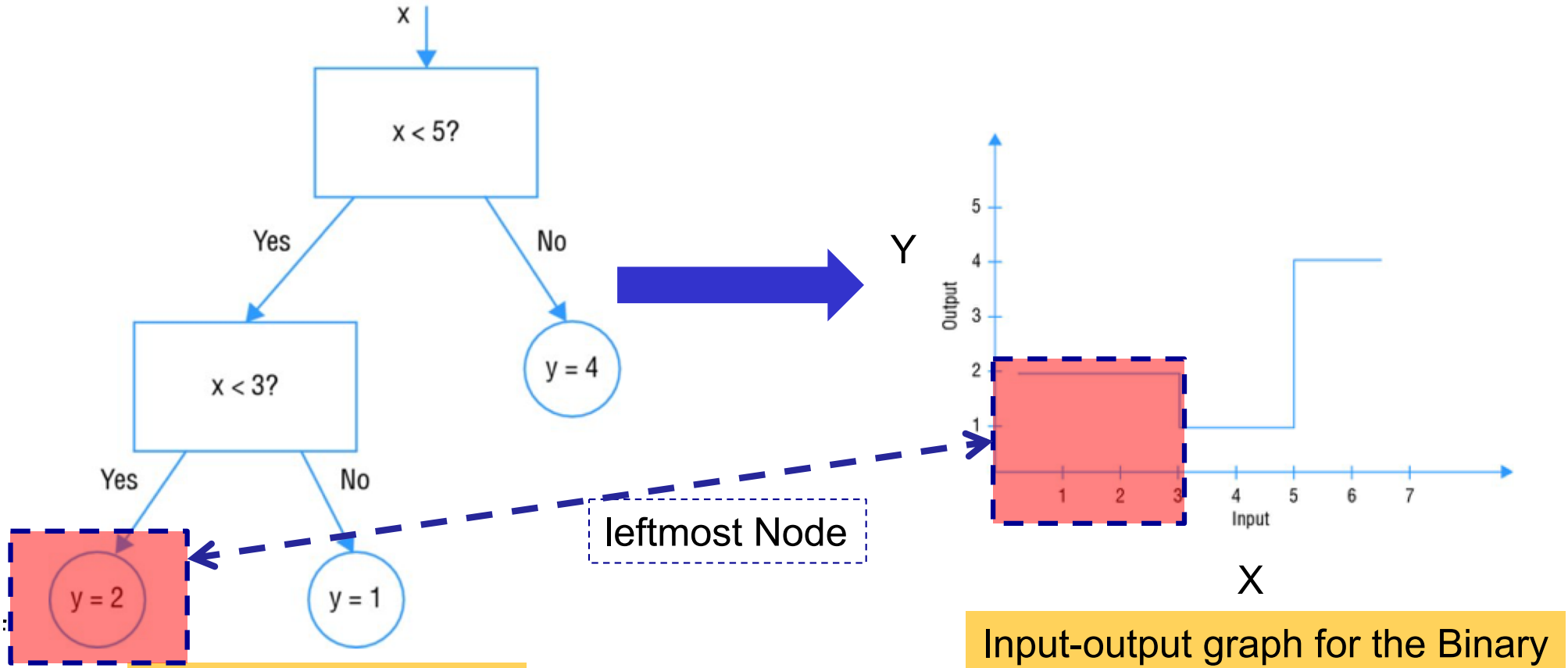
- **For continuous attributes a statistical measure called least squared deviation is employed instead of average entropy as described previously for ID3**
- **For discrete attributes it offers Gini and Twoing measures**

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**

Binary Regression Decision Tree

Test case X = 2; what is y



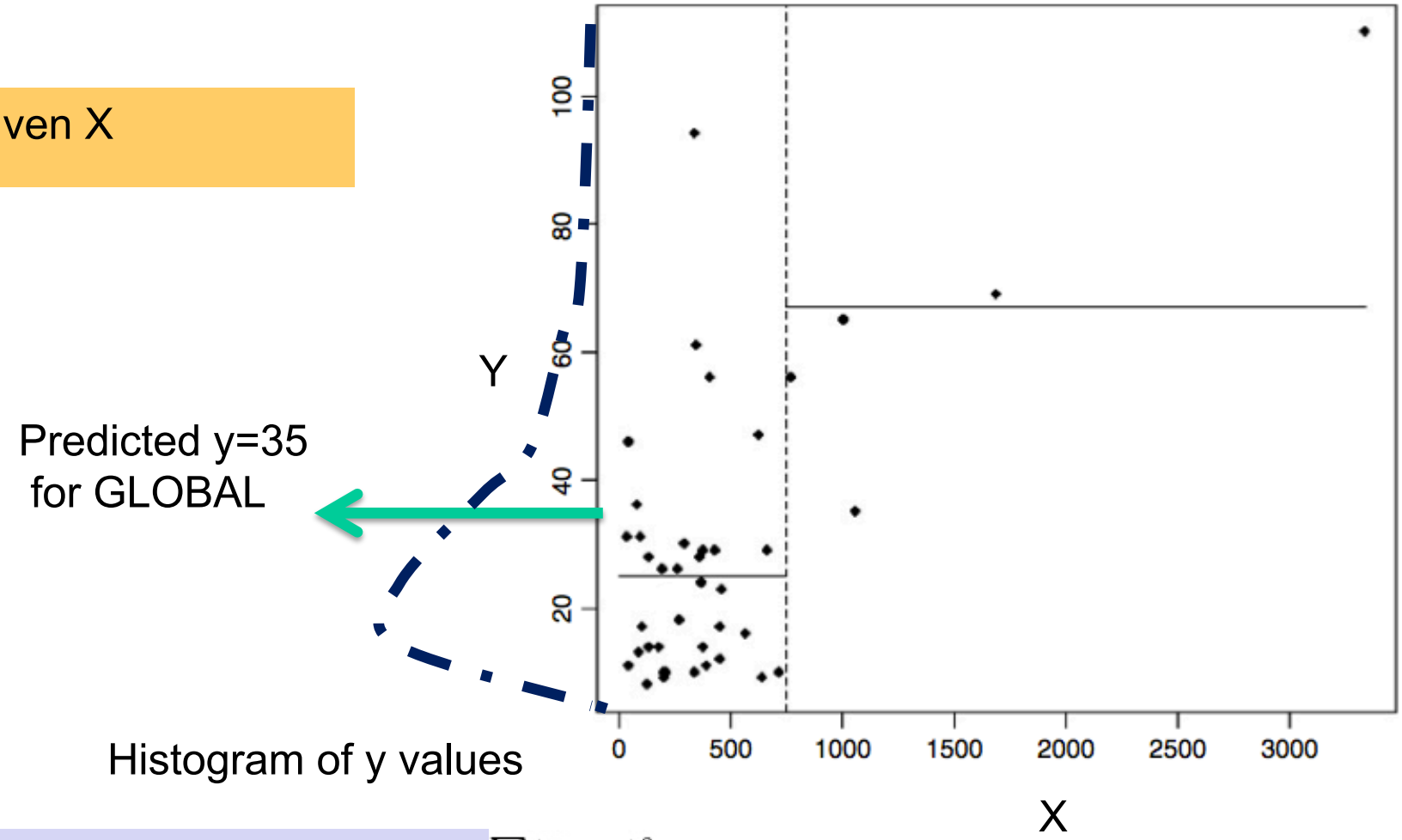
Binary Decision Tree

Input-output graph for the Binary Decision Tree example

Mean value prediction for regression trees with squared error loss

- **One region R_1, use Mean value of y**

Predict Y given X



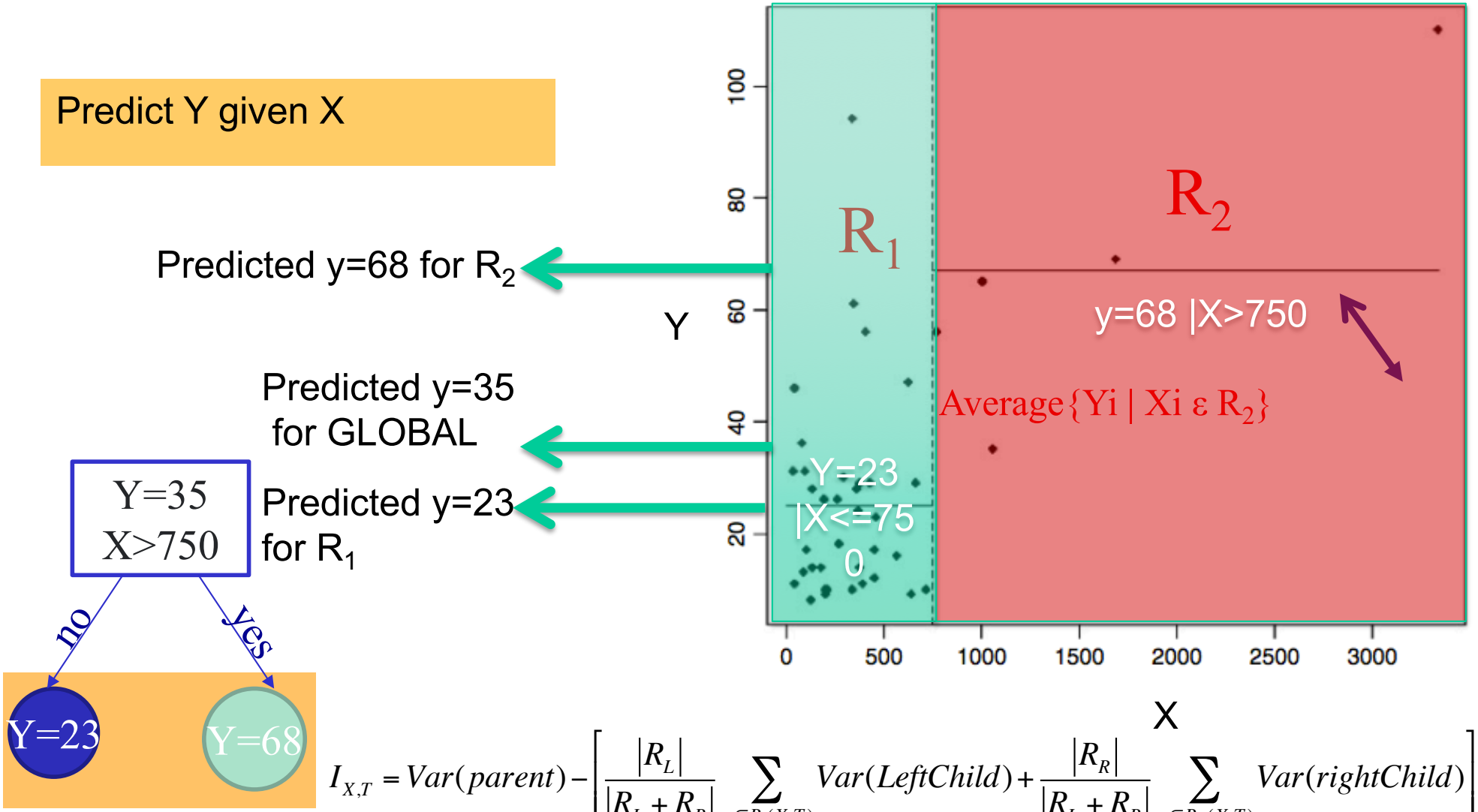
Histogram of y values

NOTE: Var(parent) = MSE(parent) $\frac{\sum(X-\mu)^2}{N}$

Mean value prediction for regression trees with squared error loss

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

Predict Y given X

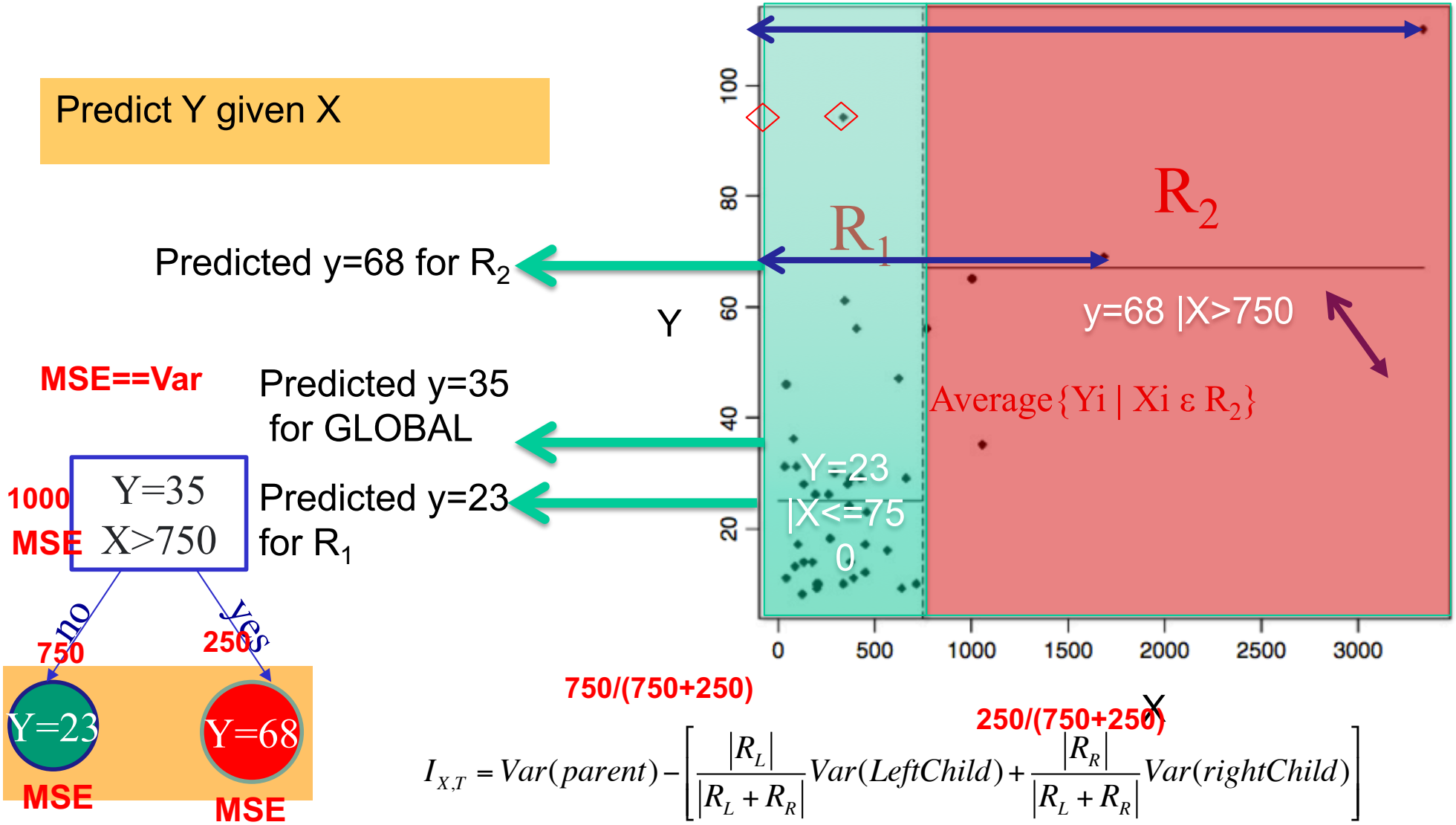


$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} Var(rightChild) \right]$$

Mean value prediction for regression trees with squared error loss

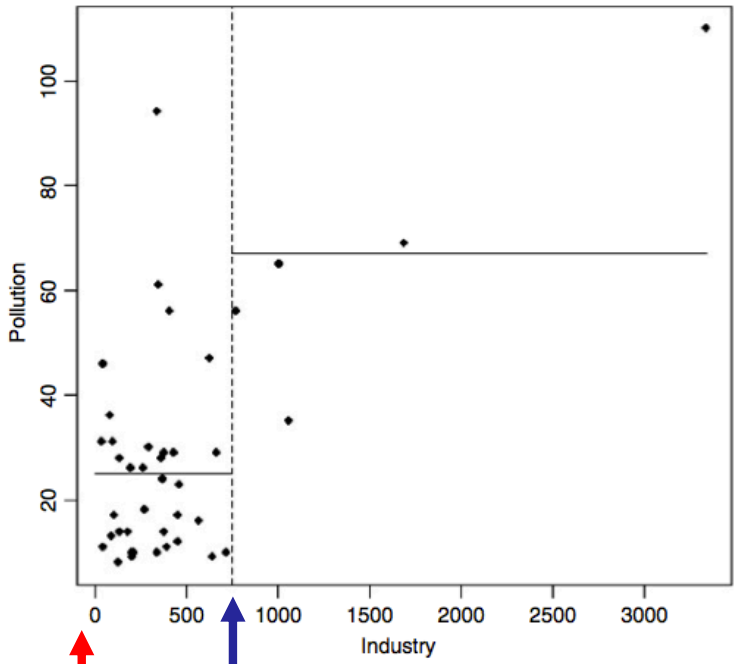
- **Partition the space into *M* regions: R_1, R_2**

Predict Y given X



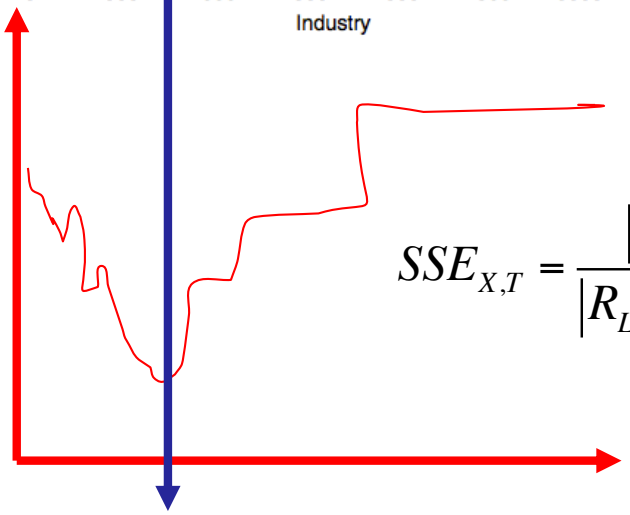
$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} Var(rightChild) \right]$$

Regression Tree: Find optimal first split point



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

$SSError_T$ As the split point of Industry is varied



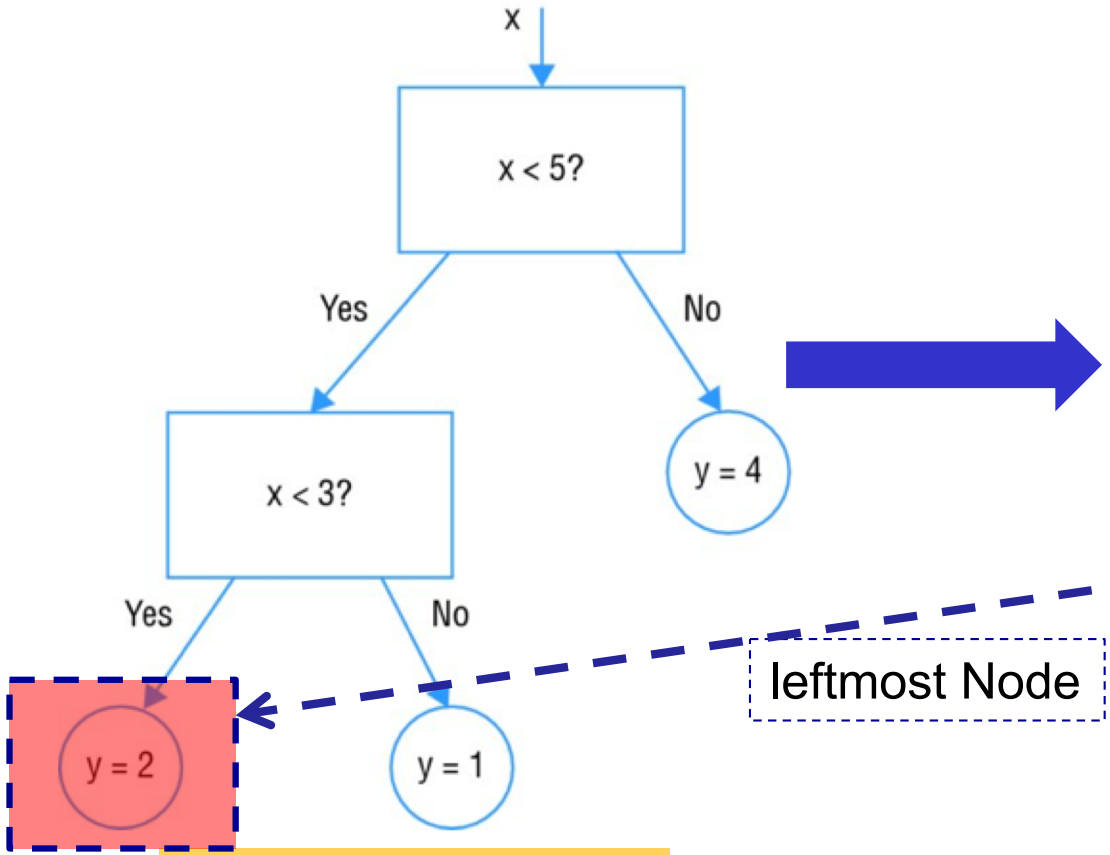
Optimal Split point

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

Binary Decision Tree: Missing Data

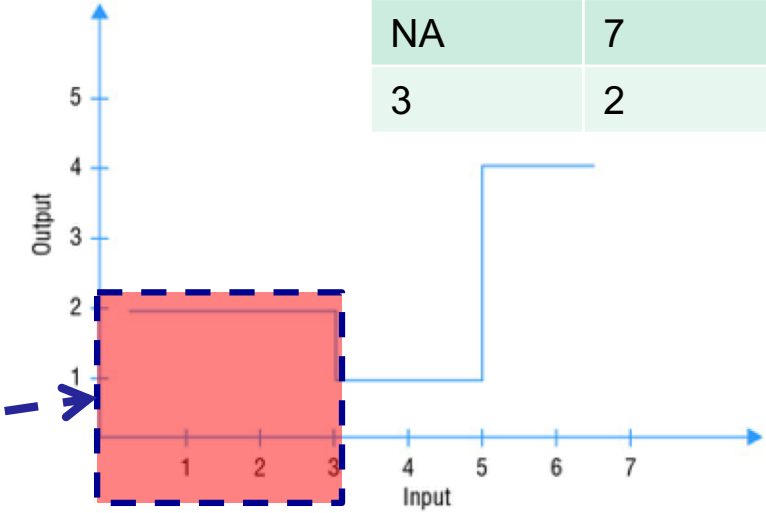
Replace by average
Delete records with missing values

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



Binary Decision Tree

leftmost Node

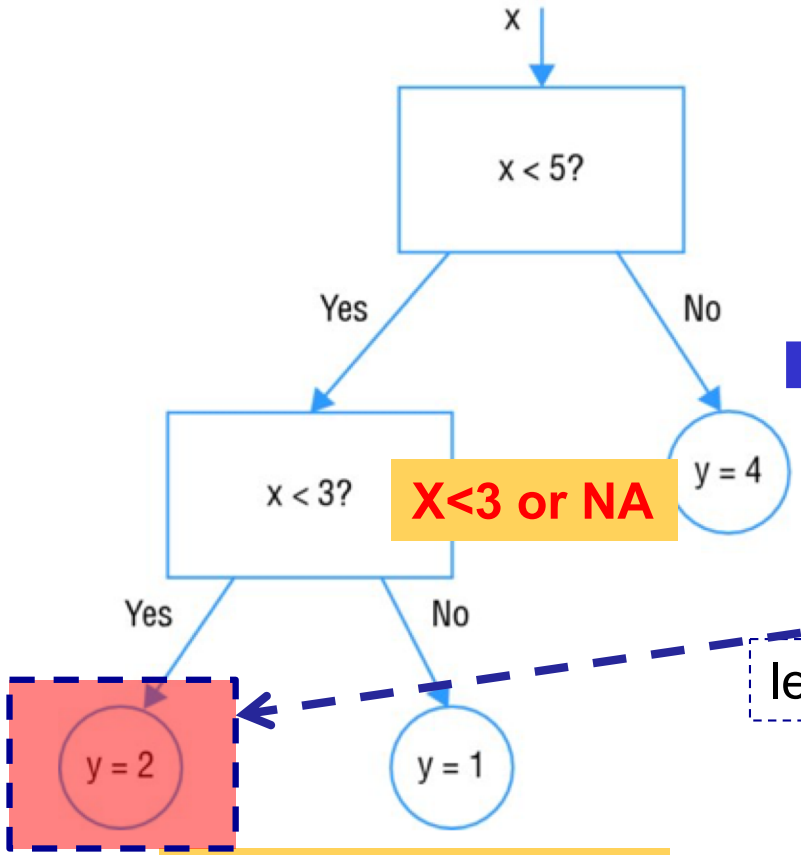


Input-output graph for the Binary Decision Tree example

Binary Decision Tree: Missing Data

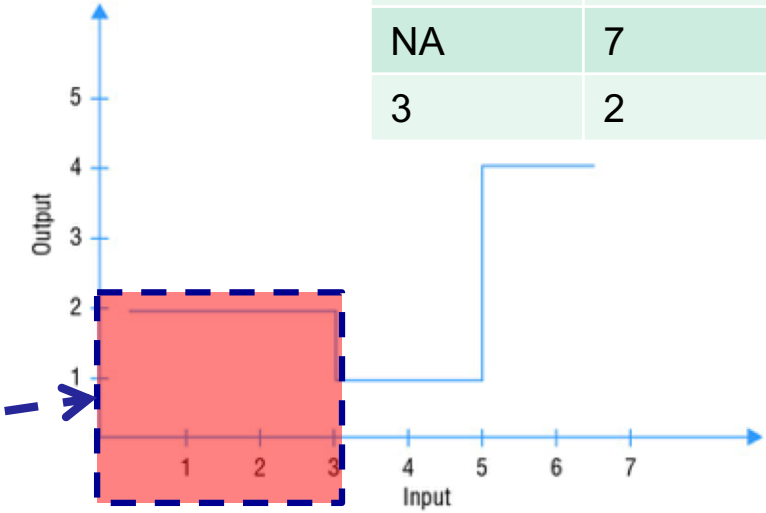
X	Y
1	2
3	2
5	4
6	4
3	1
NA	1.7
NA	7
3	2

X==NA



X<3 or NA

leftmost Node



Binary Decision Tree

Input-output graph for the Binary Decision Tree example

modes, medians and means and ML

Summary Statistics and Loss Functions

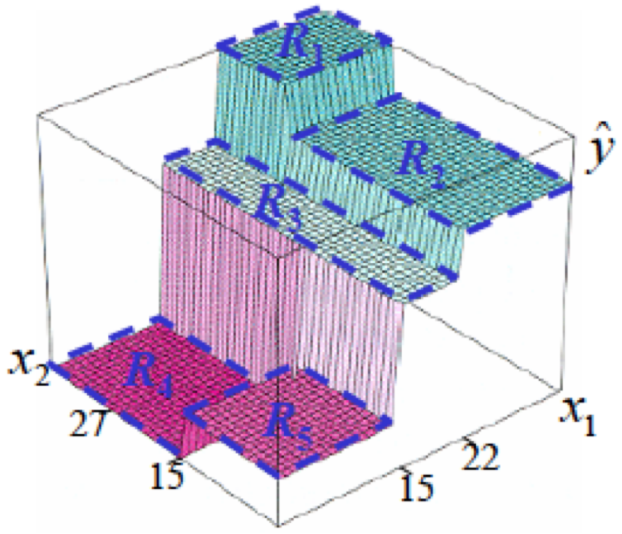
- To see how modes, medians and means arise, let's assume that we have a list of numbers, (x1,x2,…,xn), that we want to summarize (in terms of central behavior).
- We want our summary to be a single number, which we'll call s.
- How should we select s so that it summarizes the numbers, (x1,x2,…,xn),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

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 $x_i = \arg\min_s \sum_i |x_i - s|^2$

Trees in 3D and as conjunctive rules

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



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

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

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

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

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

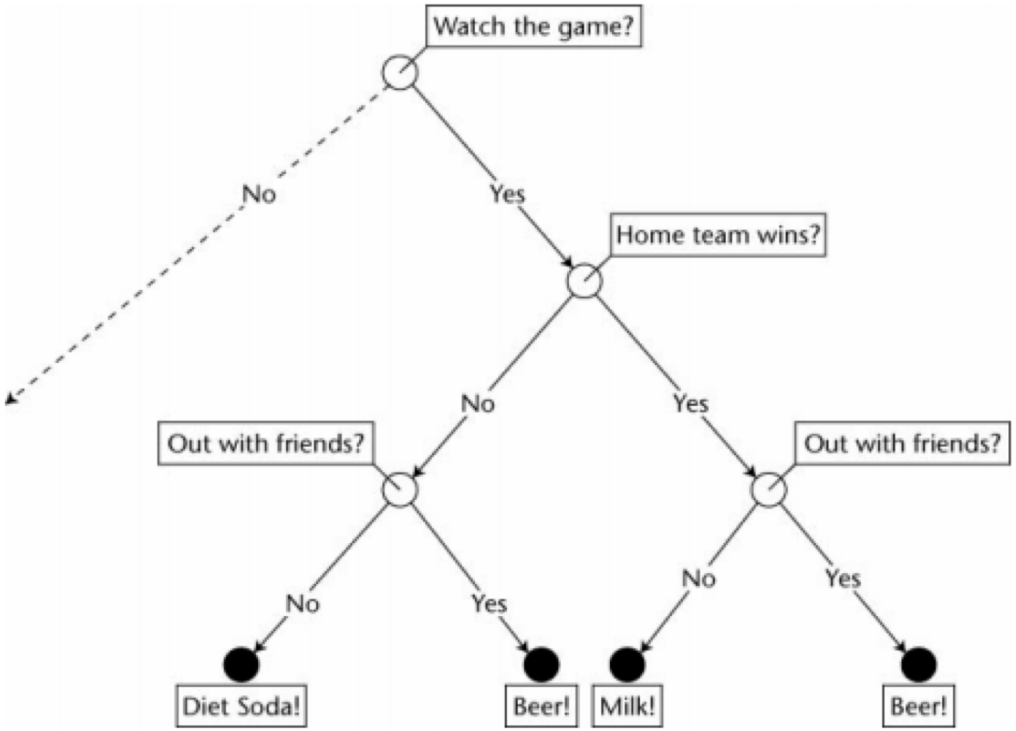
- These simple rules, $r_m(\mathbf{x}) \in \{0,1\}$, can be used as base learners
- Main motivation is *interpretability*

$T_m(\mathbf{x}) = \sum_{j=1}^{J} \hat{c}_{jm} I(\mathbf{x} \in \hat{R}_{jm})$ 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

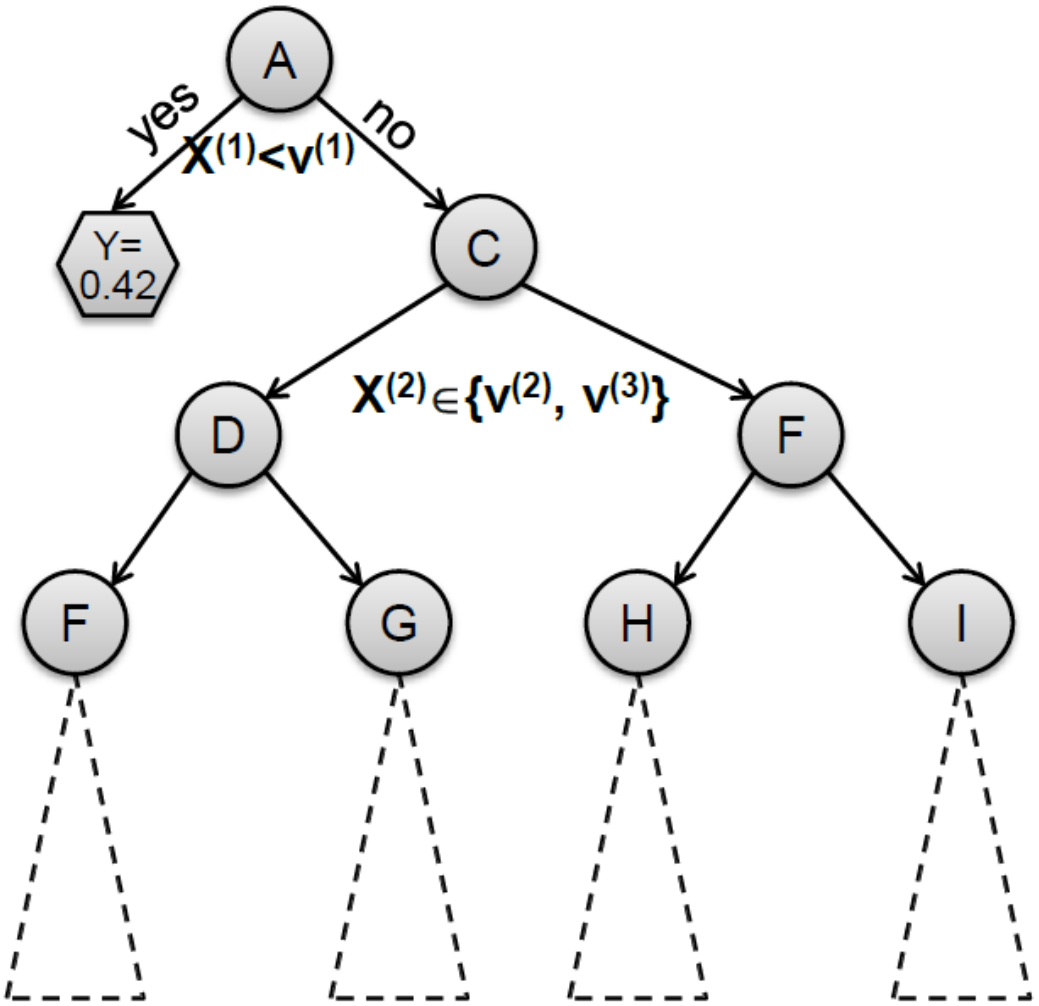
C5.0 – further generalized rules
More compact description

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

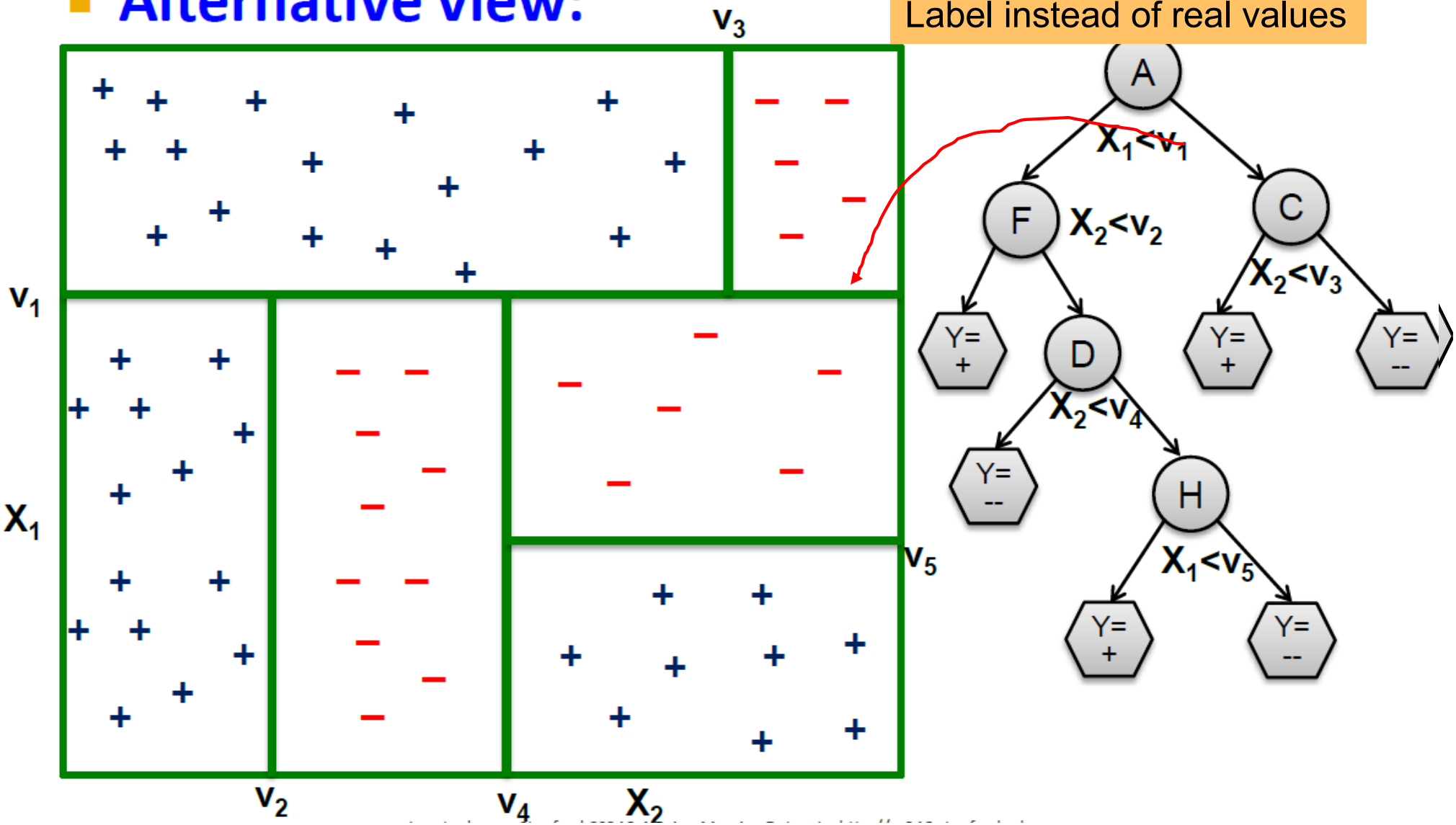
- A **Decision Tree** is a tree-structured plan of a set of attributes to test in order to predict the output



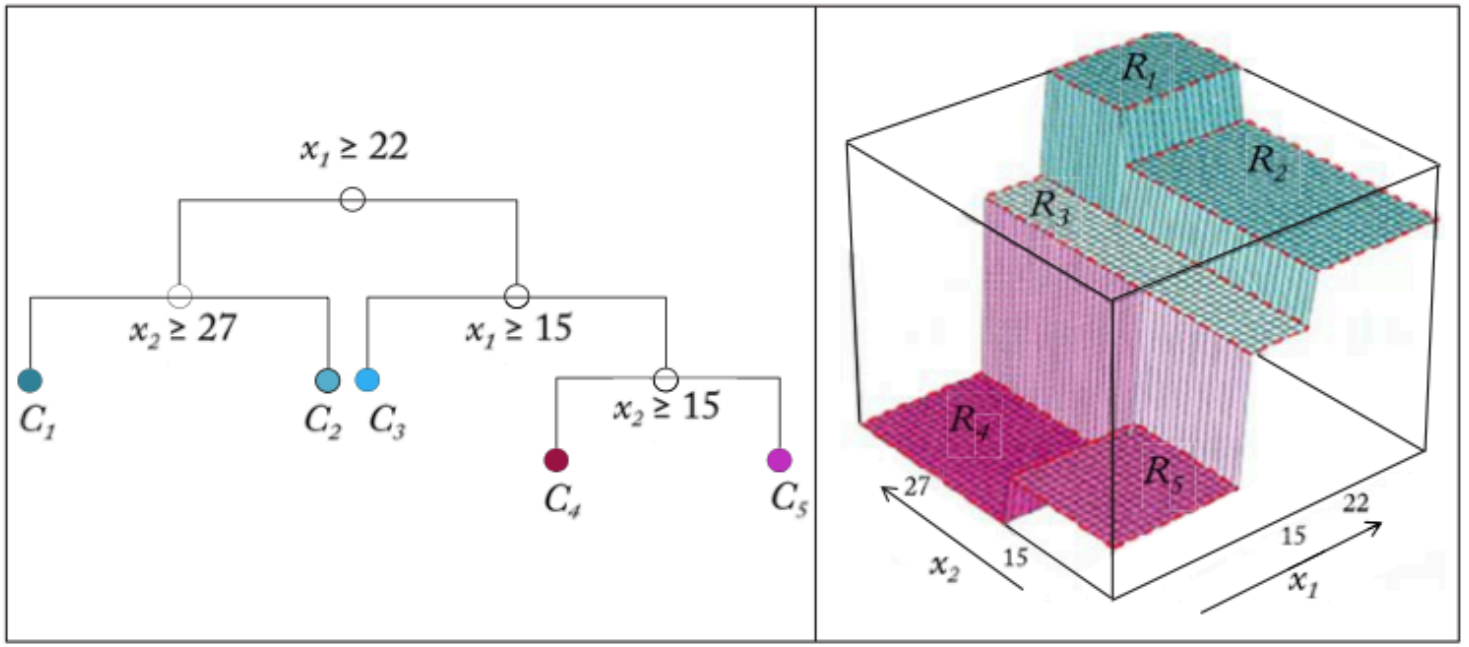
Classification Decision Trees Vs. SVM

- **Alternative view:**

Leaf nodes
Label instead of real values



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)

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 $\hat{}\mathbf{e(R_{Left})}$ and $\hat{}\mathbf{e(R_{Right})}$ computed.

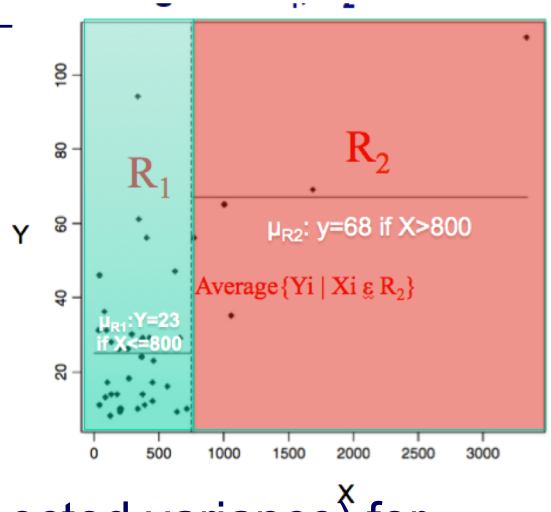
Three Steps in constructing a tree

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** $(n \rightarrow \text{right}, D_R)$

Find best split point for single input and target



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

FindBestSplit(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 $x_i > T$

}

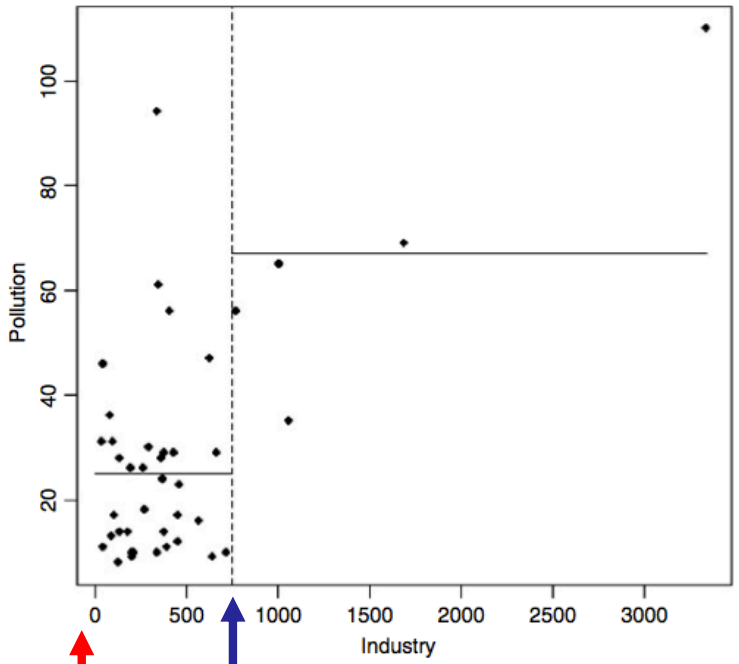
SplitPoint = Select threshold that has minimum SSE

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

$$return\left(\underset{T,\mu_{R1},\mu_{R2}}{\text{argmax}}(SSE_{X,T}),\ SSE_{X,T} \right)$$

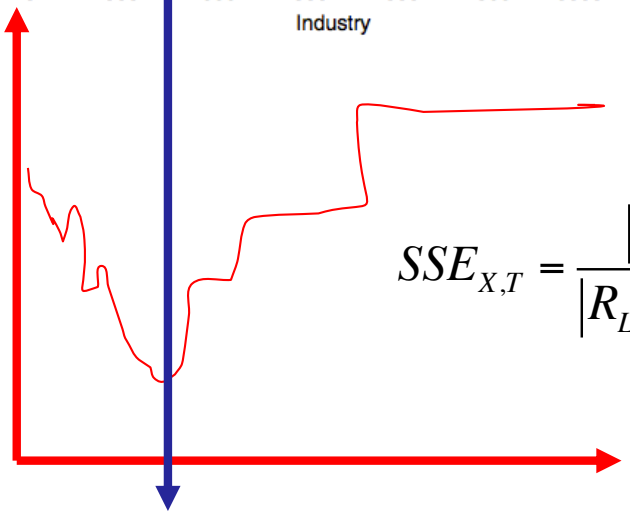
}

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



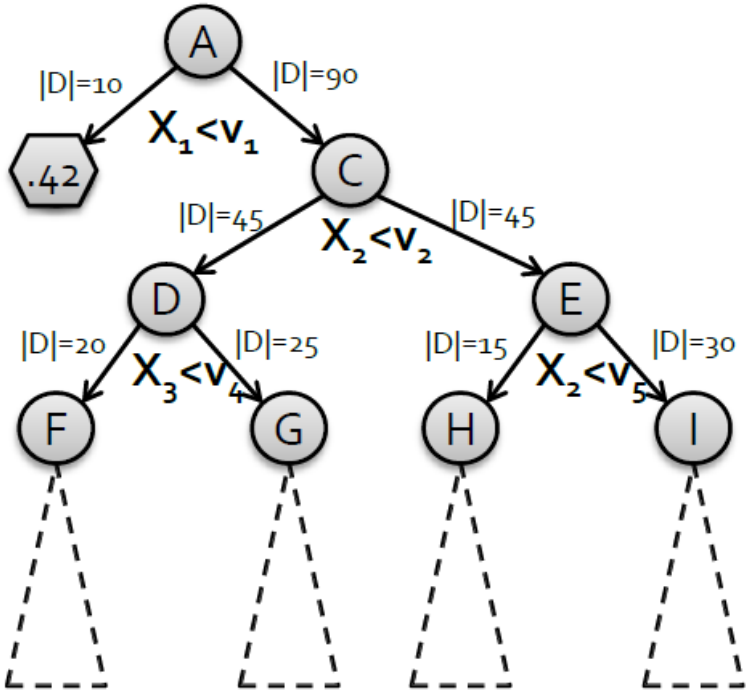
Optimal Split point

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

Stop Criterion

(2) When to stop?

- Many different heuristic options
- **Two ideas:**
  - **(1) When the leaf is "pure"**
    - The target variable does not vary too much: $Var(y_i) < \varepsilon$
  - **(2) When # of examples in the leaf is too small**
    - For example, $|D| \leq 100$



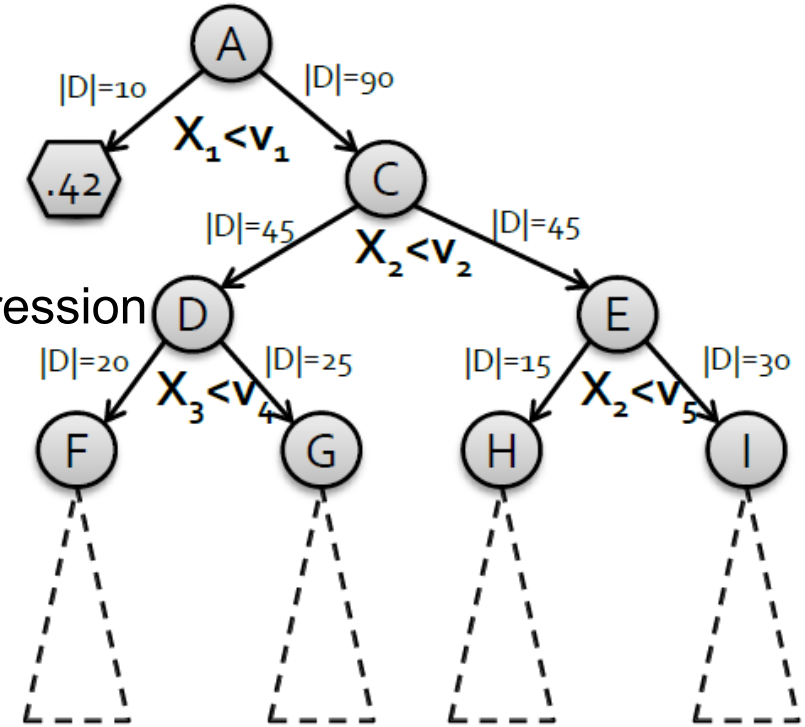
(3)Uncertainty in the node is low,
e.g., number of positive events is low (CTR)

Learn predicted value at leave node?

(3) How to predict?

- **Many options**
  - **Regression:** + binomial logistic regression
    - Predict average y_i of the examples in the leaf
    - Build a linear regression model on the examples in the leaf
  - **Classification:**
    - Predict most common y_i of the examples in the leaf

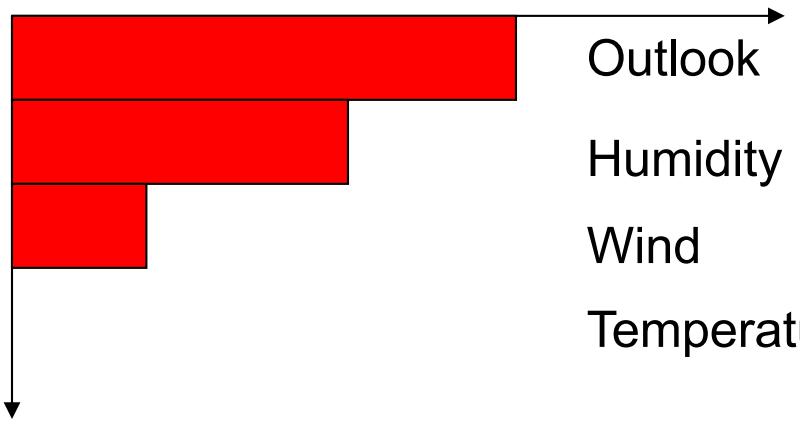
MSE Loss



Variable importance

https://www.r-bloggers.com/a-brief-tour-of-the-trees-and-forests/

- **In many applications, all variables do not contribute equally in predicting the output.**
- **We can evaluate variable importance by computing the total reduction of impurity brought by each variable:**
  - $\text{Imp}(A)=\sum_{\text{nodes where } A \text{ is tested}} |LS_{\text{node}}|\ \Delta\text{I}(LS_{\text{node}},A)$



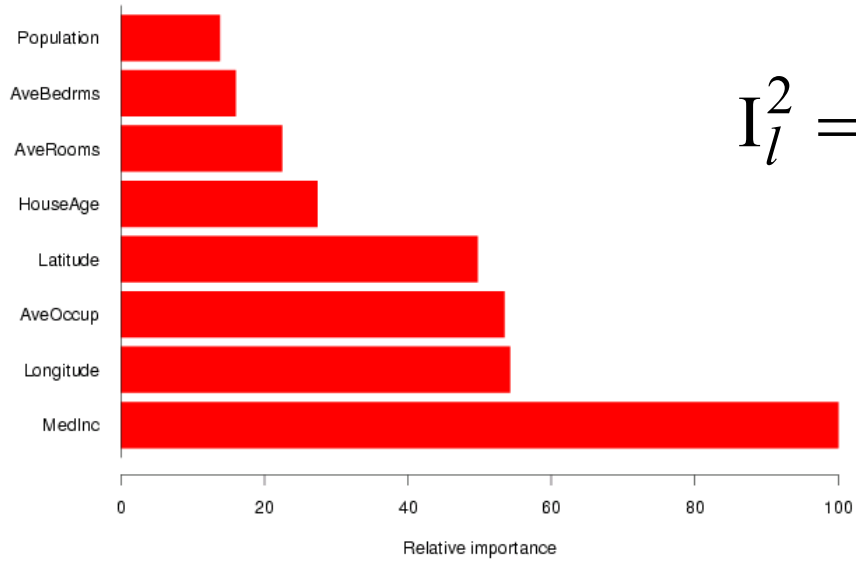
Big improvements at the Root node or near the root

Relative Importance of Variables

- For a single tree, define the importance of x_l as

- For additive tree, define the importance of x_l as

$$I_l^2(T) = \sum_{\text{over all node using } x_l \text{ for partition}} \text{improve in square error risk over for a constant fit over the region}$$



$$I_l^2 = \frac{1}{M}\sum_{m=1}^{M} I_l^2(T_m)$$

Figure 10.14: *Relative importance of the predictors for the California housing data.*

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

Live Session Outline

- Introduction
- Gradient descent versus non-gradient descent approaches
- Decision trees (DT)
- Learning decision trees
  - Classification DTs on discrete variables (X, Y)
  - Regression DTs over continuous variables (X,y)
- Ensembles of decision Trees
- Practical decision trees
- Summary

Dtree Limitations

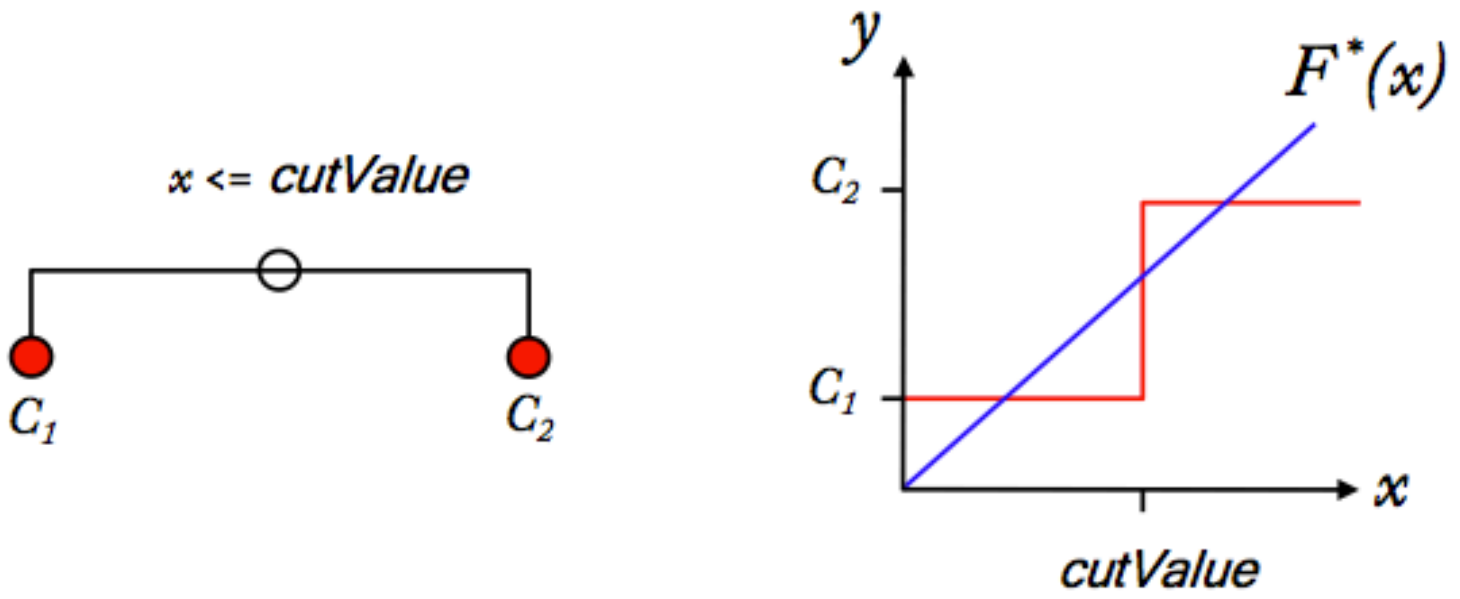


Figure 2.6: A 2-terminal node tree approximation to a linear function.

Single Dtree versus and ensemble

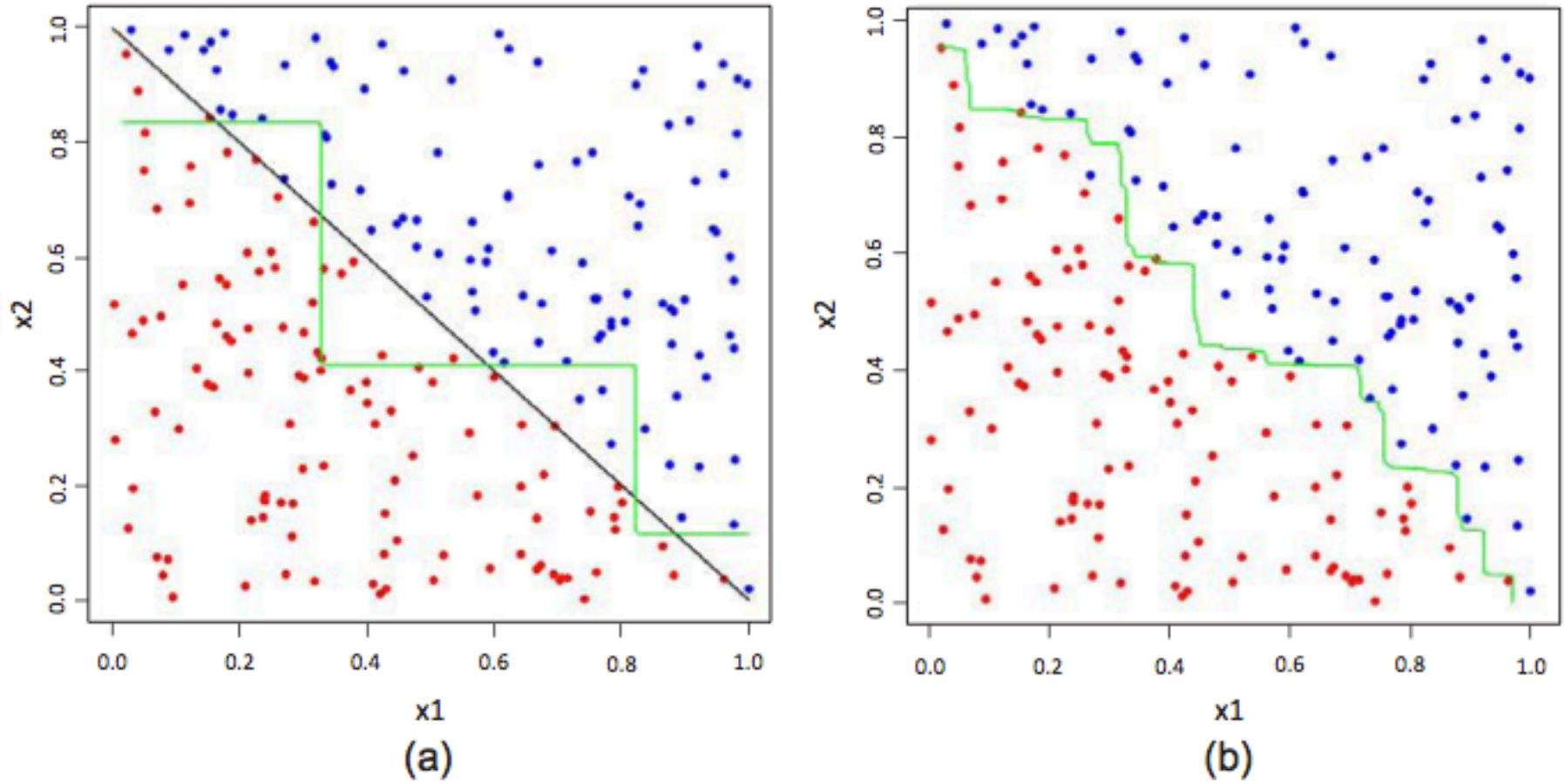
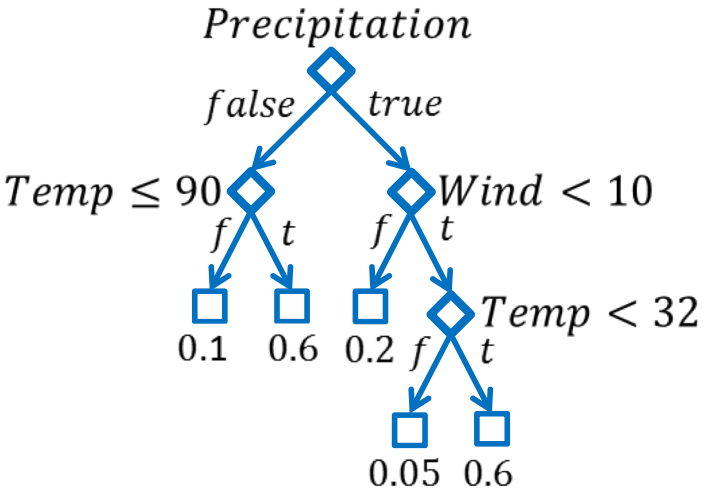


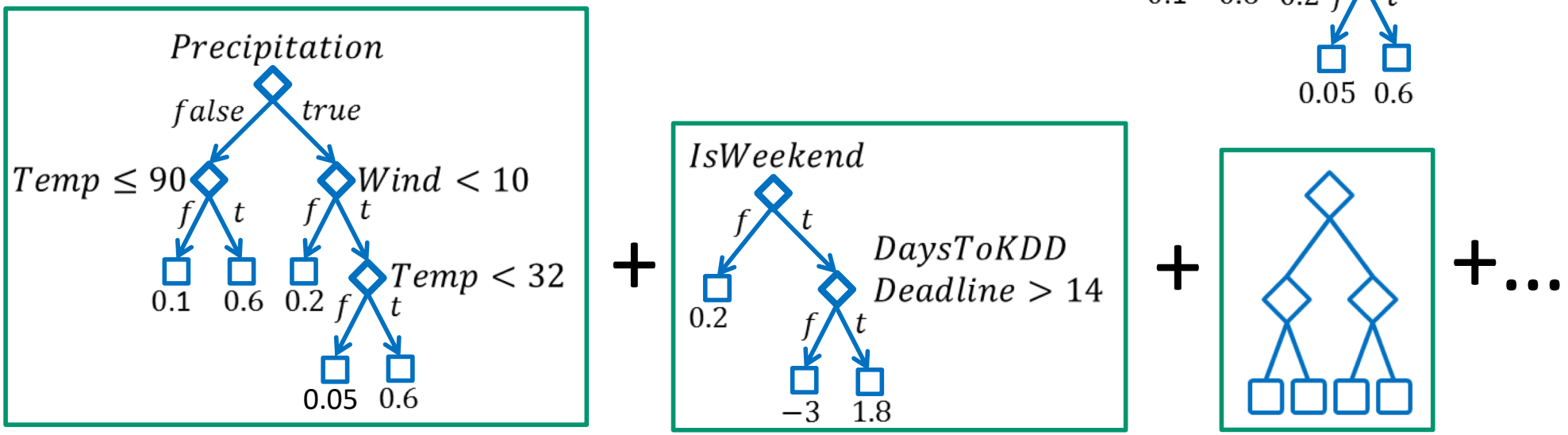
Figure 4.1: Comparison of decision boundaries generated by a single decision tree (a) and a Boosting-based ensemble model (b). The true boundary is the black diagonal. The ensemble approximation is superior.

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**



Ensemble Algorithms

- **Bagging**
- **Random forests**
- **Boosting (adaBoost, gradient boosted decision trees)**

- **Hyperparameters**
  - Number of trees
  - Depth of trees
  - Learning rate (for gradient boosted decision trees)

Bagging Algo. for classification or regression

Training

- **Build a set of classifiers using several training samples each sampled from the given dataset (with replacement)**

Predicting

- **Classification: Voting over all predictions**

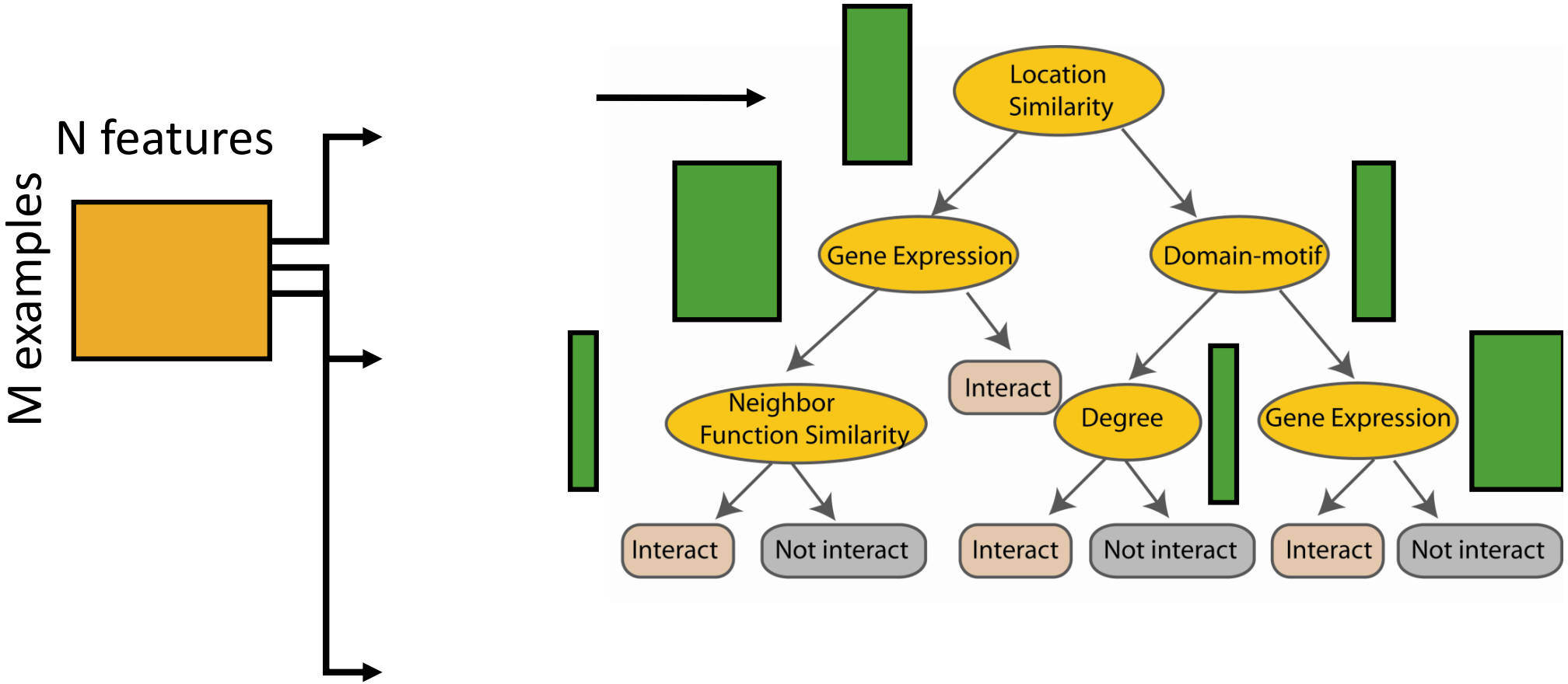
  Regression: Averaging over all predictions

Advantage

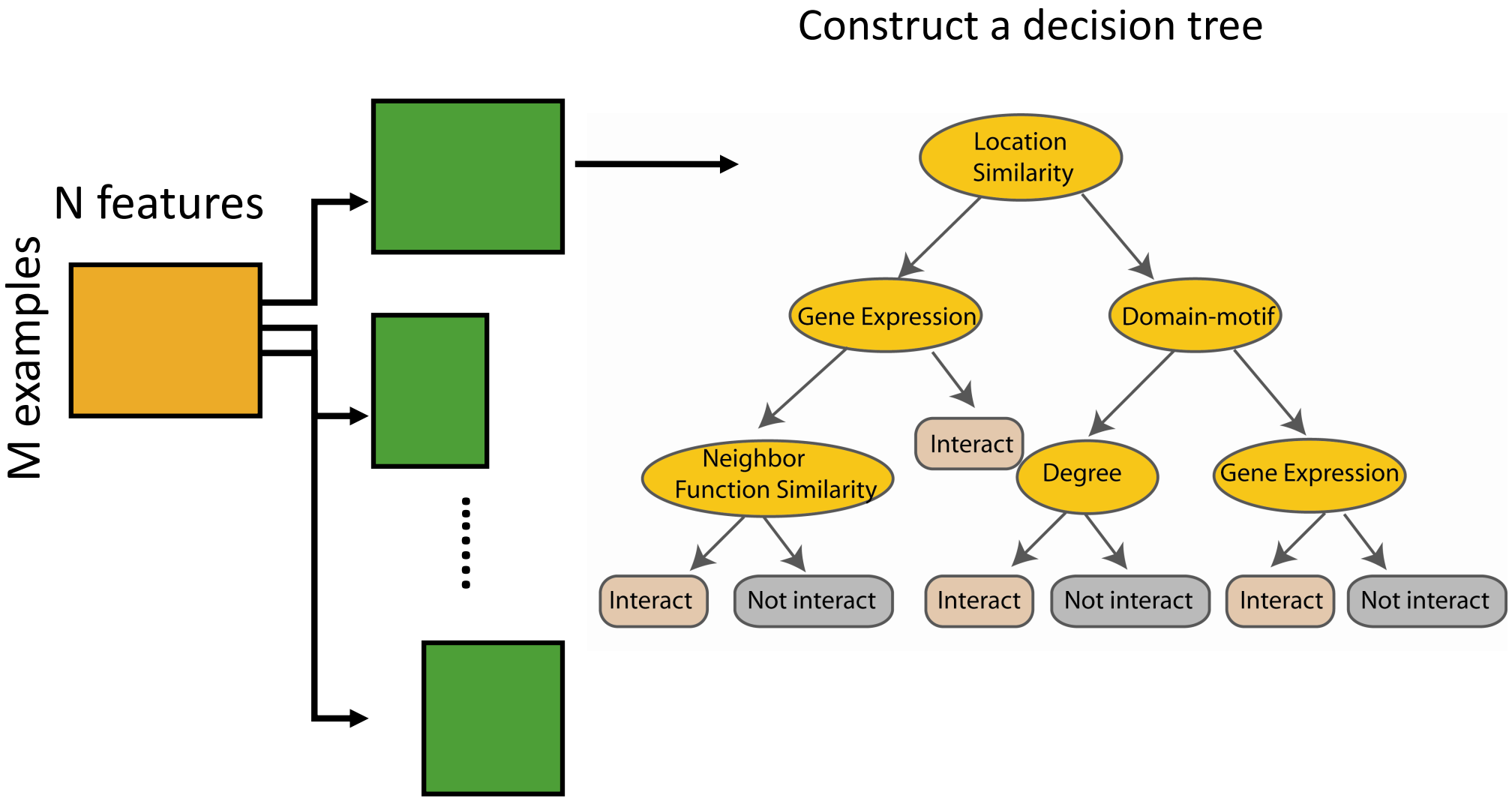
- **Reducing variance because of averaging over several predictions**
- **Reducing bias when the classifier tends to overfit the data (it has a low "inductive bias")**

Random Forest Classifier
Bagging with random feature subsets

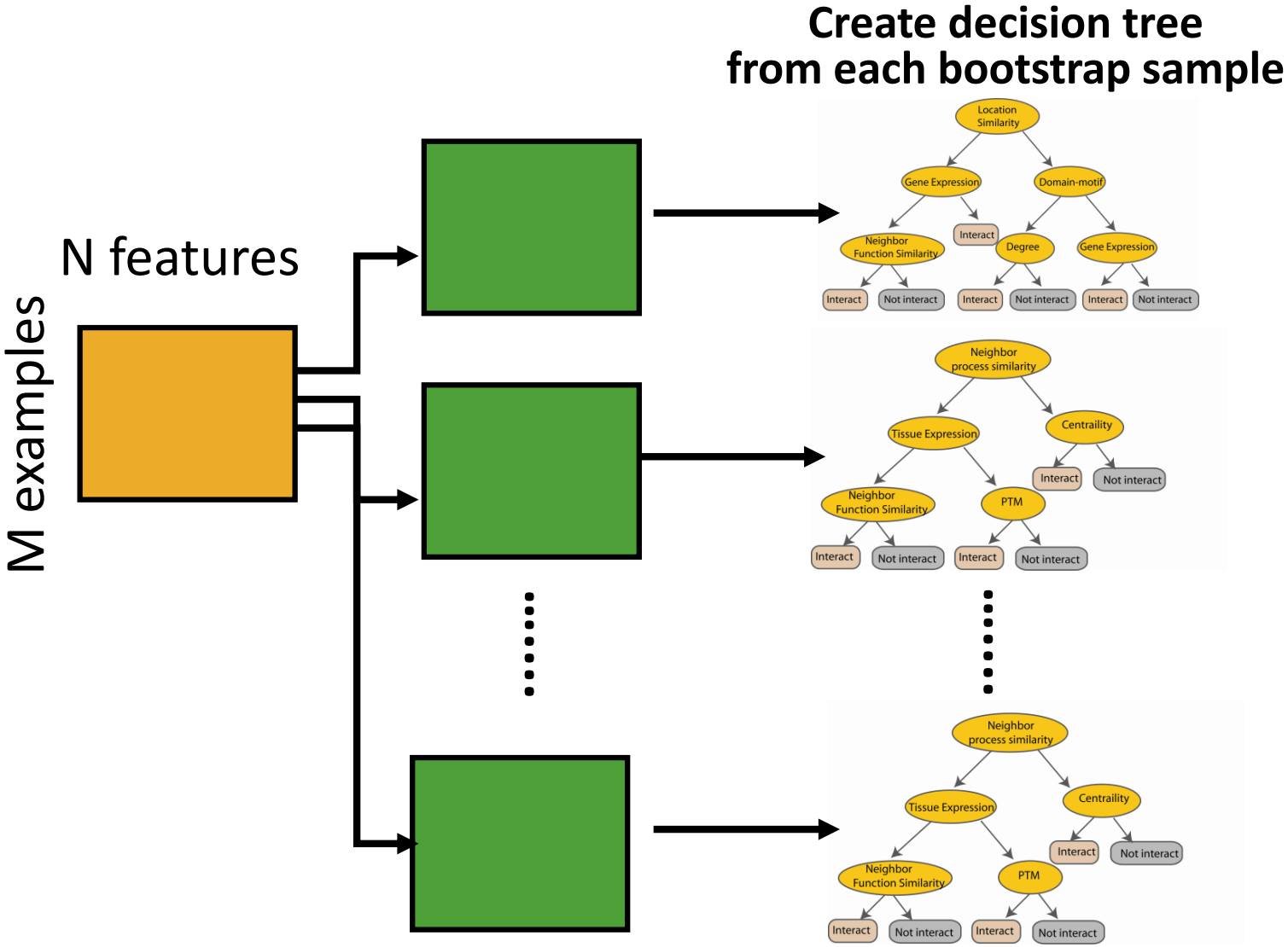
At each node in choosing the split feature
choose only among $n<N$ features (generally select log(N) features)



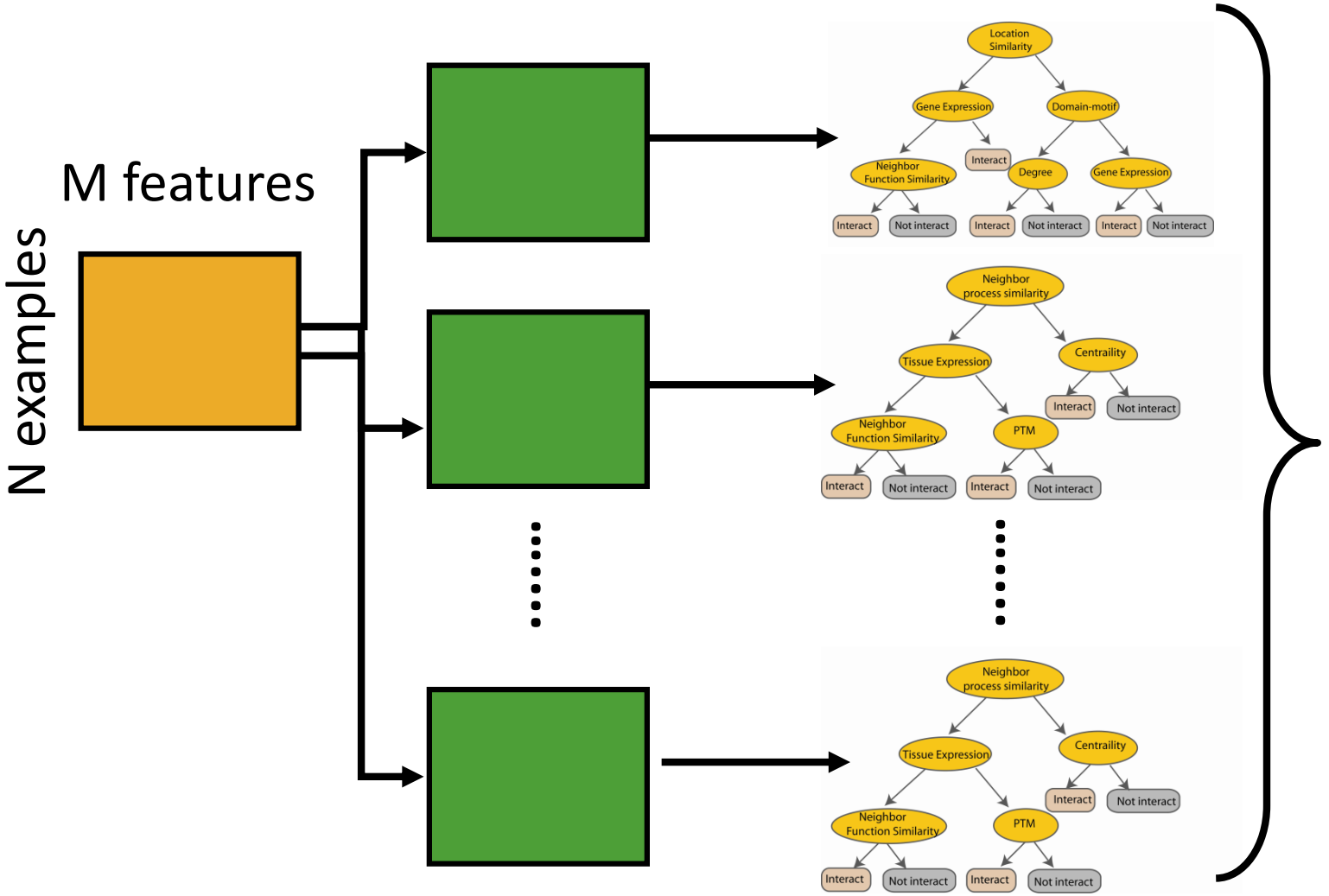
Random Forest Classifier



Random Forest Classifier



Random Forest Classifier



How RF's Work: Version 3 of 4

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

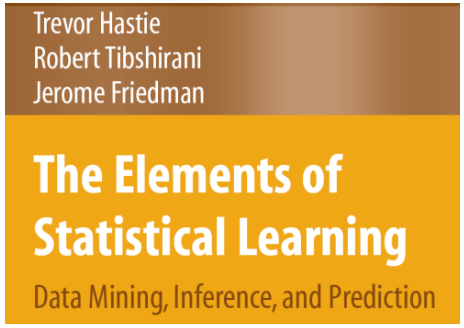
      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.



Random Forests in R

randomForest {randomForest} R Documentation

Classification and Regression with Random Forest

Description

`randomForest` implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points.

Usage

```
## S3 method for class 'formula':
randomForest(formula, data=NULL, ..., subset, na.action=na.fail)
## Default S3 method:
randomForest(x, y=NULL,  xtest=NULL, ytest=NULL, ntree=500,
             mtry=if (!is.null(y) && !is.factor(y))
             max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
             replace=TRUE, classwt=NULL, cutoff, strata,
             sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x)),
             nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
             maxnodes = NULL,
             importance=FALSE, localImp=FALSE, nPerm=1,
             proximity, oob.prox=proximity,
             norm.votes=TRUE, do.trace=FALSE,
             keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,
             keep.inbag=FALSE, ...)
```

mtry number of features selected at each node

Value

An object of class `randomForest`, which is a list with the following components:

`call`	the original call to `randomForest`
`type`	one of `regression`, `classification`, or `unsupervised`.
`predicted`	the predicted values of the input data based on out-of-bag samples.
`importance`	a matrix with `nclass` + 2 (for classification) or two (for regression) columns. For classification, the first `nclass` columns are the class-specific measures computed as mean descrease in accuracy. The `nclass` + 1st column is the mean descrease in accuracy over all classes. The last column is the mean decrease in Gini index. For Regression, the first column is the mean decrease in accuracy and the second the mean decrease in MSE. If `importance=FALSE`, the last measure is still returned as a vector.
`importanceSD`	The "standard errors" of the permutation-based importance measure. For classification, a `p` by `nclass + 1` matrix corresponding to the first `nclass + 1` columns of the importance matrix. For regression, a length `p` vector.
`localImp`	a p by n matrix containing the casewise importance measures, the [i,j] element of which is the importance of i-th variable on the j-th case. `NULL` if `localImp=FALSE`.
`ntree`	number of trees grown.
`mtry`	number of predictors sampled for spliting at each node.
`forest`	(a list that contains the entire forest; `NULL` if `randomForest` is run in unsupervised mode or if `keep.forest=FALSE`.
`err.rate`	(classification only) vector error rates of the prediction on the input data, the i-th element being the error rate for all trees up to the i-th.
`confusion`	(classification only) the confusion matrix of the prediction.
`votes`	(classification only) a matrix with one row for each input data point and one column for each class, giving the fraction or number of 'votes' from the random forest.
`oob.times`	number of times cases are 'out-of-bag' (and thus used in computing OOB error estimate)
`proximity`	if `proximity=TRUE` when `randomForest` is called, a matrix of proximity measures among the input (based on the frequency that pairs of data points are in the same terminal nodes).
`mse`	(regression only) vector of mean square errors: sum of squared residuals divided by `n`.
`rsq`	(regression only) "pseudo R-squared": 1 - `mse` / Var(y).
`test`	if test set is given (through the `xtest` or additionally `ytest` arguments), this component is a list which contains the corresponding `predicted`, `err.rate`, `confusion`, `votes` (for classification) or `predicted`, `mse` and `rsq` (for regression) for the test set. If `proximity=TRUE`, there is also a component, `proximity`, which contains the proximity among the test set as well as proximity between test and training data.

Gradient boosting DT

- Gradient boosting develops an ensemble of tree-based models by training each of the trees in the ensemble on different labels (loss) and then combining the trees.
- Sequential model
- For a regression problem where the objective is to minimize MSE, each successive tree is trained on the errors left over by the collection of earlier trees.

$Target_0$ $Target_1$

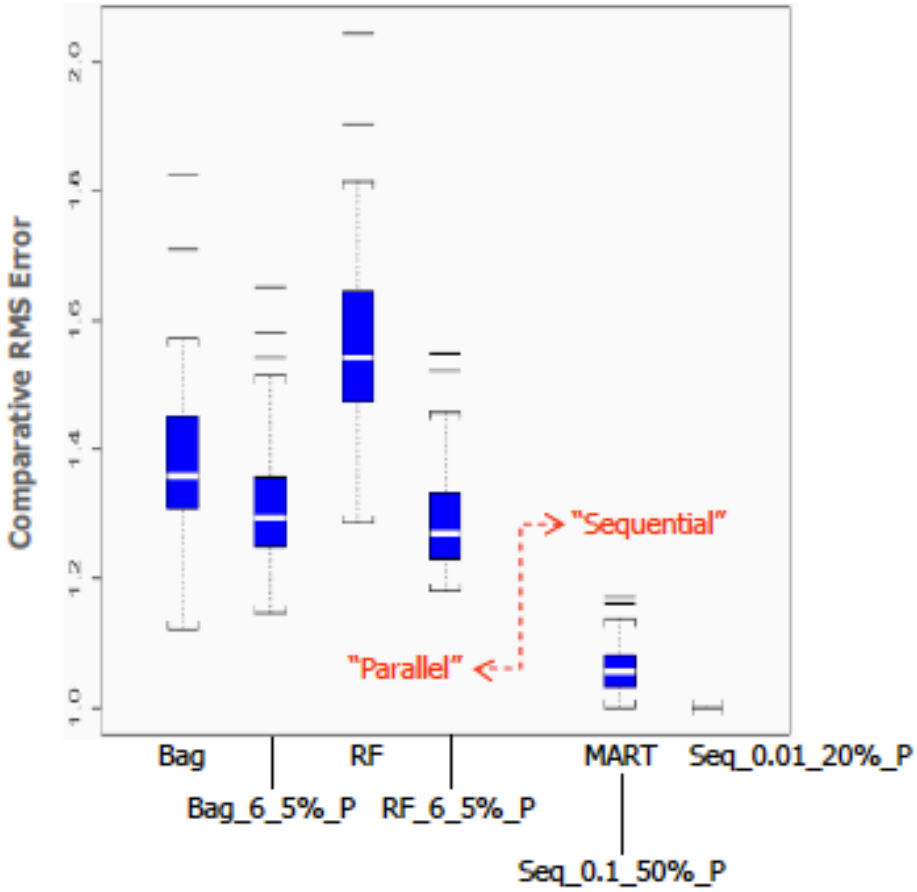
index	X1	X2	X3....	XP	Conversion	logOdds(Conversion)	CR of Train	Baseline initF=logit(CR)	Error Target_iter1	Pred_iter1
	1				1	18	0.01	-0.69	18 - -0.69	
	2				0	-17	0.01		-0.01	
							0.01			
							0.01			
							0.01			
							0.01			
							0.01			
N					1		0.01		1 - 0.01	

log(0.99999999/0.00000001)

> log((1/3)/(2/3))
[1] -0.6931472
>

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]

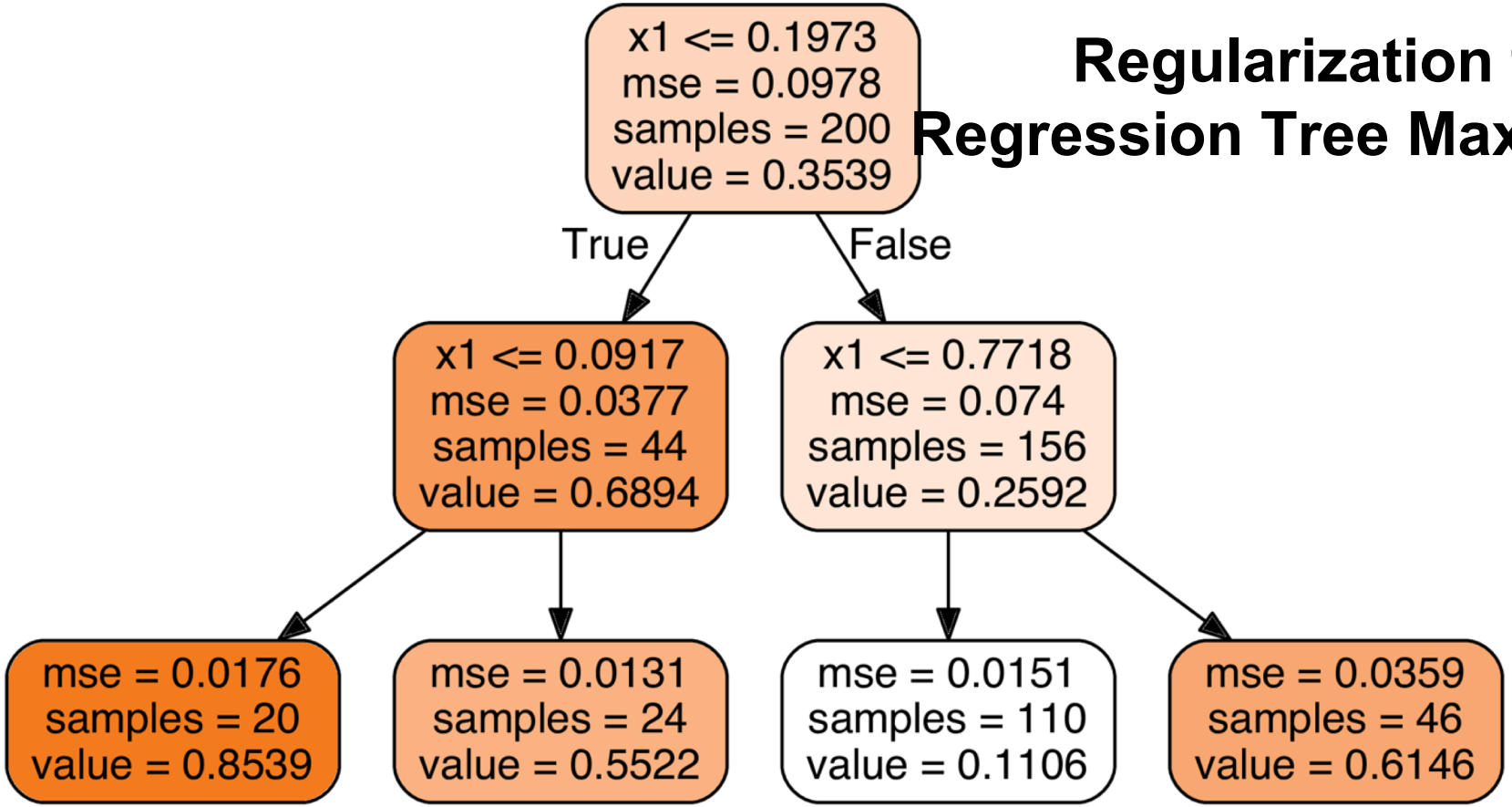
Live Session Outline

- **Introduction**
- **Gradient descent versus non-gradient descent approaches**
- **Decision trees (DT)**
- **Learning decision trees**
  - Classification DTs on discrete variables (X, Y)
  - Regression DTs over continuous variables (X,y)
- **Ensembles of decision Trees**
- **Practical decision trees**
- **Summary**

DTs are kinda non-parametric

- **Decision Trees make very few assumptions about the training data (as opposed to linear models, which obviously assume that the data is linear, for example).**
- **Prone to overfit (kinda non-parametric)**
  - If left unconstrained, the tree structure will adapt itself to the training data, fitting it very closely, and most likely overfitting it.
  - Such a model is often called a nonparametric model, not because it does not have any parameters (it often has a lot) but because the number of parameters is not determined prior to training, so the model structure is free to stick closely to the data.
  - In contrast, a parametric model such as a linear model has a predetermined number of parameters, so its degree of freedom is limited, reducing the risk of overfitting (but increasing the risk of underfitting).

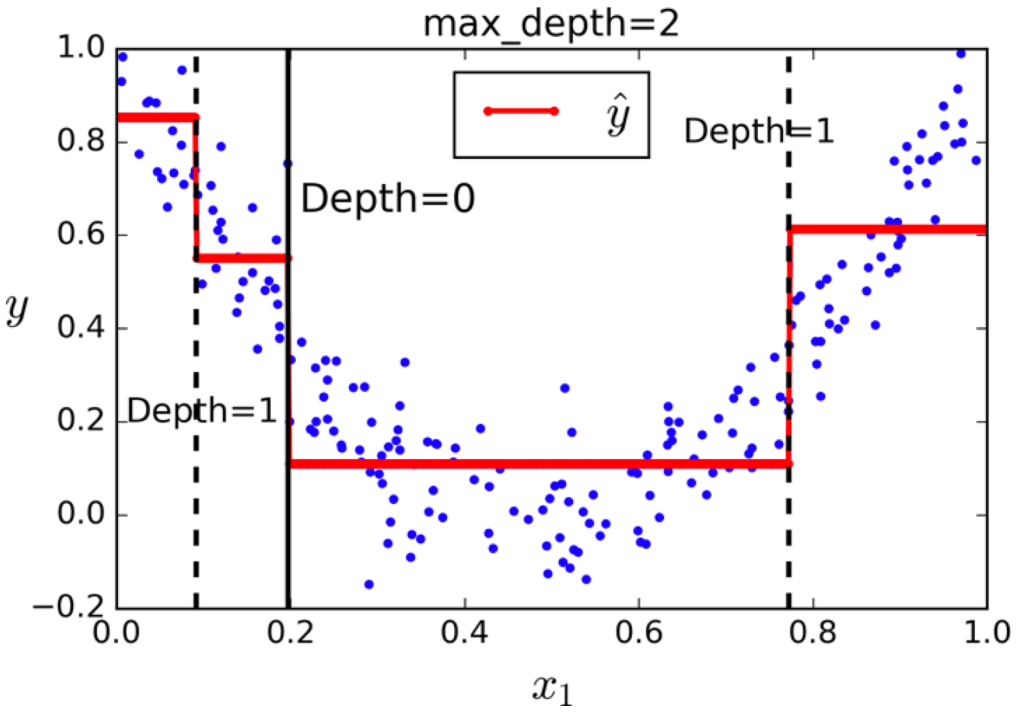
Regularization thru depth
Regression Tree Max depth = 2

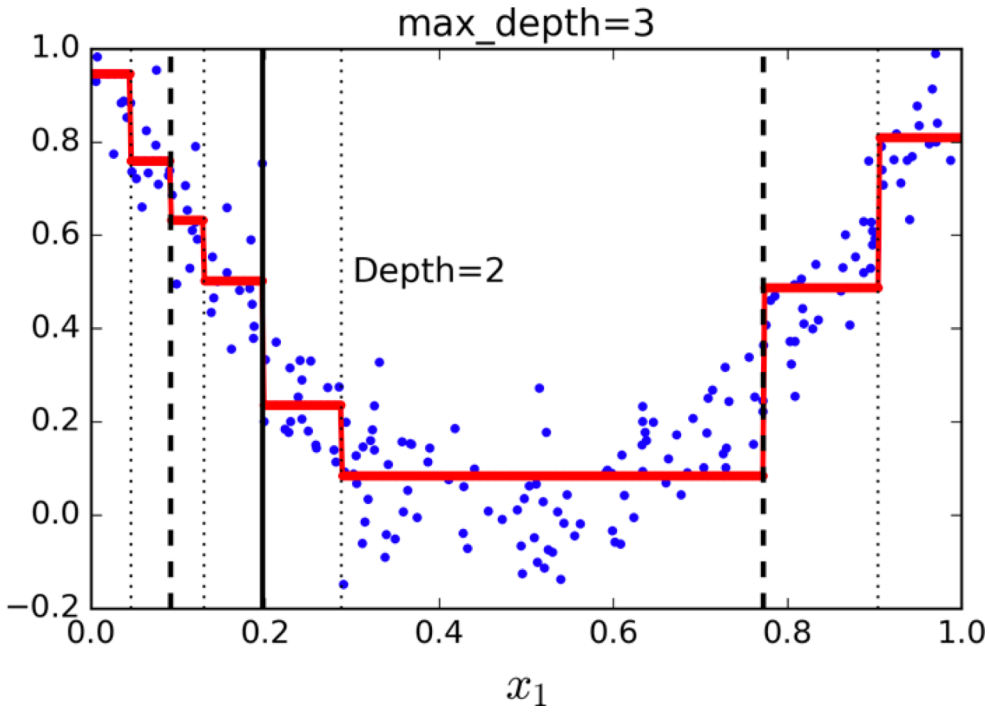


Decision Trees are also capable of performing regression tasks. Let's build a regression tree using Scikit-Learn's `DecisionTreeRegressor` class, training it on a noisy quadratic dataset with `max_depth=2`:

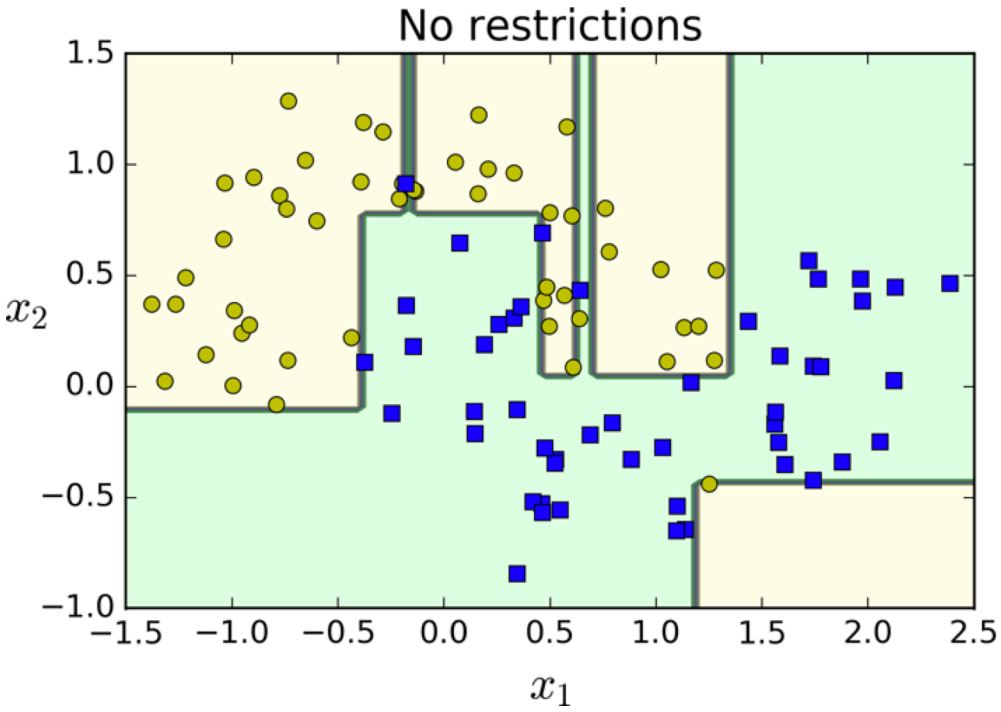
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2)
tree_reg.fit(X, y)
```

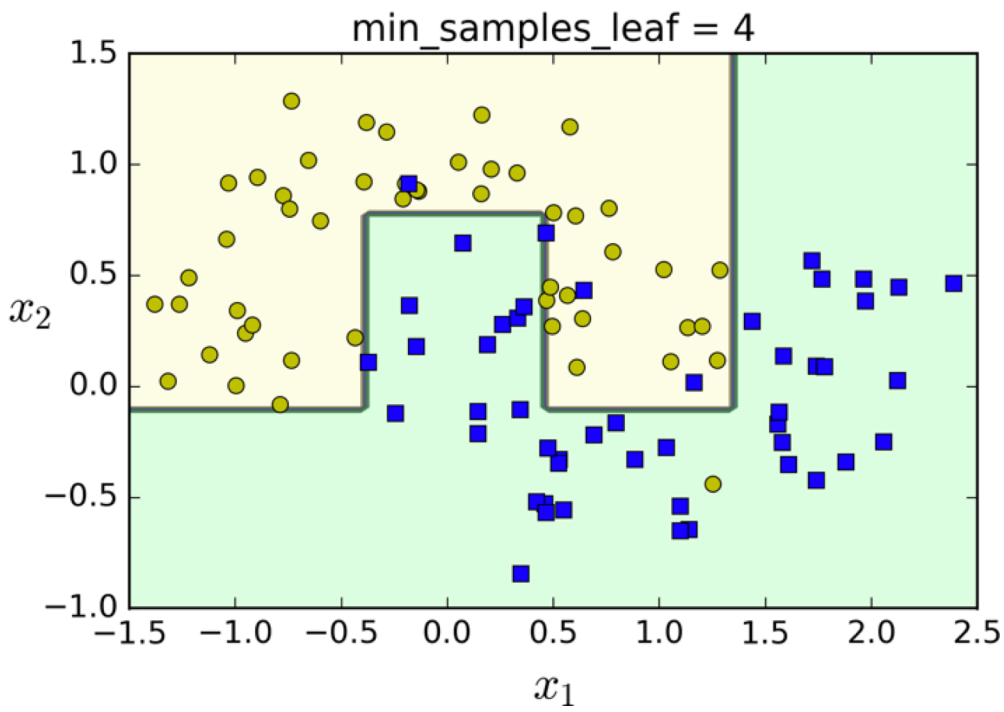
Regularization thru tree depth



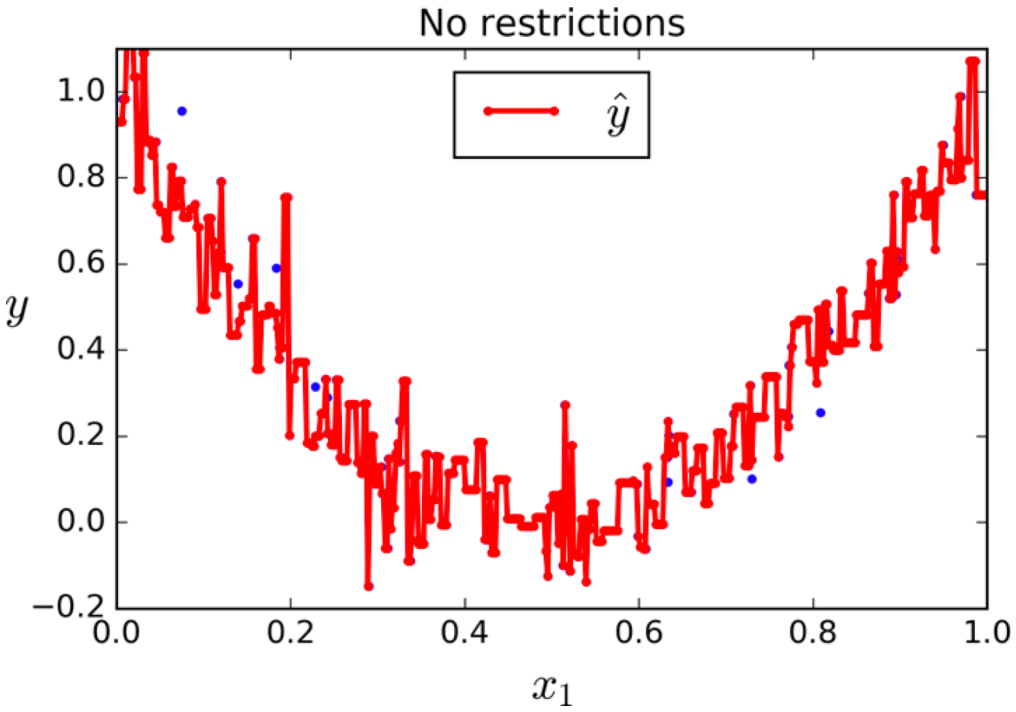


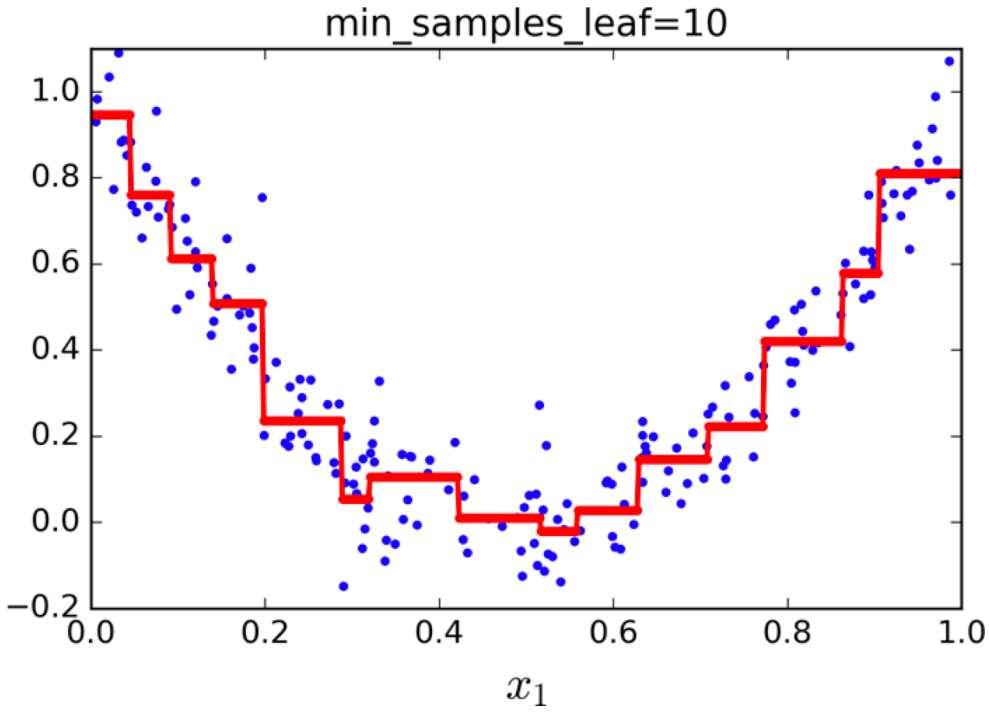
Classification Tree Regularization: min samples





Regression Tree Regularization





Packages for doing ensembles

- xgboost
- gbm

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.

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:
    - See Planet paper from 2008 [Google]

Live Session Outline

- **Introduction**
- **Gradient descent versus non-gradient descent approaches**
- **Decision trees (DT)**
- **Learning decision trees**
  - Classification DTs on discrete variables (X, Y)
  - Regression DTs over continuous variables (X,y)
- **Ensembles of decision Trees**
- **Practical decision trees**
- **Summary**

End of lecture