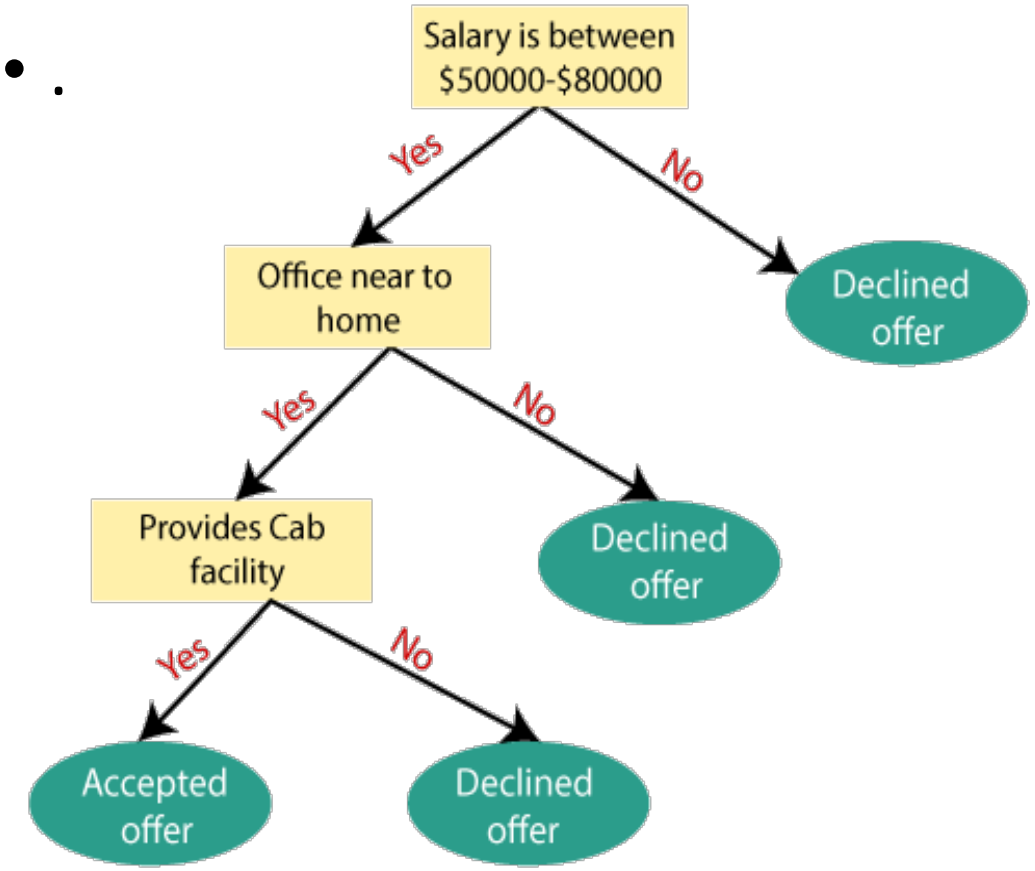
Learning using Decision Trees

Introduction to Machine Learning: Decision Trees

Decision Trees

- .



- A Decision Tree (DT) defines a hierarchy of rules to make a prediction.
- Root and internal nodes test rules. Leaf nodes make predictions.
- Decision Tree (DT) learning is about learning such a tree from labeled data

A decision tree friendly problem

Loan approval prediction

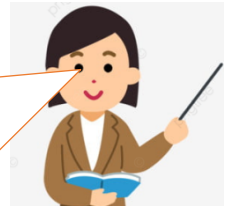
ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

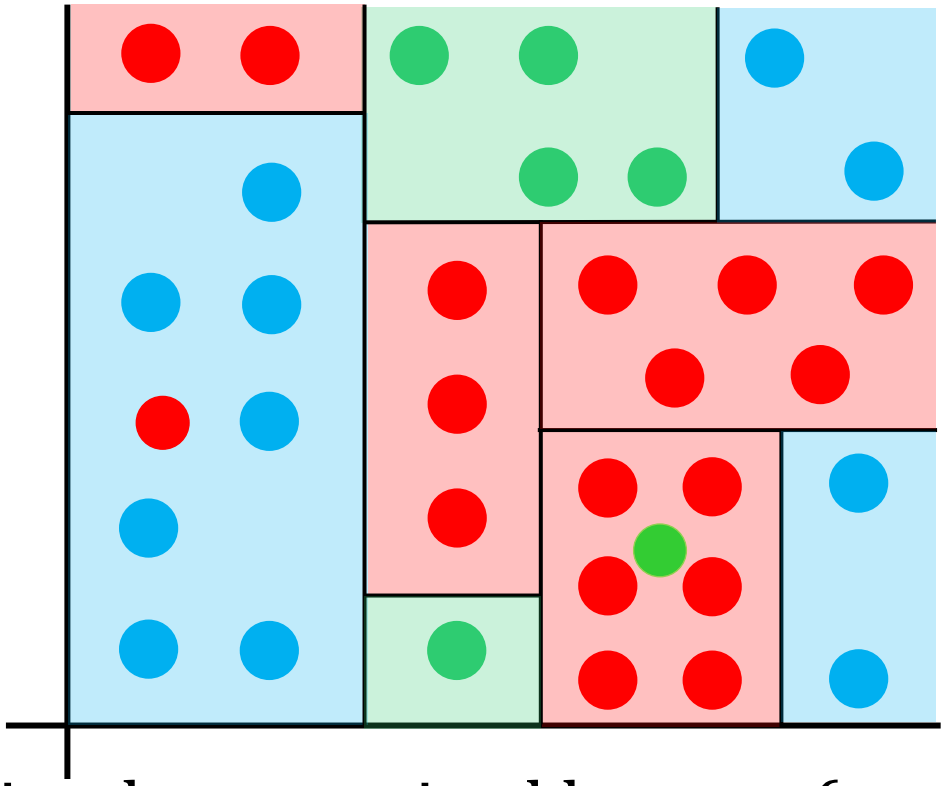
Learning Decision Trees with Supervision

- The basic idea is very simple
- Recursively partition the training data into homogeneous regions

What do you mean by "homogeneous" regions?

A homogeneous region will have all (or a majority of) training inputs with the same/similar outputs

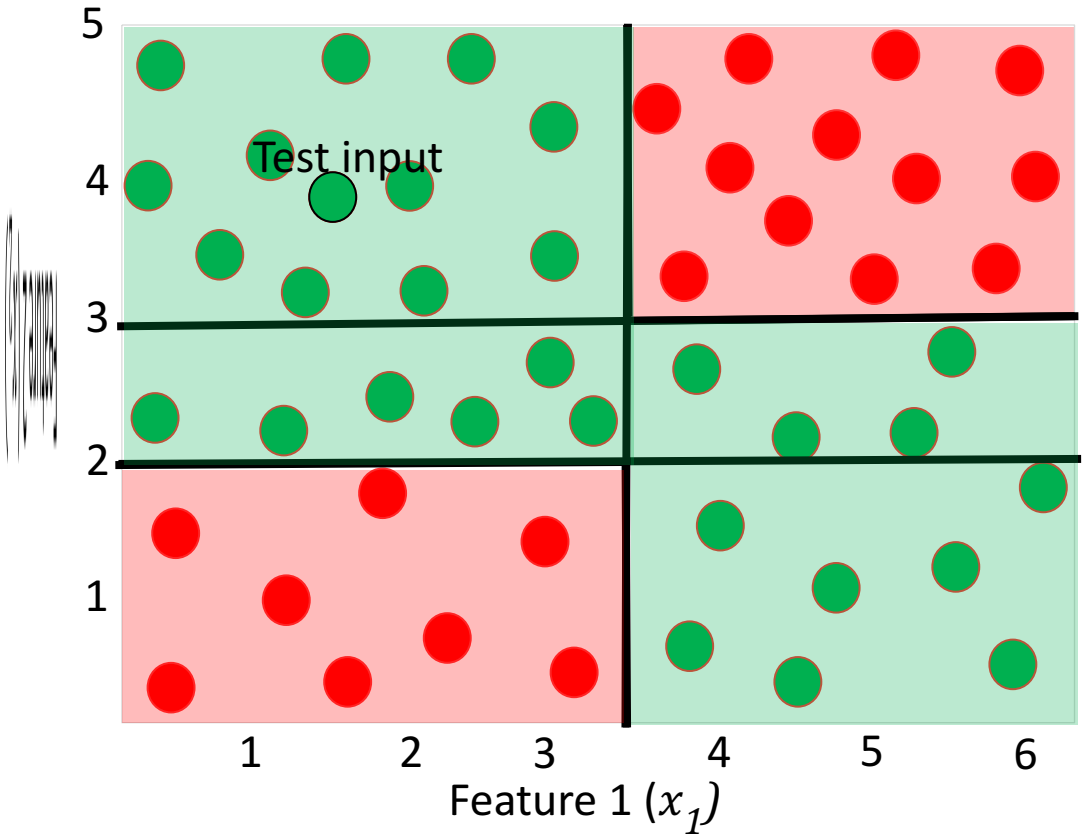


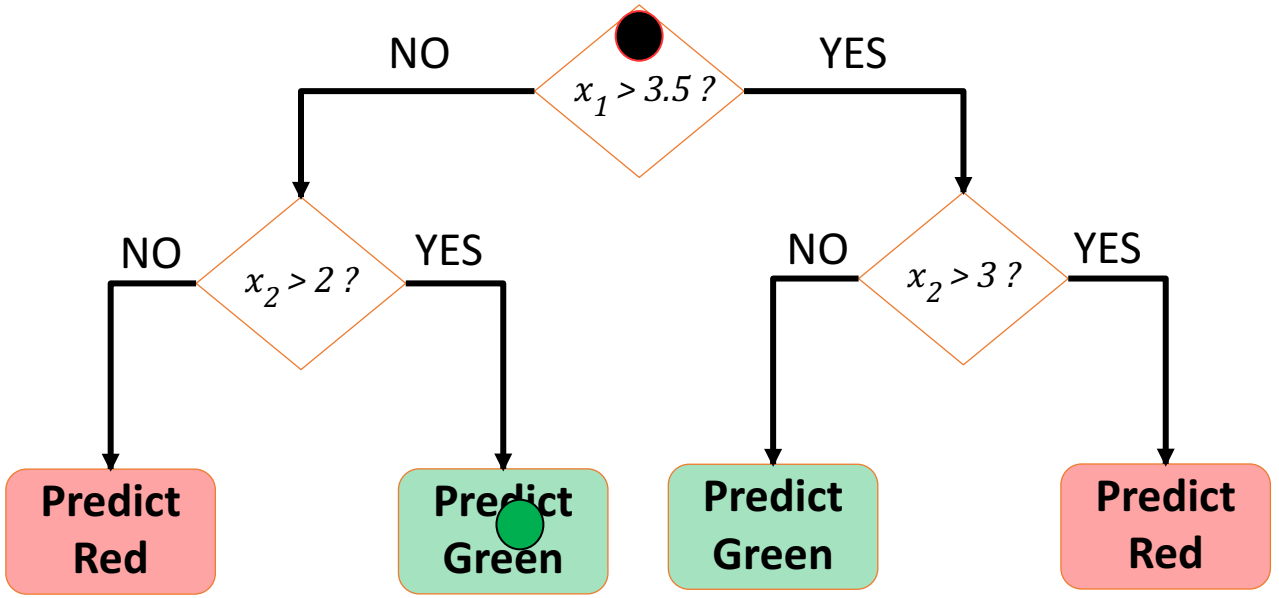


Even though the rule within each group is simple, we are able to learn a fairly sophisticated model overall (note in this example, each rule is a simple horizontal/vertical classifier but the overall decision boundary is rather sophisticated)

- Within each group, fit a simple supervised learner (e.g., predict the majority label)

Decision Trees for Classification





Feature 1 (x_1)

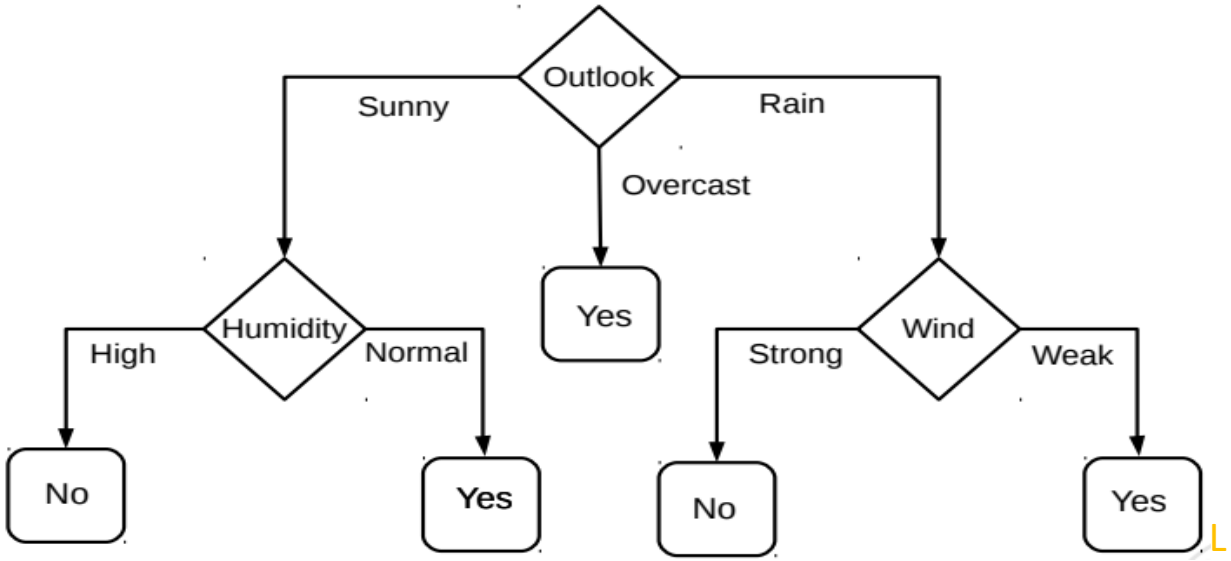
DT is very efficient at test time: To predict the label of a test point, nearest neighbors will require computing distances from 48 training inputs. DT predicts the label by doing just 2 feature-value comparisons! Way faster!

Remember: Root node contains all training inputs
Each leaf node receives a subset of training inputs

Decision Trees for Classification: Another Example

- Deciding whether to play or not to play Tennis on a Saturday
- Each input (Saturday) has 4 categorical features: Outlook, Temp., Humidity, Wind
- A binary classification problem (play vs no-play)
- Below Left: Training data, Below Right: A decision tree constructed using this data

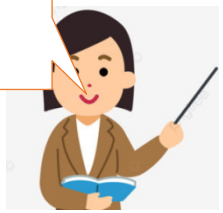
day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



Example credit: Tom Mitchell

Decision Trees: Some Considerations

Usually, cross-validation can be used to decide size/shape



- What should be the size/shape of the DT?
  - Number of internal and leaf nodes
  - Branching factor of internal nodes
  - Depth of the tree

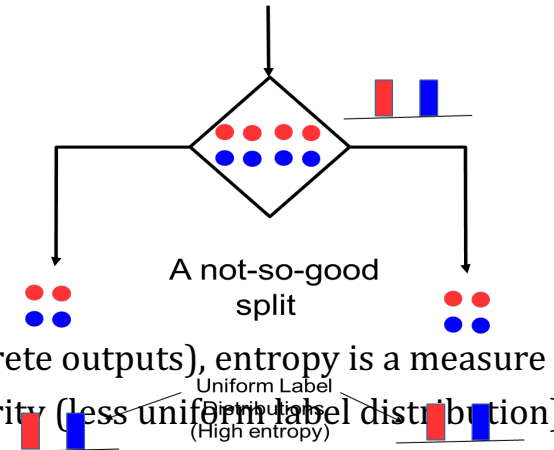
- Split criterion at internal nodes
  - Use another classifier?
  - Or maybe by doing a simpler test?

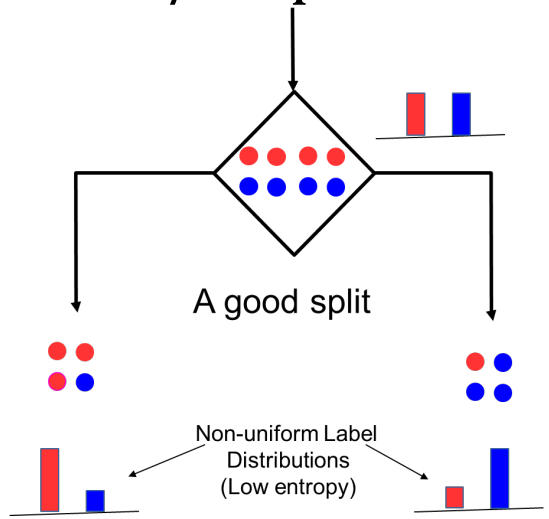
- What to do at the leaf node? Some options:
  - Make a constant prediction for each test input reaching there
  - Use a nearest neighbor based prediction using training inputs at that leaf node
  - Train and predict using some other sophisticated supervised learner on that node

Usually, constant prediction at leaf nodes used since it will be very fast

How to Split at Internal Nodes?

- Recall that each internal node receives a subset of all the training inputs
- Regardless of the criterion, the split should result in as "pure" groups as possible
  - A pure group means that the majority of the inputs have the same label/output





For classification problems (discrete outputs), entropy is a measure of purity

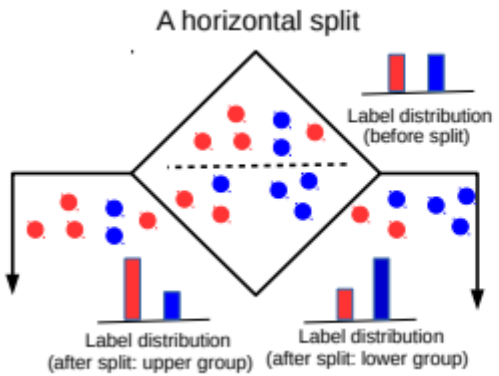
- Low entropy ⇒ high purity (less uniform label distribution)
- Splits that give the largest reduction (before split vs after split) in entropy are preferred (this reduction is also known as "information gain")

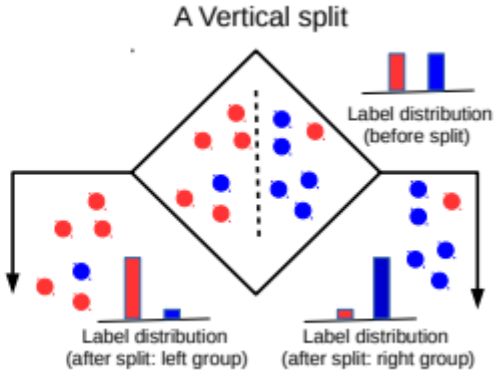
Techniques to Split at Internal Nodes

- Each internal node decides which outgoing branch an input should be sent to
- This decision/split can be done using various ways, e.g.,
  - Testing the value of a single feature at a time (such internal node called "Decision Stump")

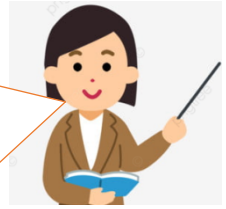
With this approach, all features and all possible values of each feature need to be evaluated in selecting the feature to be tested at each internal node
(can be slow but can be made faster using some tricks)

re

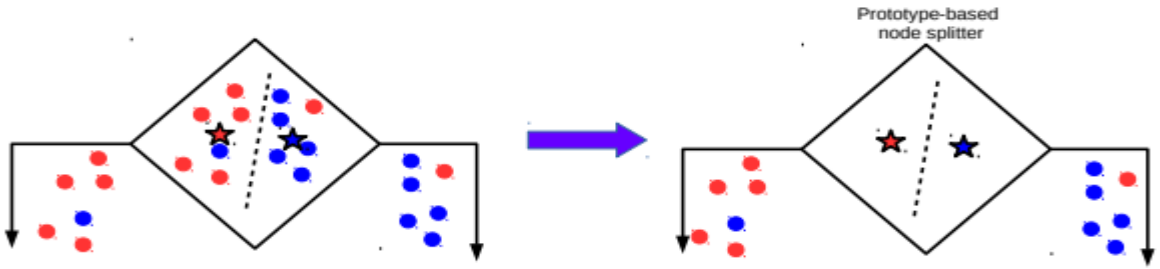




DT methods based on testing a single feature at each internal node are faster and more popular (e.g., ID3, C4.5 algos)

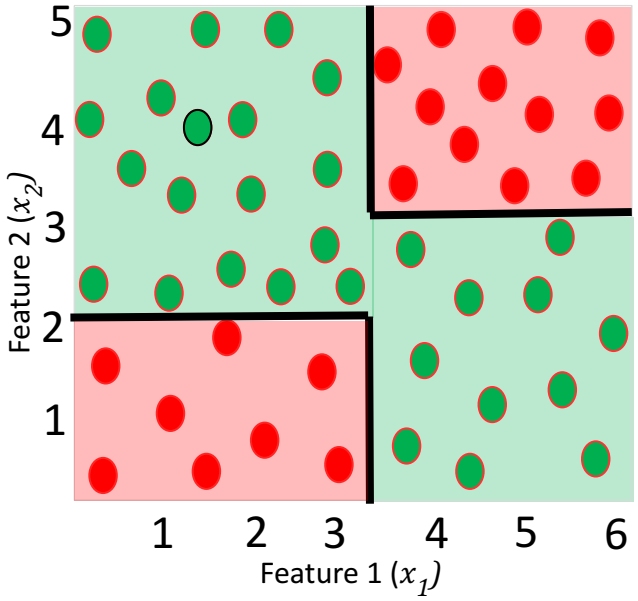


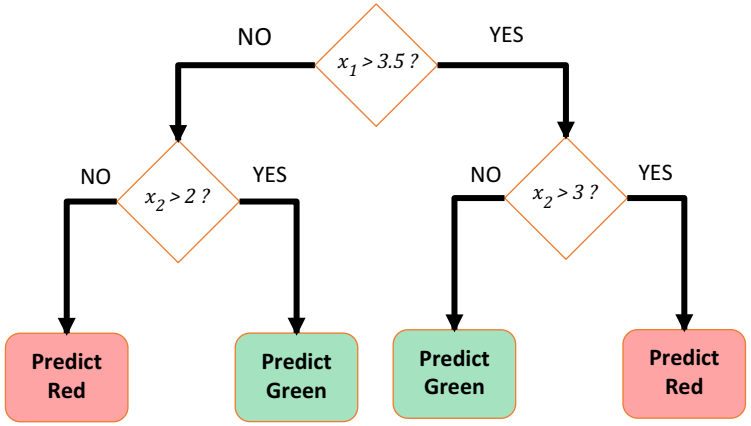
ıor



DT methods based on learning and using a separate classifier at each internal node are less common. But this approach can be very powerful and are sometimes used in some advanced DT methods

Constructing Decision Trees



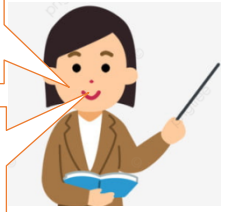


Given some training data, what's the "optimal" DT?

How to decide which rules to test for and in what order?

How to assess informativeness of a rule?

In general, constructing DT is an intractable problem (NP-hard)



Often we can use some "greedy" heuristics to construct a "good" DT

The rules are organized in the DT such that most informative rules are tested first

To do so, we use the training data to figure out which rules should be tested at each node

Hmm.. So DTs are like the "20 questions" game (ask the most useful questions first)

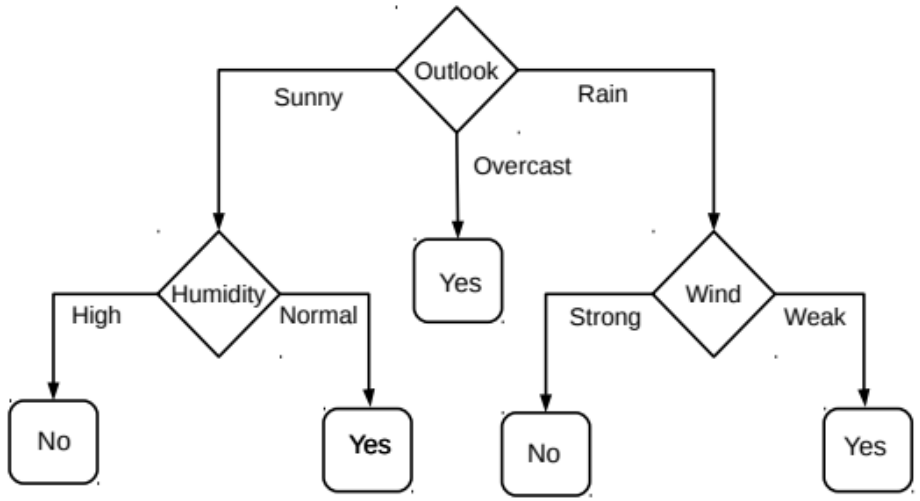
Informativeness of a rule is of related to the extent of the purity of the split arising due to that rule. More informative rules yield more pure splits

The same rules will be applied on the test inputs to route them along the tree until they reach some leaf node where the prediction is made

Decision Tree Construction: An Example

- Let's consider the playing Tennis example
- Assume each internal node will test the value of one of the features

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Question: Why does it make more sense to test the feature "outlook" first?
- Answer: Of all the 4 features, it's the most informative
  - It has the highest information gain as the root node

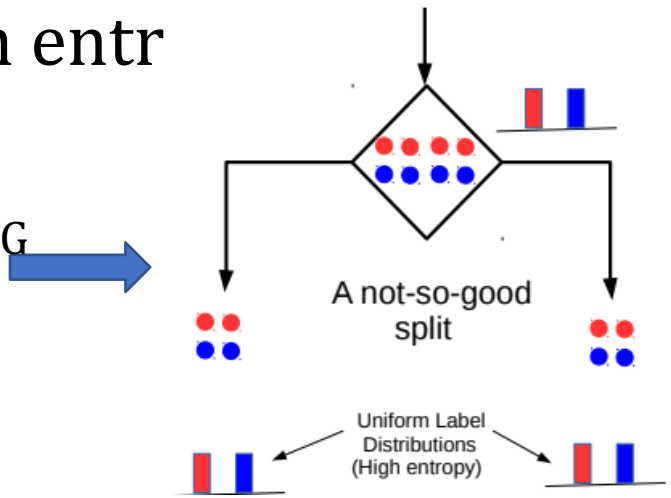
Entropy and Information Gain

- Assume a set of labelled inputs $\boldsymbol{S}$ **from** $\boldsymbol{C}$ classes, p_c as fraction of class c inputs
- **Entropy** of the set $\boldsymbol{S}$ is defined as $H(\boldsymbol{S}) = -\sum_{c \in C} p_c \log p_c$
- Suppose a rule splits $\boldsymbol{S}$ into two smaller disjoint sets $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$
- Reduction in entr

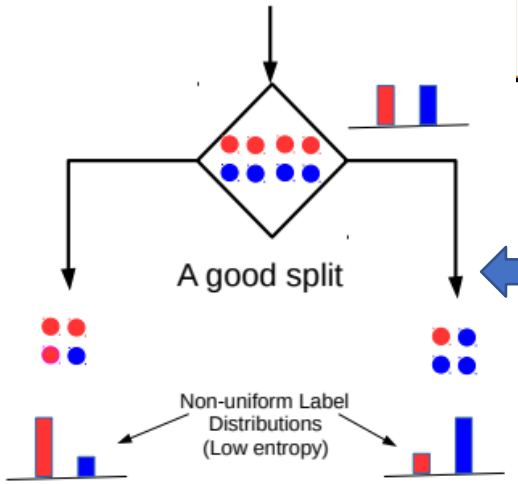
$$IG = H(S) - \frac{|S_1|}{|S|} H(S_1) - \frac{|S_2|}{|S|} H(S_2)$$

Uniform sets (all classes roughly equally present) have high entropy; skewed sets low

This split has a low IG (in fact zero IG)



VS



This split has higher IG

Entropy and Information Gain

- Let's use IG based criterion to construct a DT for the Tennis example
- At root node, let's compute IG of each of the 4 features
- Consider feature "wind". Root contains all examples **S** = [9+,5

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

$$H(S) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.94$$

$$\boldsymbol{S}_{\text{weak}} = [6+, 2-] \Rightarrow \text{H}(\boldsymbol{S}_{\text{weak}}) = 0.811$$

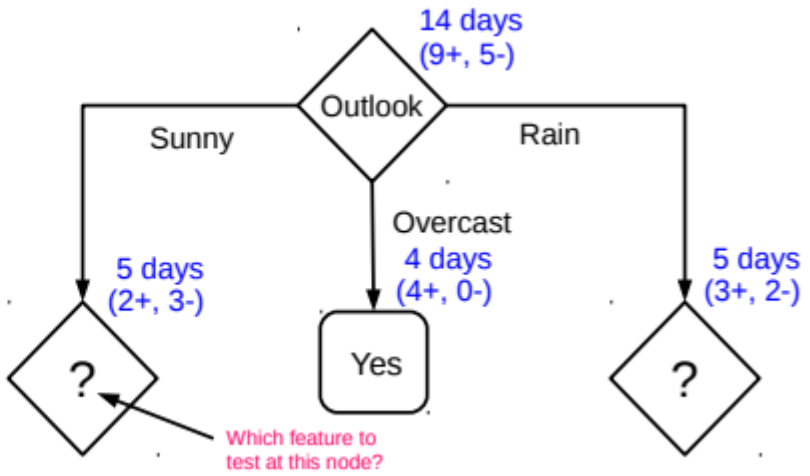
$$\boldsymbol{S}_{\text{strong}} = [3+, 3-] \Rightarrow \text{H}(\boldsymbol{S}_{\text{strong}}) = 1$$

$$IG(S,wind) = H(S) - \frac{|S_{\text{weak}}|}{|S|}H(S_{\text{weak}}) - \frac{|S_{\text{strong}}|}{|S|}H(S_{\text{strong}}) = 0.94 - 8/14 * 0.811 - 6/14 * 1 = 0.048$$

- Likewise, at root: IG(S, outlook) = 0.246, IG(S, humidity) = 0.151, IG(S,temp) = 0.029
- Thus we choose "outlook" feature to be tested at the root node
- Now how to grow the DT, i.e., what to do at the next level? Which feature to test next?
- Rule: Iterate - for each child node, select the feature with the highest IG

Growing the tree

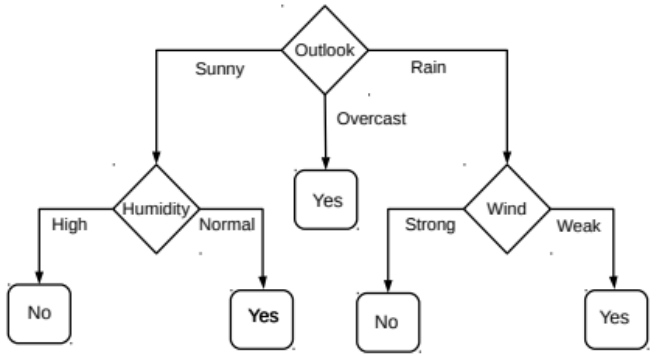
day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Proceeding as before, for level 2, left node, we can verify that
  - IG(S,temp) = 0.570, IG(S, humidity) = 0.970, IG(S, wind) = 0.019
- Thus humidity chosen as the feature to be tested at level 2, left node
- No need to expand the middle node (already "pure" - all "yes" training examples)
- Can also verify that wind has the largest IG for the right node
- Note: If a feature has already been tested along a path earlier, we don't consider it again

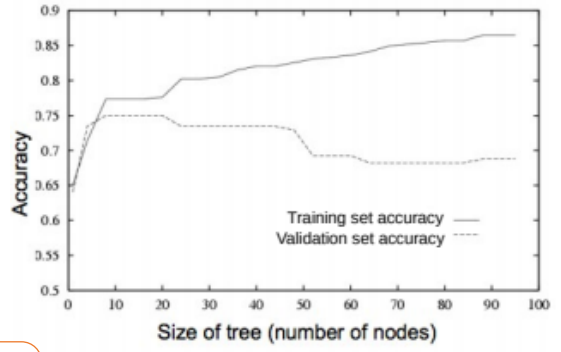
When to stop growing the tree?

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

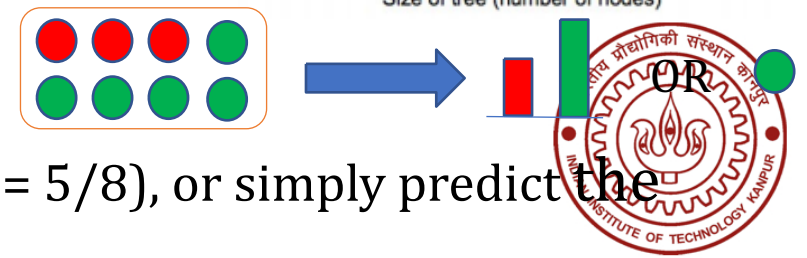


- Stop expanding a node further (i.e., make it a leaf node) when
  - It consist of all training examples having the same label (the node becomes "pure")
  - We run out of features to test along the path to that node
  - The DT starts to overfit (can be checked by monitoring the validation set accuracy)

To help prevent the tree from growing too much!



- **Important:** No need to obsess too much for purity
  - It is okay to have a leaf node that is not fully pure, e.g., this
  - At test inputs that reach an impure leaf, can predict probability of belonging to each class (in above example, p(red) = 3/8, p(green) = 5/8), or simply predict the majority label



Avoiding Overfitting in DTs

- Desired: a DT that is not too big in size, yet fits the training data reasonably
- Note: An example of a very simple DT is "decision-stump"
  - A decision-stump only tests the value of a single feature (or a simple rule)
  - Not very powerful in itself but often used in large ensembles of decision stumps
- Mainly two approaches to prune a complex DT
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)
- Criteria for judging which nodes could potentially be pruned
  - Use a validation set (separate from the training set)
    - Prune each possible node that doesn't hurt the accuracy on the validation set
    - Greedily remove the node that improves the validation accuracy the most
    - Stop when the validation set accuracy starts worsening
  - Use model complexity control, such as Minimum Description Length (will see later)

Either can be done using a validation set

Pruning: Getting an Optimal Decision tree

- *Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.*
- A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- Advantages of the Decision Tree
- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.
- Disadvantages of the Decision Tree
- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

Comparison Between ID3 and CART Decision Tree Algorithms

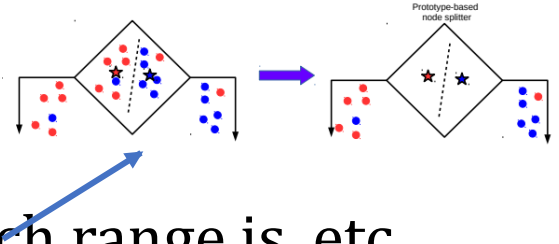
Aspect	ID3 (Iterative Dichotomiser 3)	CART (Classification and Regression Trees)
Full Form	Iterative Dichotomiser 3	Classification and Regression Trees
Purpose	Primarily used for classification tasks.	Used for both classification and regression tasks.
Splitting Criteria	Based on **Information Gain**.	Based on **Gini Index** (classification) or Mean Squared Error (regression).
Output Tree Type	Generates multi-way splits (not binary).	Always generates binary trees (two splits per node).
Handling Continuous Variables	Converts continuous features into discrete ranges before splitting.	Handles continuous variables naturally using thresholds.
Pruning Method	Does not include built-in pruning.	Supports **post-pruning** to prevent overfitting.
Overfitting	More prone to overfitting due to lack of pruning.	Less prone to overfitting because of pruning capabilities.
Ease of Implementation	Simpler algorithm, easier to understand.	More complex due to pruning and binary tree structure.
Computation Efficiency	Relatively faster but may lead to overfitting.	Computationally intensive due to pruning but often results in better models.
Applications	Used in simpler classification problems.	Widely used in complex classification and regression problems.

Decision Trees: Some Comments

- Gini-index defined as $\sum_{c=1}^{C} p_c(1 - p_c)$ can be an alternative to IG

- For DT regression[1], variance in the outputs can be used to assess purity

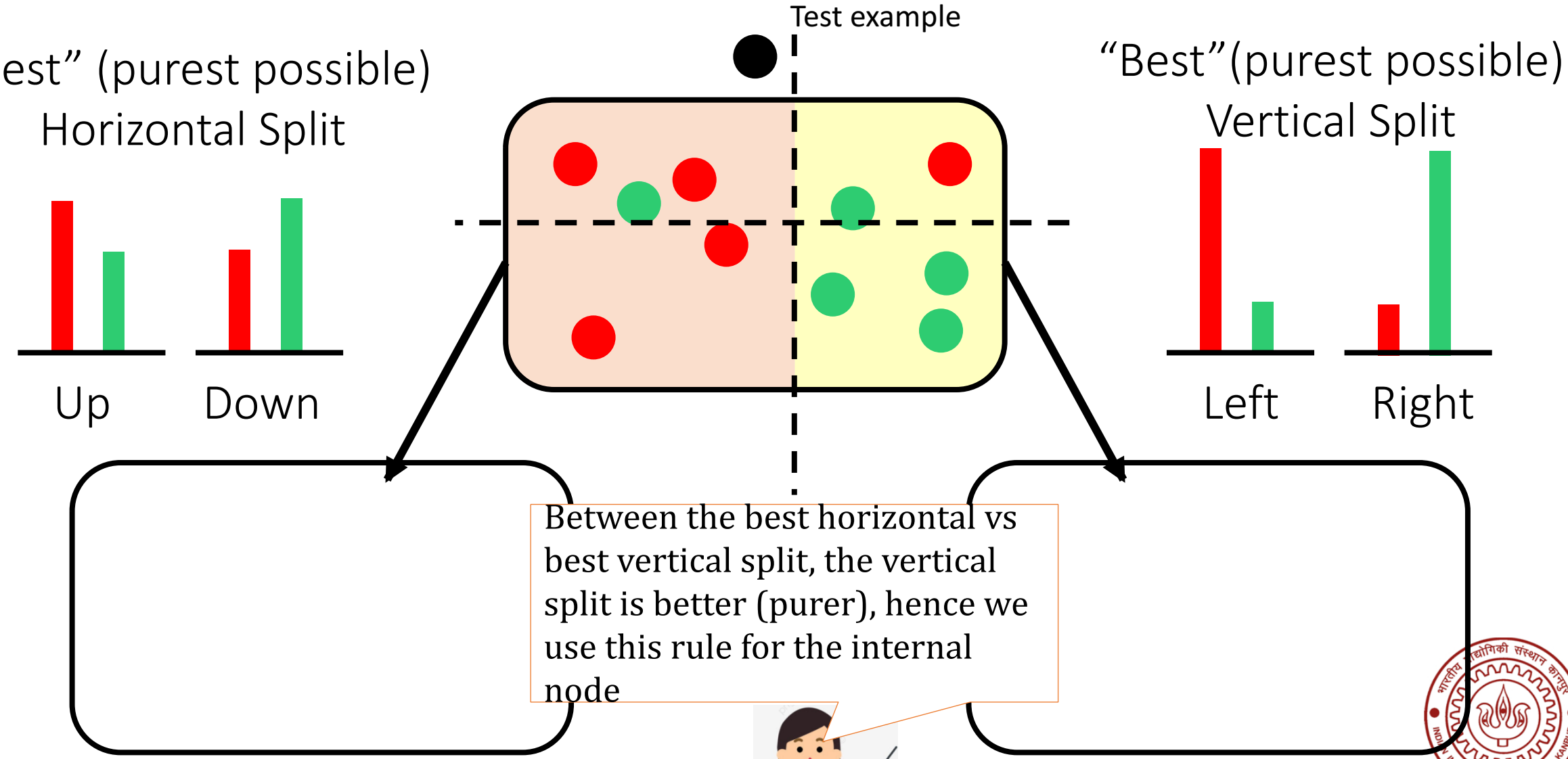
- When features are real-valued (no finite possible values to try), things are a bit more tricky
  - Can use tests based on thresholding feature values (recall our sy examples)
  - Need to be careful w.r.t. number of threshold points, how fine each range is, etc.



- More sophisticated decision rules at the internal nodes can also be used
  - Basically, need some rule that splits inputs at an internal node into homogeneous groups
  - The rule can even be a machine learning classification algo (e.g., LwP or a deep

[1]Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees

An Illustration: DT with Real-Valued Features

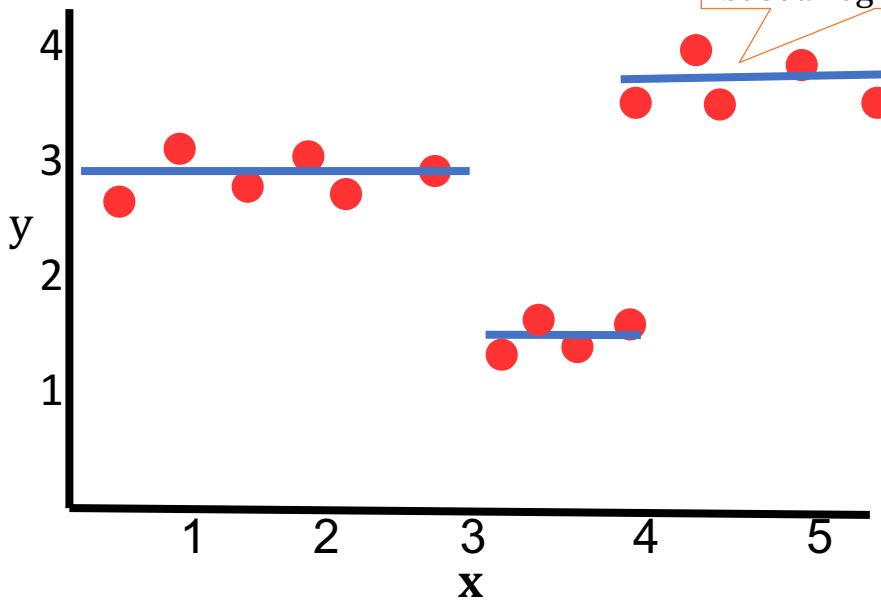


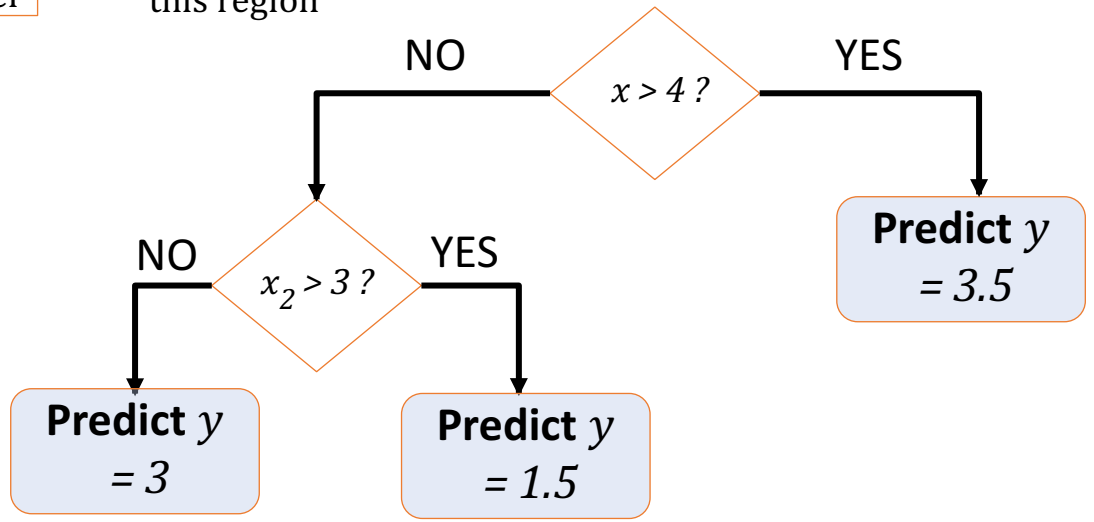
This illustration's credit: Purushottam Kar

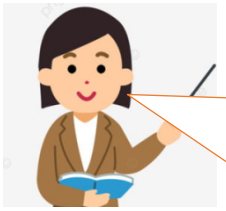
Decision Trees for Regression

Can use any regression model but would like a simple one, so let's use a constant prediction based regression model

Another simple option can be to predict the average output of the training inputs in this region







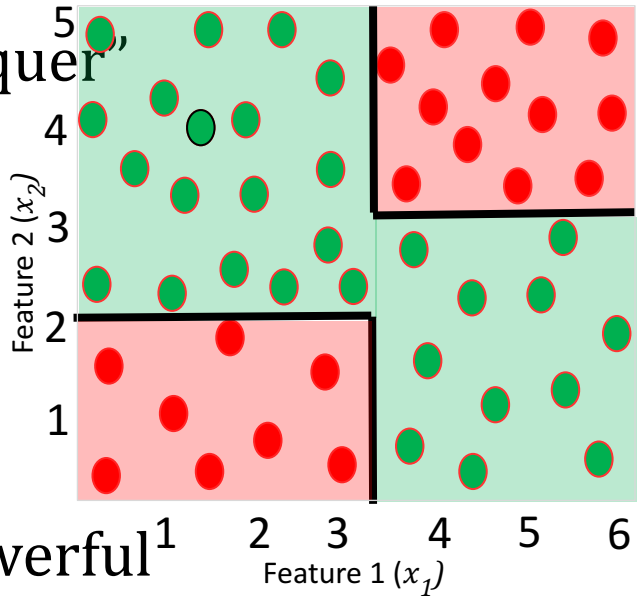
To predict the output for a test point, nearest neighbors will require computing distances from 15 training inputs. DT predicts the label by doing just at most feature-value comparisons! Way faster!

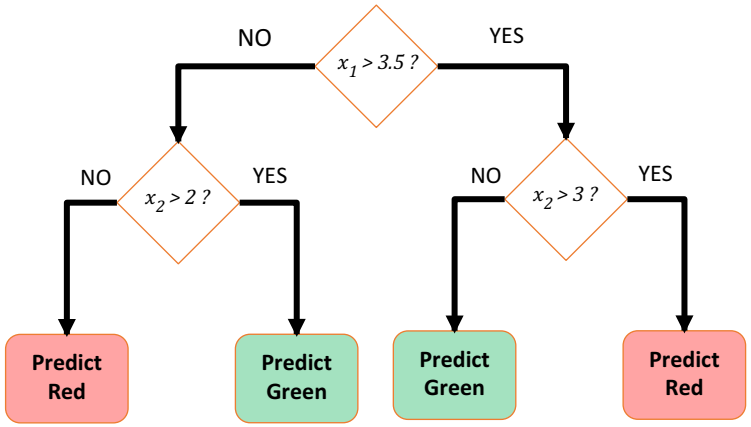
Decision Trees: A Summary

.. thus helping us learn complex rule as a combination of several simpler rules

Some key strengths:

- Simple and easy to interpret
- Nice example of "divide and conquer" paradigm in machine learning
- Easily handle different types of features (real, categorical, etc.)
- Very fast at test time
- Multiple DTs can be combined via ensemble methods: more powerful (e.g., Decision Forests; will see later)





Human-body pose estimation

- Used in several real-world ML applications, e.g., recommender systems, gaming (Kinect)

Some key weaknesses:

- Learning optimal DT is (NP-hard) intractable. Existing algos mostly greedy heuristics