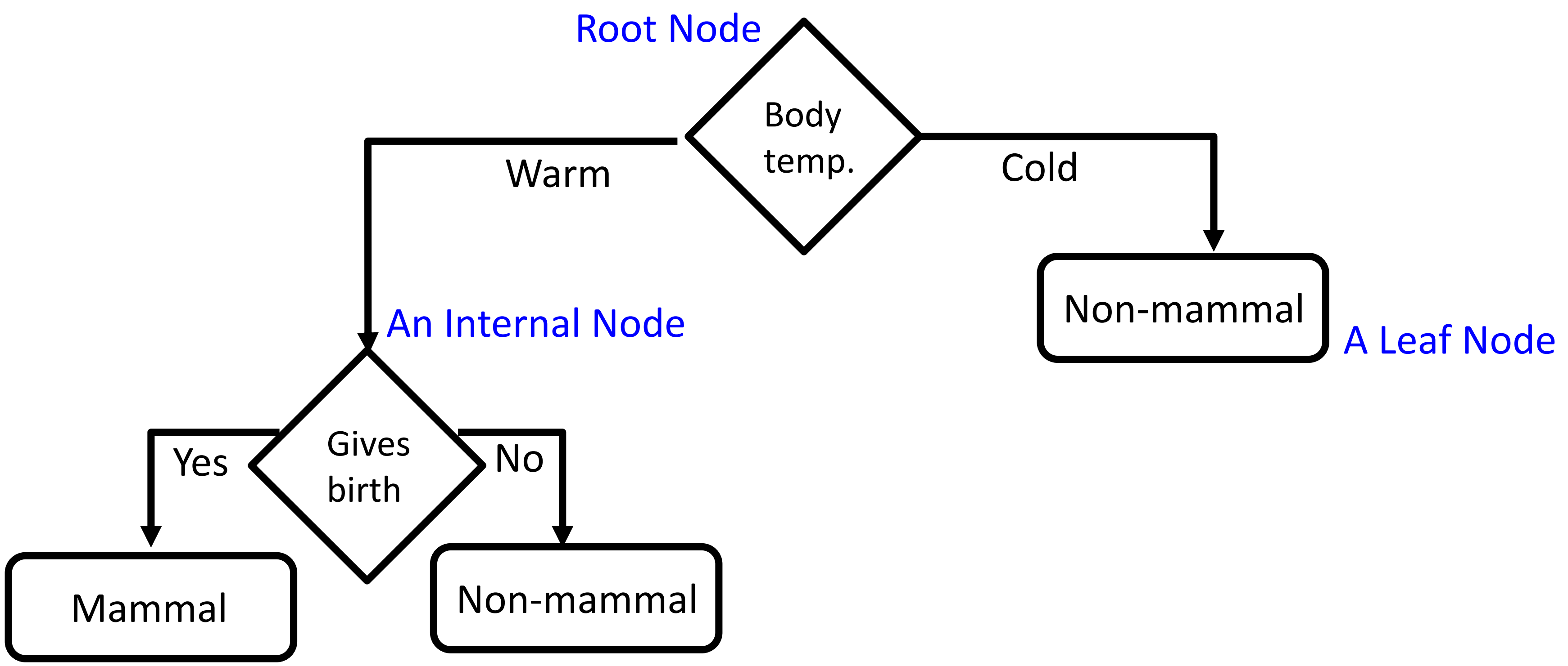
Decision Trees

CS771: Introduction to Machine Learning

Decision Trees

- A Decision Tree (DT) defines a hierarchy of rules to make a prediction



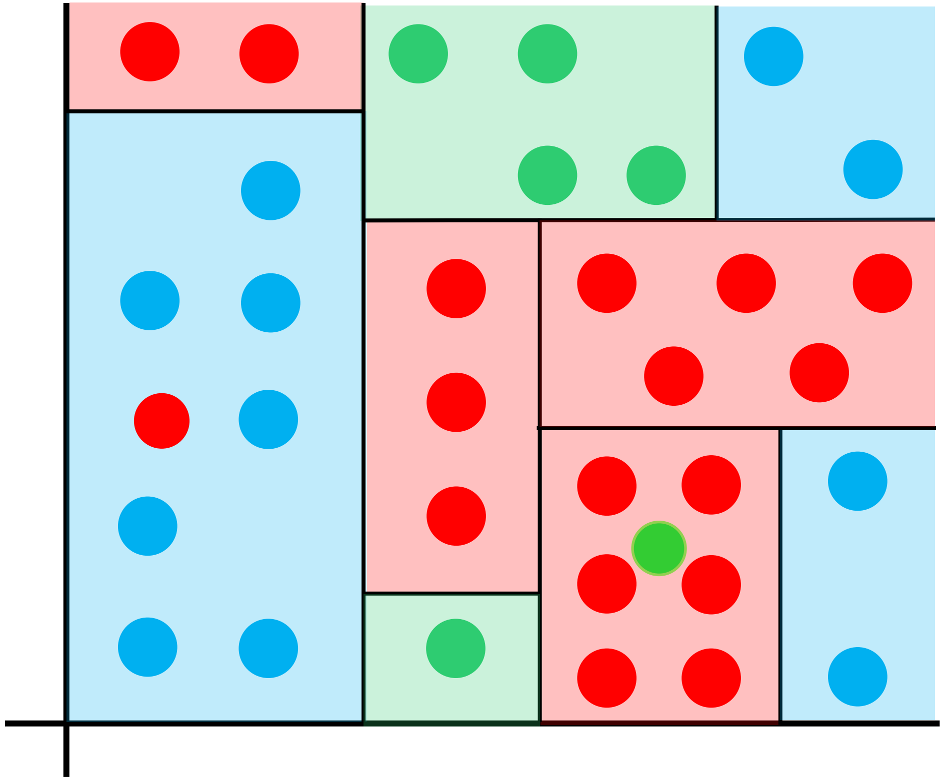
- Root and internal nodes test rules. Leaf nodes make predictions
- DT learning is about learning such a tree from labeled training data

Decision Tree Learning: The Basic Idea

- Recursively partition training data till you get (roughly) homogeneous regions

What do you mean by "homogeneous" regions?

A homogeneous region will have all (or most of) the training inputs with the same outputs/labels



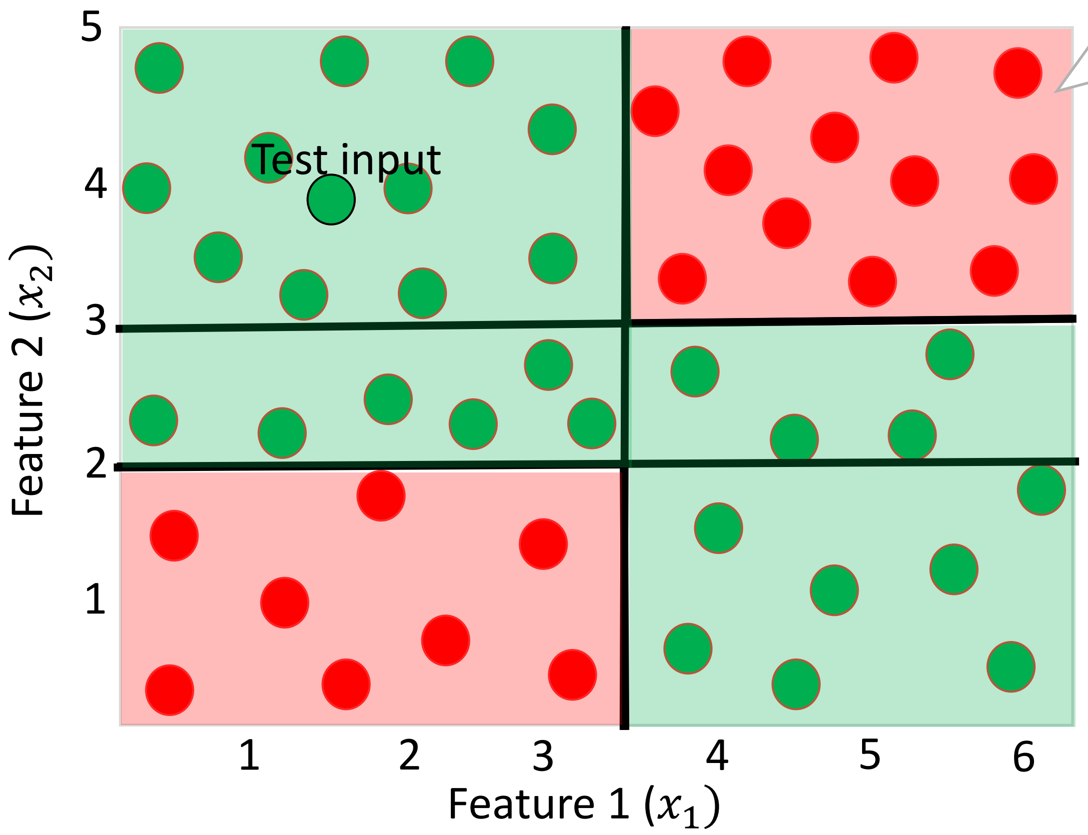
We will use training data to learn a DT that defines the partitioning

Test time: Given a test input, first locate its region using DT. Then use the prediction rule of that region to predict its label

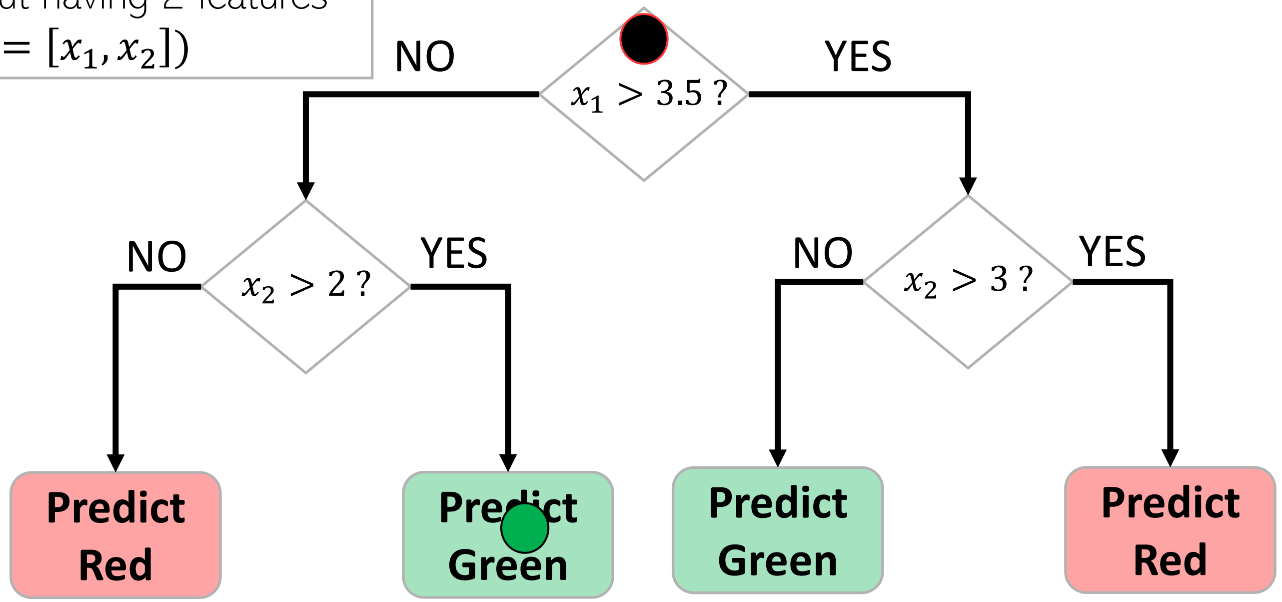
Within each region, we can even use a very sophisticated model (like a deep neural network) but we usually prefer a simple rule (constant label, or maybe use a simple ML model like LwP) so that training and test phases are fast

- Some typical prediction rules for each region
  - Use a constant label (e.g., majority) if region fully/almost homogeneous
  - Learn another prediction model (e.g., LwP) if region not fully homogeneous

Decision Tree for Classification: An Example



Training data with each input having 2 features ($\boldsymbol{x} = [x_1, x_2]$)



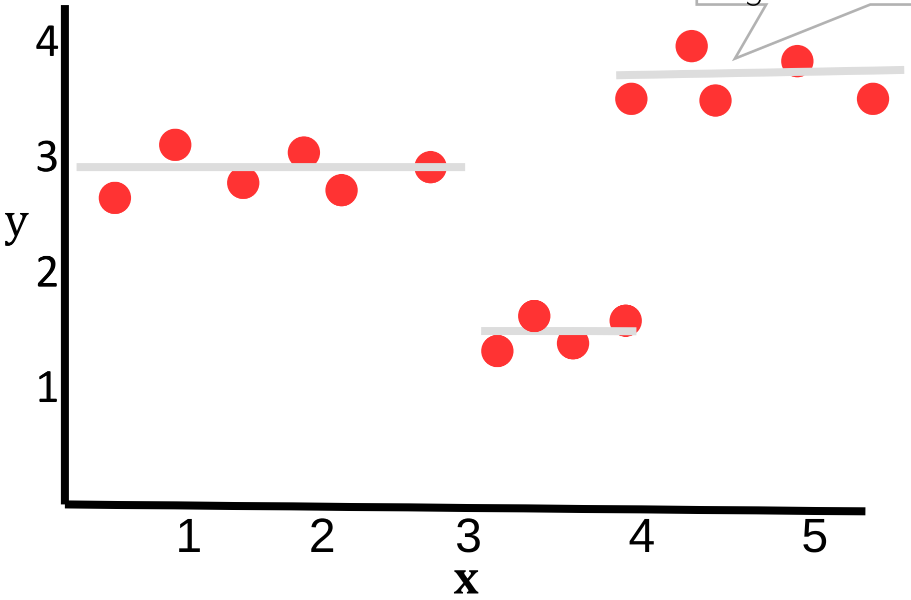
DT is very efficient at test time: To predict the label of a test point, nearest neighbors will require computing distances from 48 training inputs. DT predicts the label by doing just 2 feature-value comparisons! Way more fast!!!

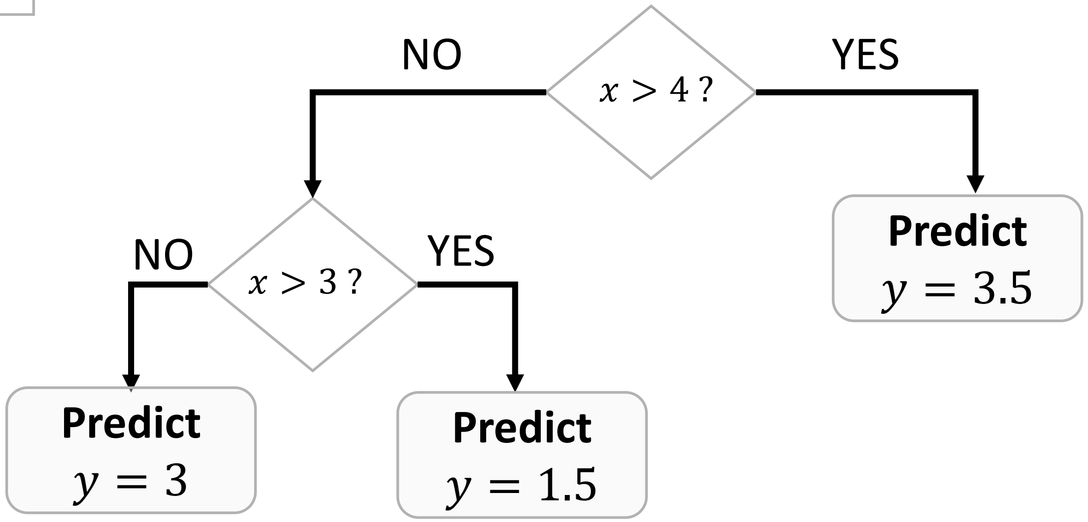
Remember: Root node contains all training inputs. Internal/leaf nodes receive a subset of training inputs

Decision Tree for Regression: An Example

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





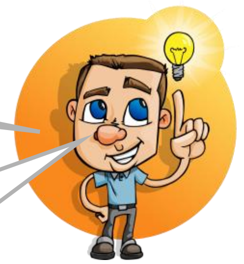
To predict the output for a test point, nearest neighbors will require computing distances from 15 training inputs. DT predicts the label by doing just at most 2 feature-value comparisons! Way more fast!!!

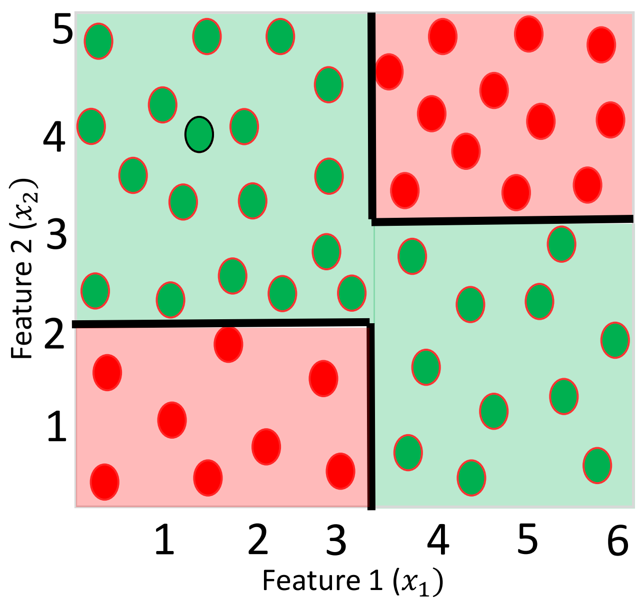
Constructing a Decision Tree

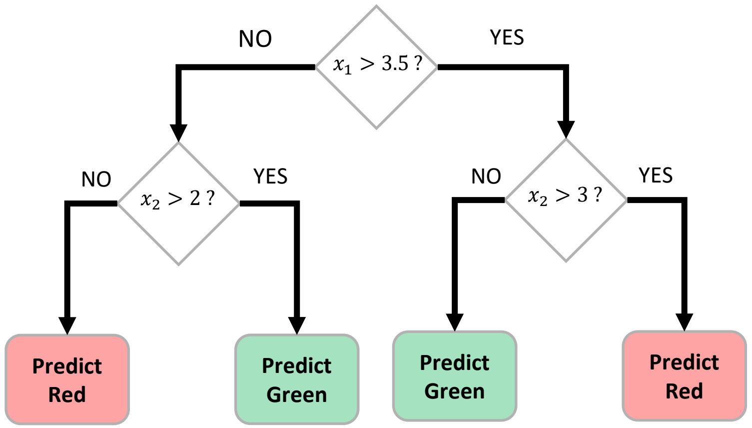
Given some training data, what's the "optimal" DT?

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)

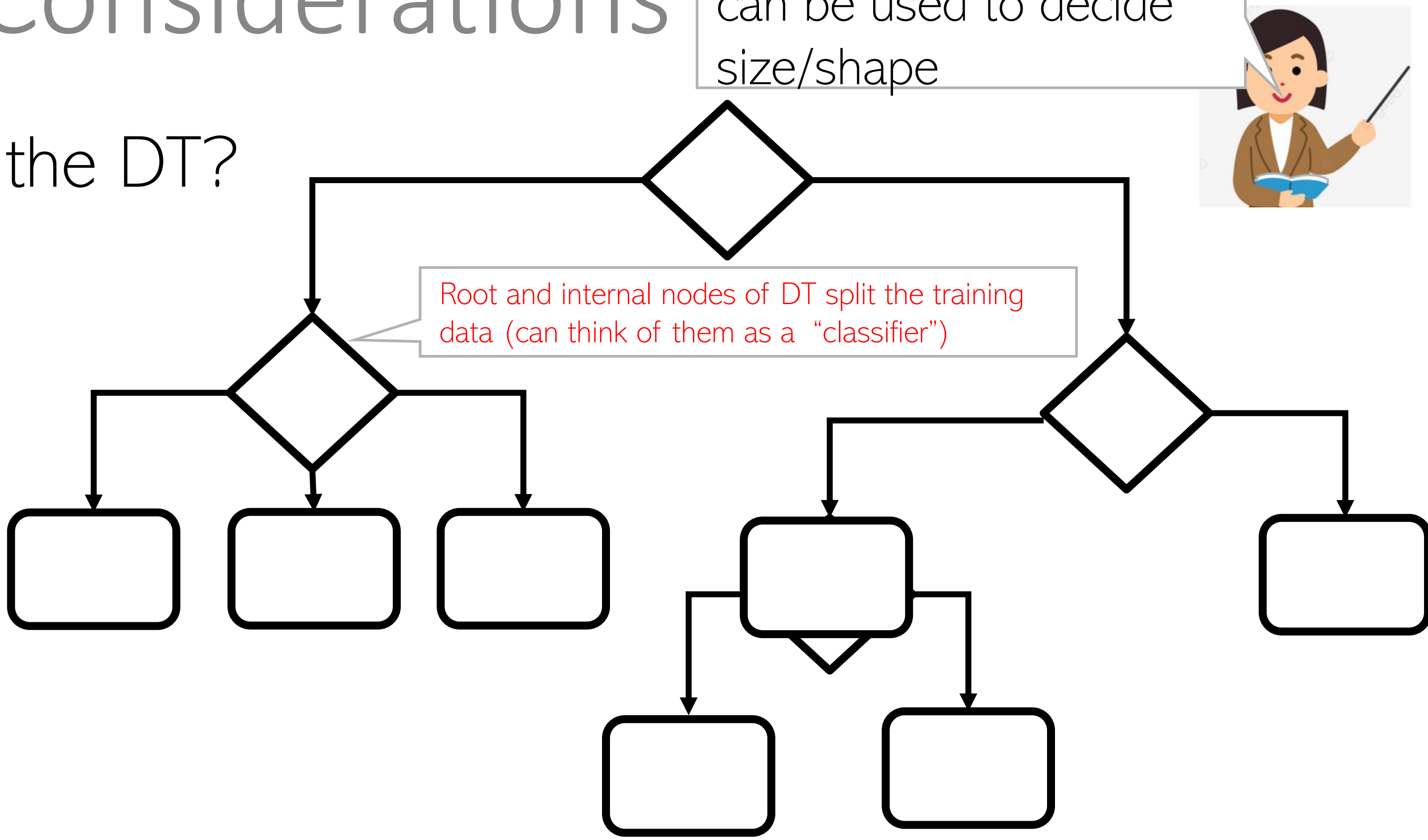
Informativeness of a rule is related to the extent of the purity of the split arising due to that rule. More informative rules yield more pure splits

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 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



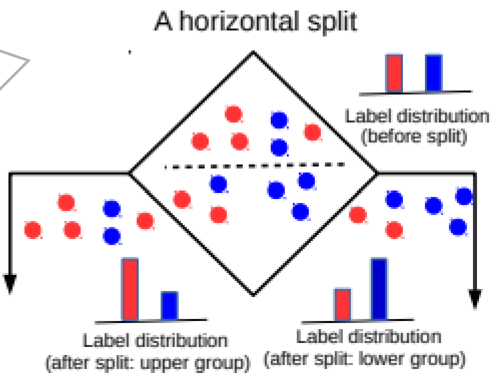
- Split criterion at root/int. nodes
  - Use another classifier?
  - Or maybe by doing a simpler test?
- What to do at the leaf node? Some options:
  - Make a constant prediction for each test input reaching there
  - Use a nearest neighbor based prediction using training inputs at that leaf node
  - Train and predict using some other sophisticated supervised learner on that node

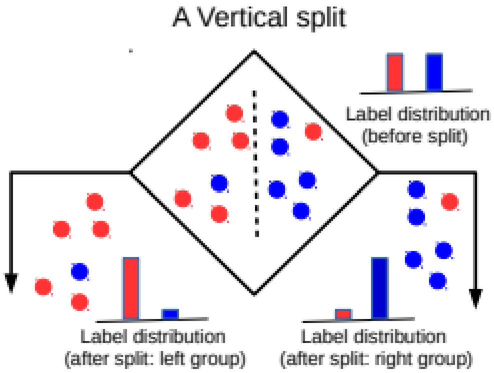
Usually, constant prediction at leaf nodes used since it will be very fast

Techniques to Split at Internal Nodes?

- 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")

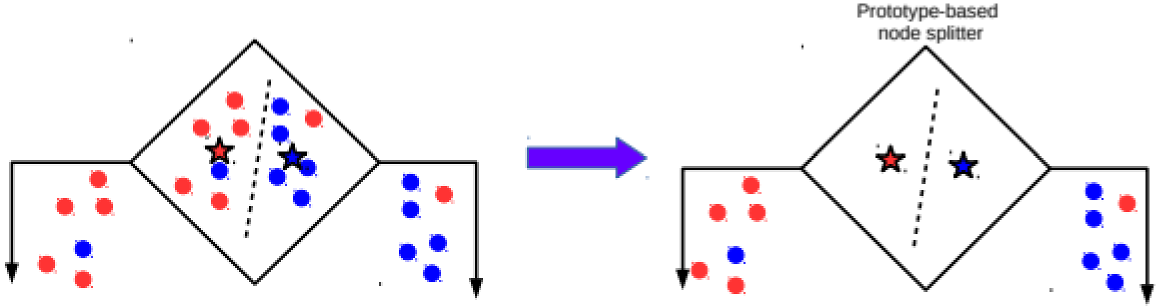
With this approach, all features (2 real-valued features in this example) and all possible values of each feature need to be evaluated in selecting the feature to be tested at each internal node. If features binary/discrete (only finite possible values), it is reasonably easy





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

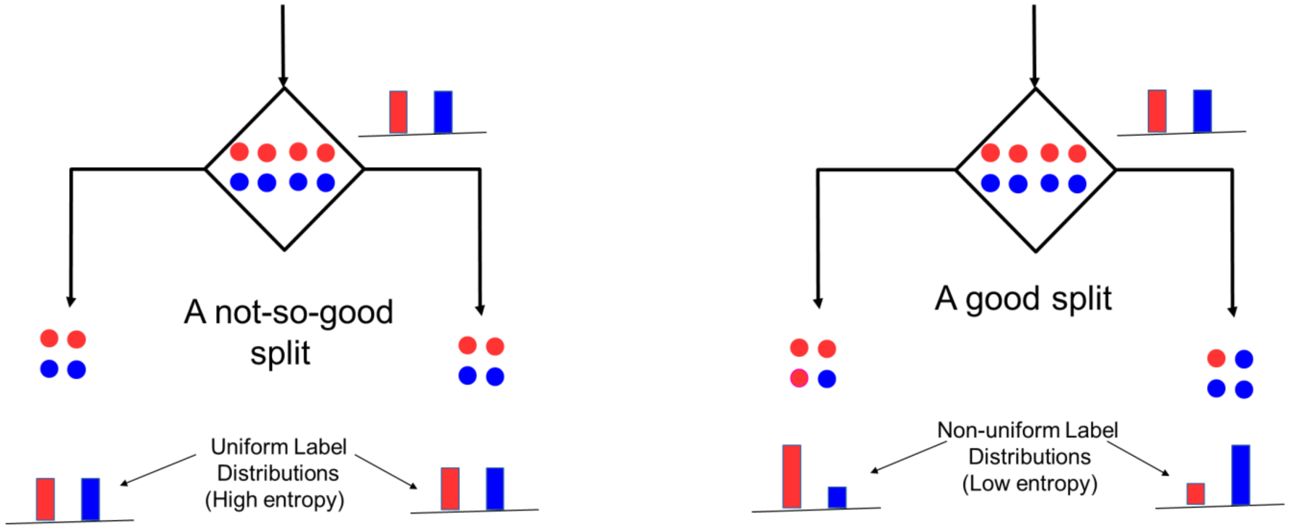
  - Testing the value of a combination of features (maybe 2-3 features)
  - Learning a classifier (e.g., LwP or some more sophisticated classifier)



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 sometimes used in some advanced DT methods

Internal Nodes: Good vs Bad Splits

- 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
  - Meaning: After split, in each group, majority of the inputs have the same label/output



- For classification problems (discrete outputs), entropy is a measure of purity
  - Low entropy $\Rightarrow$ 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")

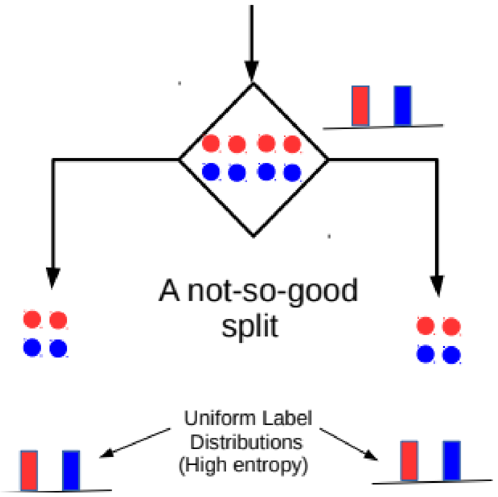
Entropy and Information Gain

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

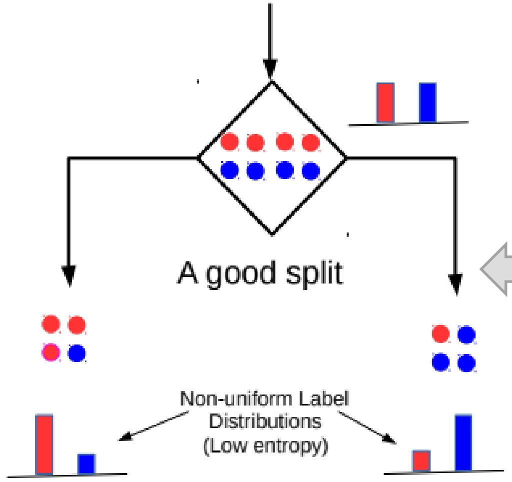
Uniform sets (all classes roughly equally present) have high entropy; skewed sets low

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

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



VS

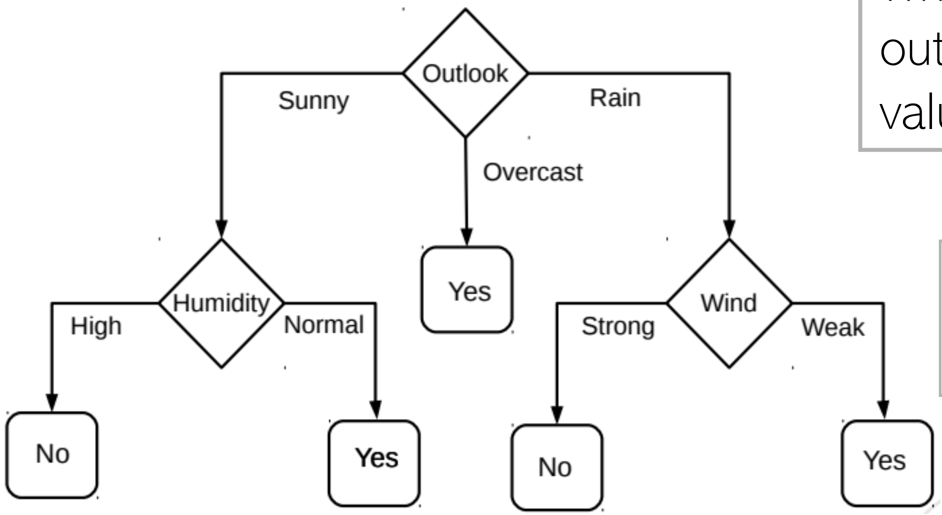


This split has higher IG

Decision Tree 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



Why did we test outlook feature's value first?

Because outlook feature is the most informative (has highest IG) at the root node position

Example credit: Tom Mitchell

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 <u>all</u> 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

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

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

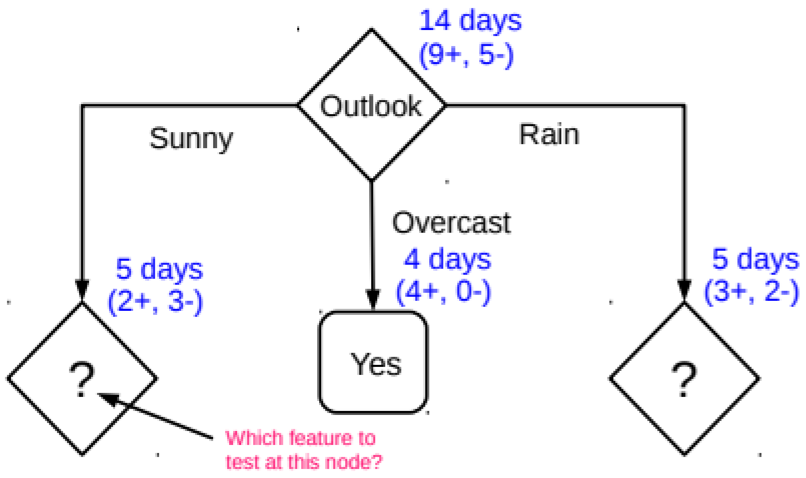
$$S_{\text{strong}} = [3+, 3-] \Rightarrow \text{H}(S_{\text{strong}}) = 1$$

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

- 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

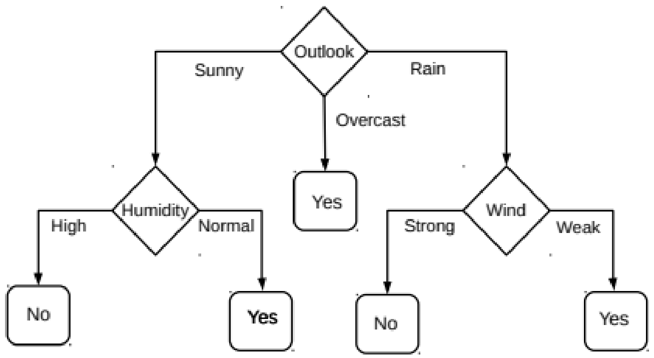
day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain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



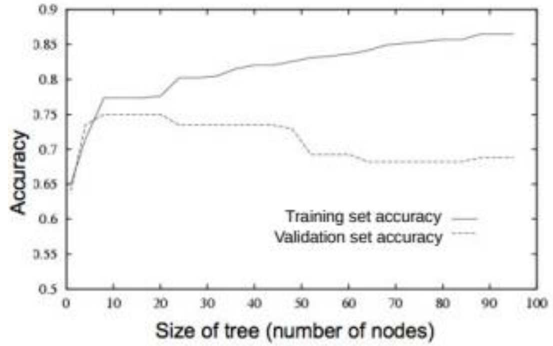
- Proceeding as before, for level 2, left node, we can verify that
  - IG(S,temp) = 0.570, IG(S, humidity) = 0.970, IG(S, wind) = 0.019
- Thus humidity chosen as the feature to be tested at level 2, left node
- No need to expand the middle node (already “pure” - all “yes” training examples ☺)
- Can also 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

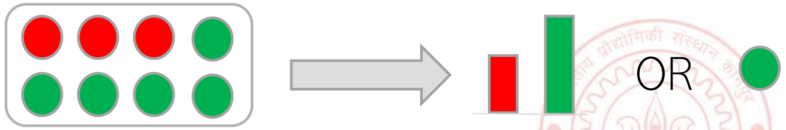


- Stop expanding a node further (i.e., make it a leaf node) when
  - It consist of all training examples having the same label (the node becomes "pure")
  - We run out of features to test along the path to that node
  - The DT starts to overfit (can be checked by monitoring the validation set accuracy)



To help prevent the tree from growing too much!

- **Important:** No need to obsess with 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)

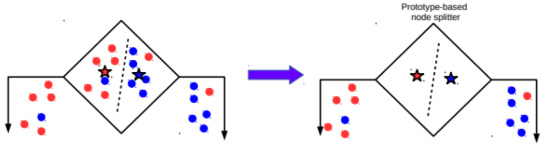
Either can be done using a validation set

- 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)

Decision Trees: Some Comments

For regression, outputs are real-valued and we don't have a "set" of classes, so quantities like entropy/IG/gini etc. are undefined

- 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 synthetic data 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 learner)
  - However, in DTs, we want the tests to be fast so single feature based rules are preferred
- Need to take care handling training or test inputs that have some features missing

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

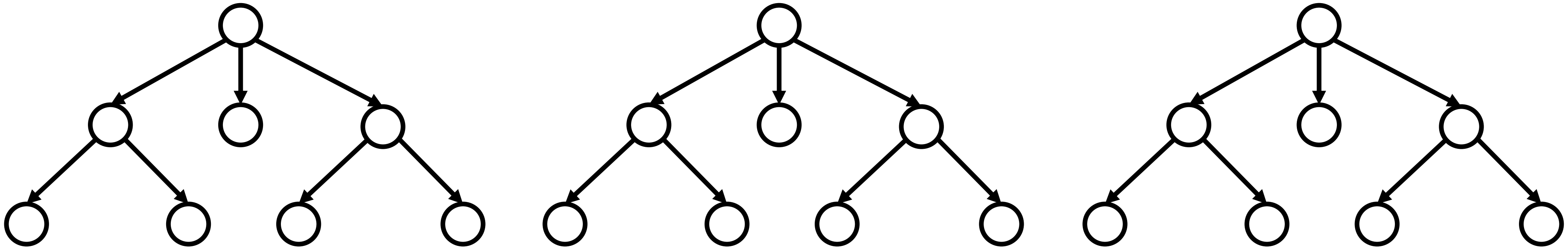
Ensemble of Trees

All trees can be trained in parallel

- Ensemble is a collection of models
- Each model makes a prediction. Take their majority as the final prediction
- Ensemble of trees is a collection of simple DTs
  - Often preferred as compared to a single massive, complicated tree
- A popular example: Random Forest (RF)

Each tree is trained on a subset of the training inputs/features

An RF with 3 simple trees. The majority prediction will be the final prediction



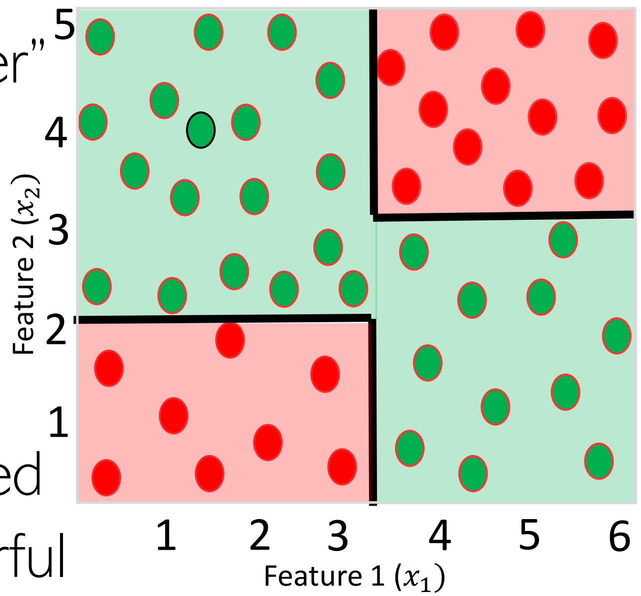
- XGBoost is another popular ensemble of trees
  - Based on the idea of "boosting" (will study boosting later) simple trees
  - Sequentially trains a set of trees with each correcting errors of previous ones

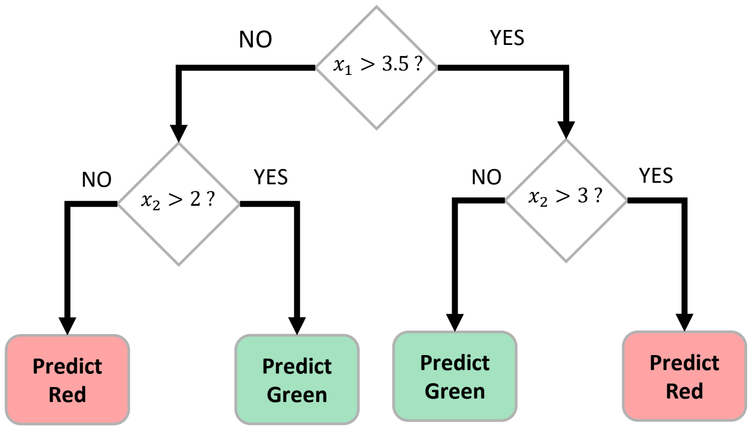
Decision Trees: A Summary

.. 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 simple DTs can be combined via ensemble methods: more powerful
- Used in several real-world ML applications, e.g., recommender systems, gaming (Kinect)





Human-body pose estimation

Some key weaknesses:

- Learning optimal DT is (NP-hard) intractable. Existing algos mostly greedy heuristics
- Can sometimes become very complex unless some pruning is applied