Decision Trees

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Data Mining and Machine Learning: Techniques and Algorithms

Eneldo Loza Mencía

eneldo@ke.tu-darmstadt.de

Knowledge Engineering Group, TU Darmstadt

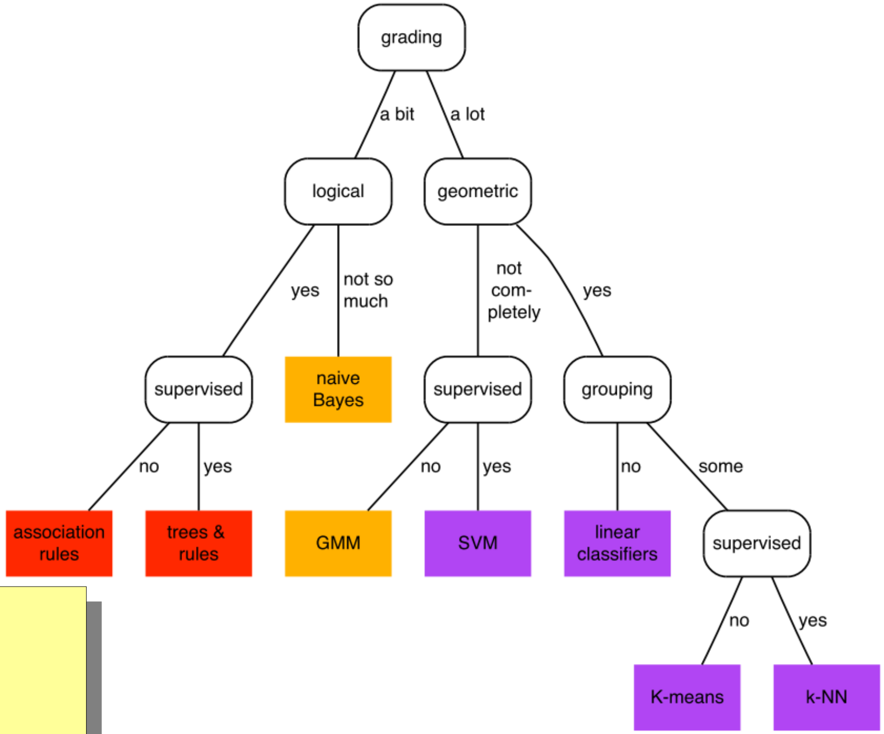
International Week 2019, 21.1. – 24.1.
University of Economics, Prague

Outline

- Introduction
  - Decision Trees
  - TDIDT: Top-Down Induction of Decision Trees
- ID3
  - Attribute selection
  - Entropy, Information, Information Gain
  - Gain Ratio
- C4.5
  - Missing Values
  - Numeric Values
  - Split Encoding
  - Pruning
- Regression and Model Trees

Decision Trees

- a decision tree consists of
  - **Nodes:**
    - test for the value of a certain attribute
  - **Edges:**
    - correspond to the outcome of a test
    - connect to the next node or leaf
  - **Leaves:**
    - terminal nodes that predict the outcome



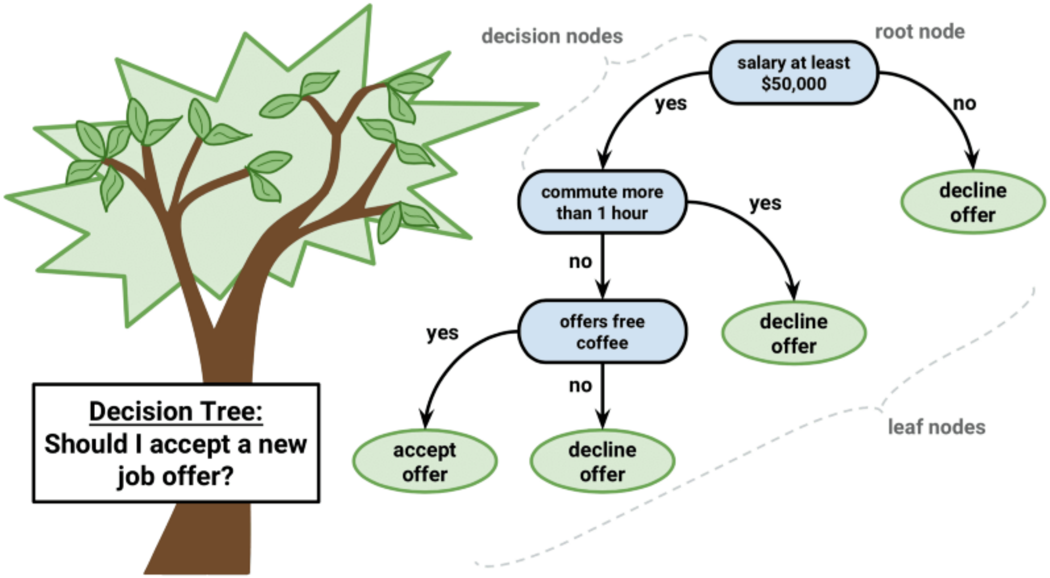
to classifiy an example:

1. start at the root
2. perform the test
3. follow the edge corresponding to outcome
4. goto 2. unless leaf
5. predict that outcome associated with the leaf

https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

Decision Trees

- a decision tree consists of
  - **Nodes:**
    - test for the value of a certain attribute
  - **Edges:**
    - correspond to the outcome of a test
    - connect to the next node or leaf
  - **Leaves:**
    - terminal nodes that predict the outcome



to classifiy an example:

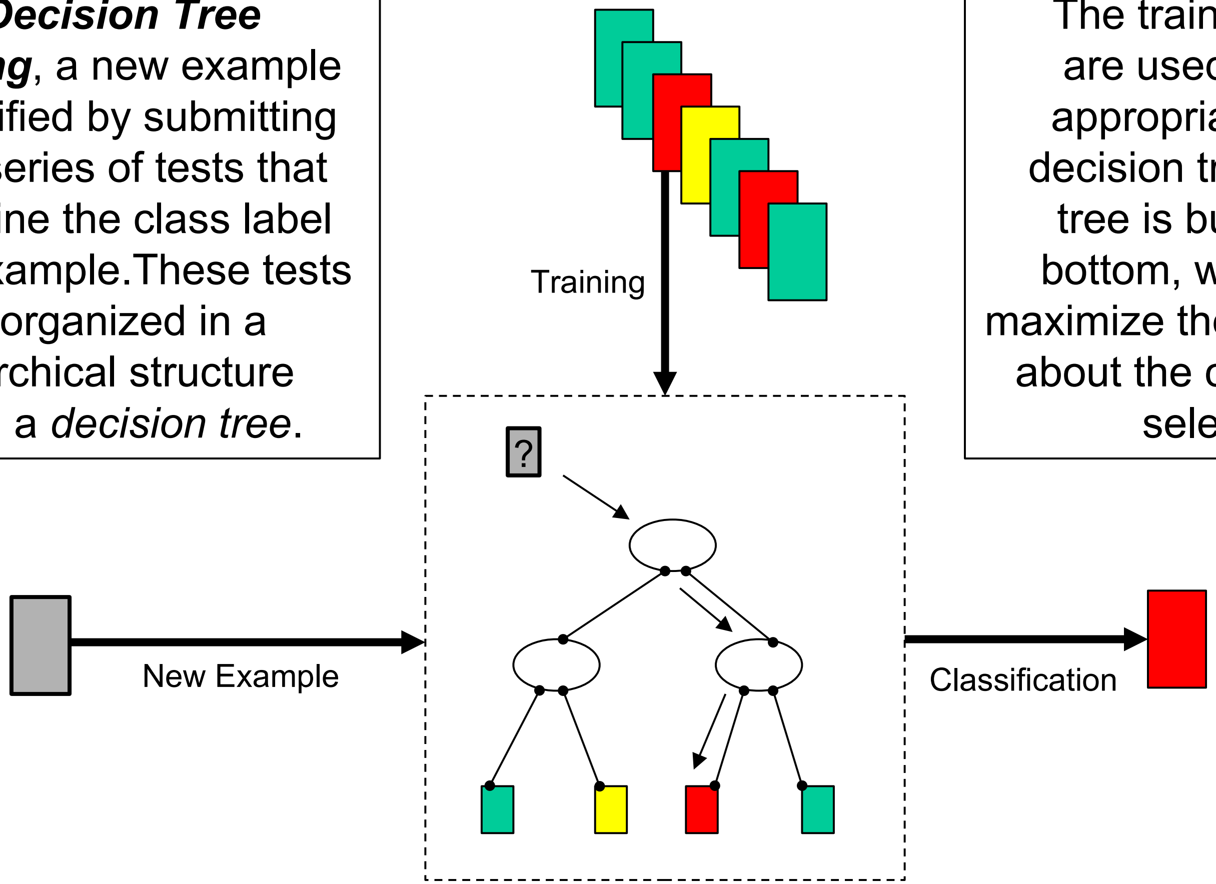
1. start at the root
2. perform the test
3. follow the edge corresponding to outcome
4. goto 2. unless leaf
5. predict that outcome associated with the leaf

https://towardsdatascience.com/a-tour-of-the-top-10-algorithms-for-machine-learning-newbies-dde4edffae11

Decision Tree Learning

In ***Decision Tree Learning***, a new example is classified by submitting it to a series of tests that determine the class label of the example.These tests are organized in a hierarchical structure called a *decision tree*.

The training examples are used for choosing appropriate tests in the decision tree. Typically, a tree is built from top to bottom, where tests that maximize the information gain about the classification are selected first.

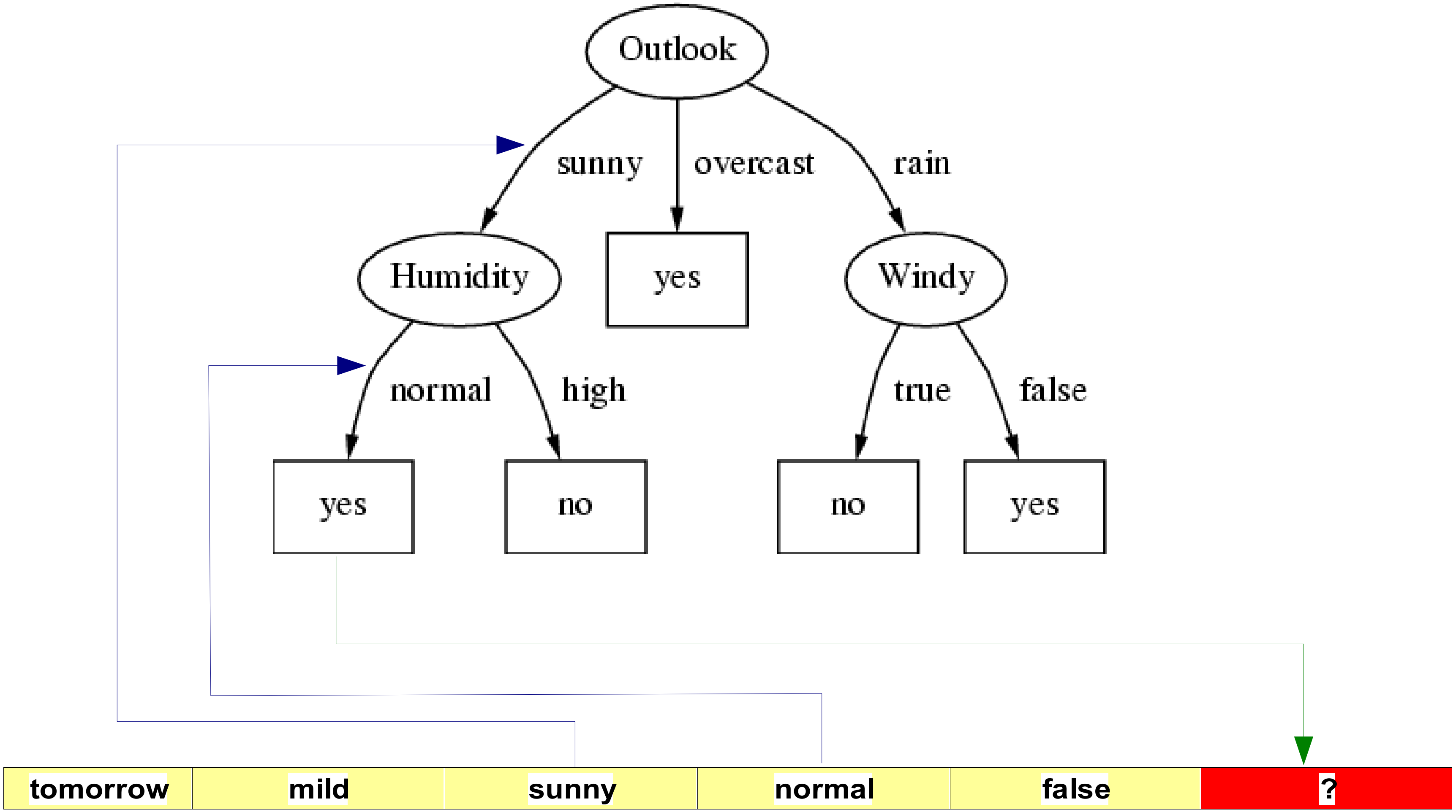


A Sample Task

Day	Temperature	Outlook	Humidity	Windy	Play Golf?
07-05	hot	sunny	high	false	no
07-06	hot	sunny	high	true	no
07-07	hot	overcast	high	false	yes
07-09	cool	rain	normal	false	yes
07-10	cool	overcast	normal	true	yes
07-12	mild	sunny	high	false	no
07-14	cool	sunny	normal	false	yes
07-15	mild	rain	normal	false	yes
07-20	mild	sunny	normal	true	yes
07-21	mild	overcast	high	true	yes
07-22	hot	overcast	normal	false	yes
07-23	mild	rain	high	true	no
07-26	cool	rain	normal	true	no
07-30	mild	rain	high	false	yes

today	cool	sunny	normal	false	?
tomorrow	mild	sunny	normal	false	?

Decision Tree Learning



Divide-And-Conquer Algorithms

- Family of decision tree learning algorithms
  - TDIDT: Top-Down Induction of Decision Trees
- Learn trees in a Top-Down fashion:
  - divide the problem in subproblems
  - solve each problem

Basic Divide-And-Conquer Algorithm:

1. select a test for root node
   Create branch for each possible outcome of the test
2. split instances into subsets
   One for each branch extending from the node
3. repeat recursively for each branch, using only instances that reach the branch
4. stop recursion for a branch if all its instances have the same class

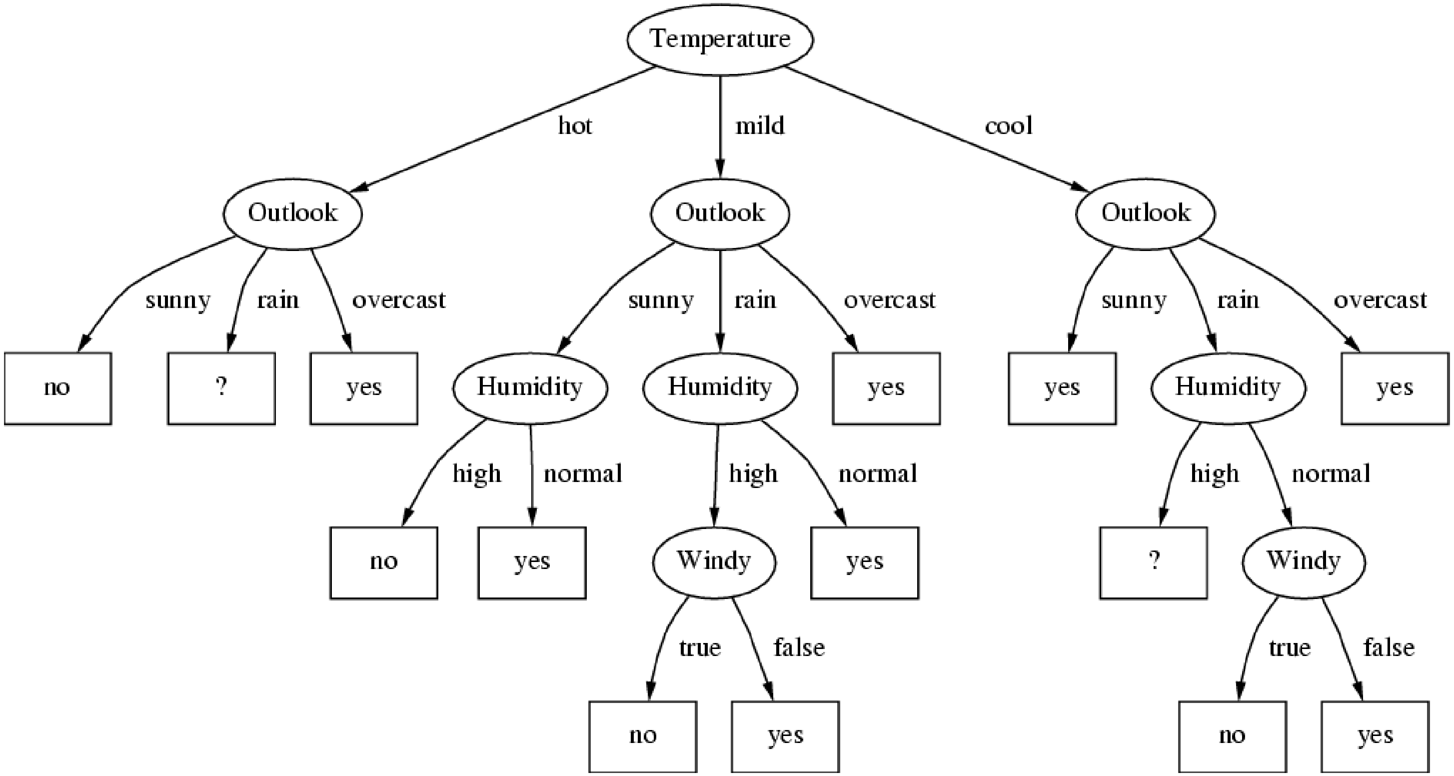
ID3 Algorithm

Function ID3

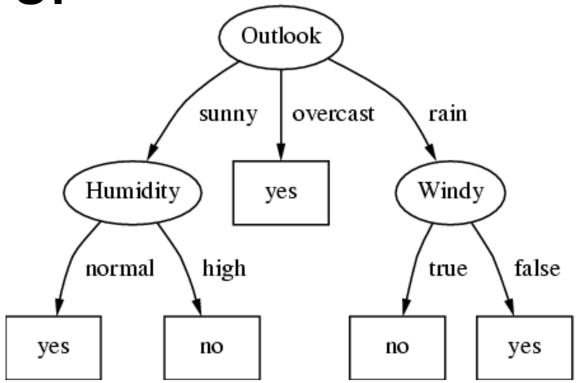
- **Input:** Example set S
- **Output:** Decision Tree DT

- If all examples in S belong to the same class c
  - return a new leaf and label it with c
- Else
  i. Select an attribute A according to some heuristic function
  ii. Generate a new node DT with A as test
  iii. For each Value v_i of A
    (a) Let S_i = all examples in S with $A = v_i$
    (b) Use ID3 to construct a decision tree DT_i for example set S_i
    (c) Generate an edge that connects DT and DT_i

A Different Decision Tree

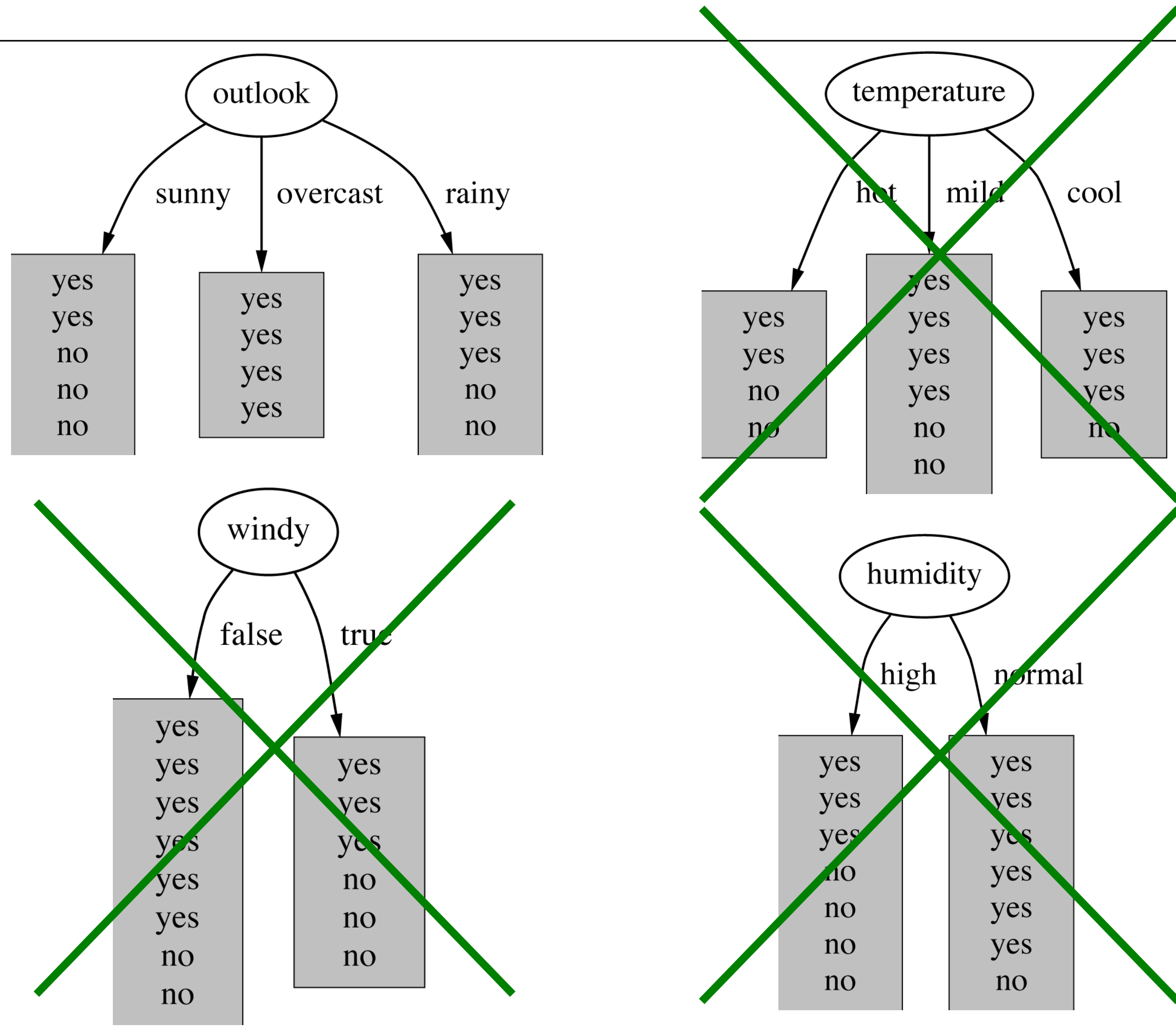


vs.



- also explains all of the training data
- will it generalize well to new data?

Which attribute to select as the root?



What is a good Attribute?

- We want to grow a simple tree
  → a good heuristic prefers attributes that split the data so that each successor node is as *pure* as posssible
  - i.e., the distribution of examples in each node is so that it mostly contains examples of a single class
- In other words:
  - We want a measure that prefers attributes that have a high degree of „order“:
    - Maximum order: All examples are of the same class
    - Minimum order: All classes are equally likely

  → Entropy is a measure for (un-)orderedness
  - Another interpretation:
    - Entropy is the amount of information that is contained in the node
    - all examples of the same class → no information

Entropy (for two classes)

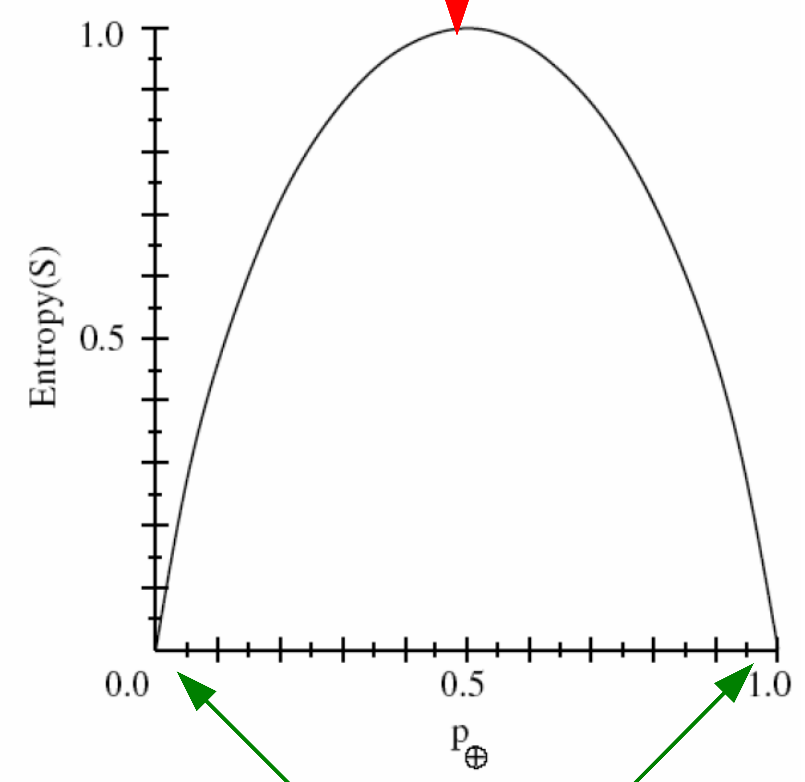
- S is a set of examples
- $p_\oplus$ is the proportion of examples in class $\oplus$
- $p_\ominus = 1 - p_\oplus$ is the proportion of examples in class $\ominus$

Entropy:

$$E(S) = -p_\oplus \cdot \log_2 p_\oplus - p_\ominus \cdot \log_2 p_\ominus$$

- Interpretation:
  - amount of unorderedness in the class distribution of S

maximal value at equal class distribution



minimal value if only one class left in S

Example: Attribute Outlook

- Outlook = sunny: 2 examples yes, 3 examples no

$$E(\text{Outlook}=\text{sunny})=-\frac{2}{5}\log_2\left(\frac{2}{5}\right)-\frac{3}{5}\log_2\left(\frac{3}{5}\right)=0.971$$

- Outlook = overcast: 4 examples yes, 0 examples no

$$E(\text{Outlook}=\text{overcast})=-1\cdot\log_2(1)-0\cdot\log_2(0)=0$$

Note: this is normally undefined. Here: = 0

- Outlook = rainy : 3 examples yes, 2 examples no

$$E(\text{Outlook}=\text{rainy})=-\frac{3}{5}\log_2\left(\frac{3}{5}\right)-\frac{2}{5}\log_2\left(\frac{2}{5}\right)=0.971$$

`

Entropy (for more classes)

Entropy can be easily generalized for $n > 2$ classes

- p_i is the proportion of examples in S that belong to the i-th class

$$E(S) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 \ldots - p_n \log_2 p_n = -\sum_{i=1}^{n} p_i \log_2 p_i$$

- Calculation can be simplified using absolute counts c_i of examples in class i instead of fractions
  - If :

$$p_i = \frac{c_i}{|S|}$$

  - Example: $E(S) = -\sum_{i=1}^{n} p_i \log_2 p_i = -\frac{1}{|S|} \cdot \left(\sum_{i=1}^{n} c_i \log_2 c_i - |S| \cdot \log_2 |S| \right)$

$$E([2,3,4]) = -\tfrac{2}{9} \cdot \log_2\left(\tfrac{2}{9}\right) - \tfrac{3}{9} \cdot \log_2\left(\tfrac{3}{9}\right) - \tfrac{4}{9} \cdot \log_2\left(\tfrac{4}{9}\right)$$
$$= -\tfrac{1}{9}\left(2 \cdot \log_2(2) + 3 \cdot \log_2(3) + 4 \cdot \log_2(4) - 9 \cdot \log_2(9)\right)$$

Average Entropy / Information

- **Problem:**
  - Entropy only computes the quality of a single (sub-)set of examples
    - corresponds to a single value
  - How can we compute the quality of the entire split?
    - corresponds to an entire attribute
- **Solution:**
  - Compute the weighted average over all sets resulting from the split
    - weighted by their size

$$I(S, A) = \sum_i \frac{|S_i|}{|S|} \cdot E(S_i)$$

- **Example:**
  - Average entropy for attribute *Outlook*:

$$I(\text{Outlook}) = \frac{5}{14} \cdot 0.971 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.971 = 0.693$$

Information Gain

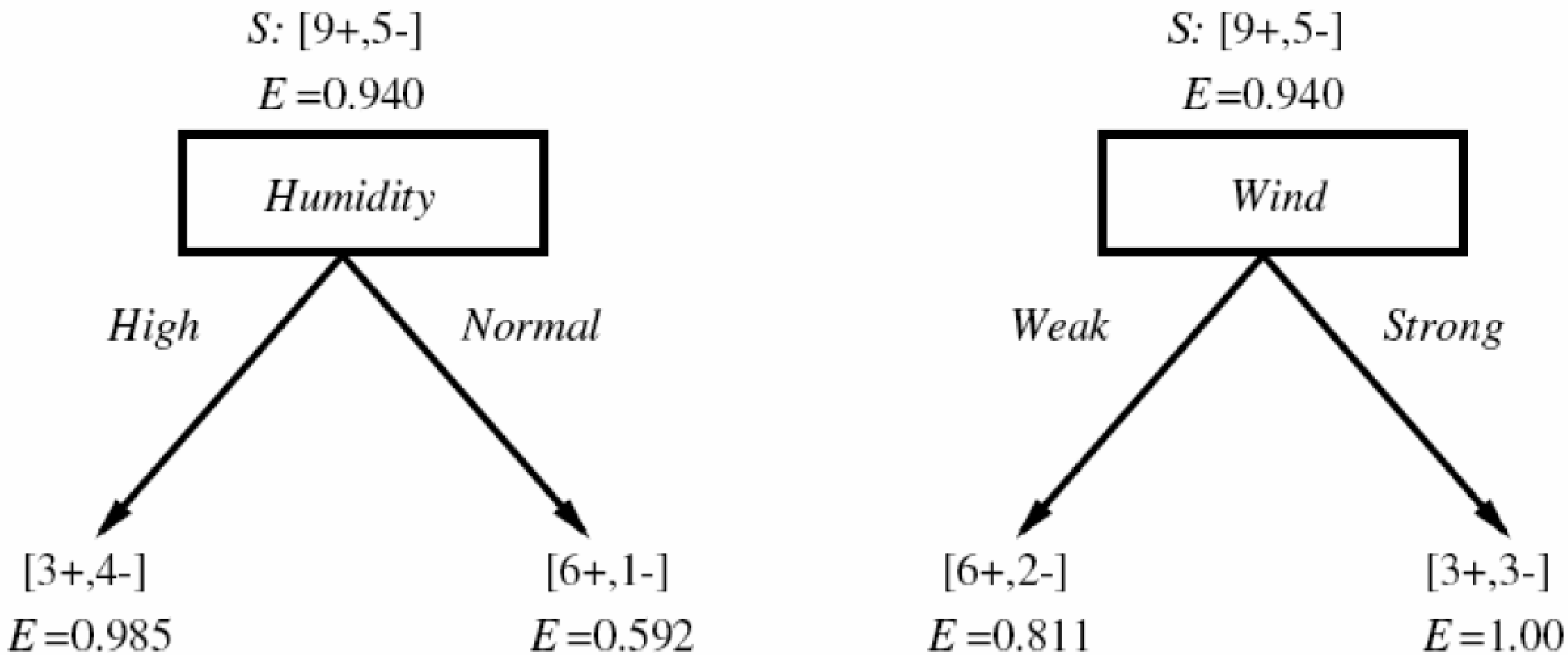
- When an attribute A splits the set S into subsets S_i
  - we compute the average entropy
  - and compare the sum to the entropy of the original set S

Information Gain for Attribute A

$$Gain(S, A) = E(S) - I(S, A) = E(S) - \sum_i \frac{|S_i|}{|S|} \cdot E(S_i)$$

- The attribute that maximizes the difference is selected
  - i.e., the attribute that reduces the unorderedness most!
- **Note:**
  - maximizing information gain is equivalent to minimizing average entropy, because $E(S)$ is constant for all attributes A

Example



Gain (S, Humidity)

= .940 - (7/14).985 - (7/14).592
= .151

Gain (S, Wind)

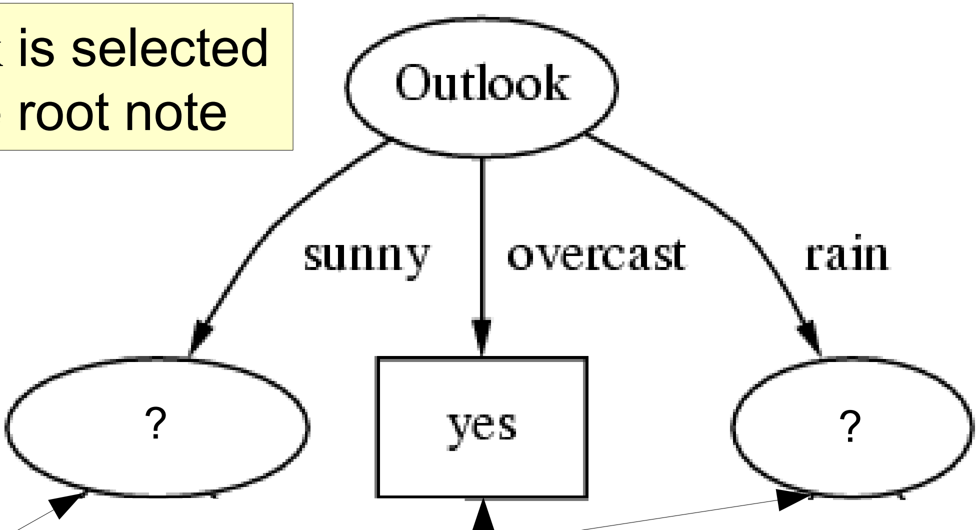
= .940 - (8/14).811 - (6/14)1.0
= .048

$Gain(S, Outlook) = 0.246$

$Gain(S, Temperature) = 0.029$

Example (Ctd.)

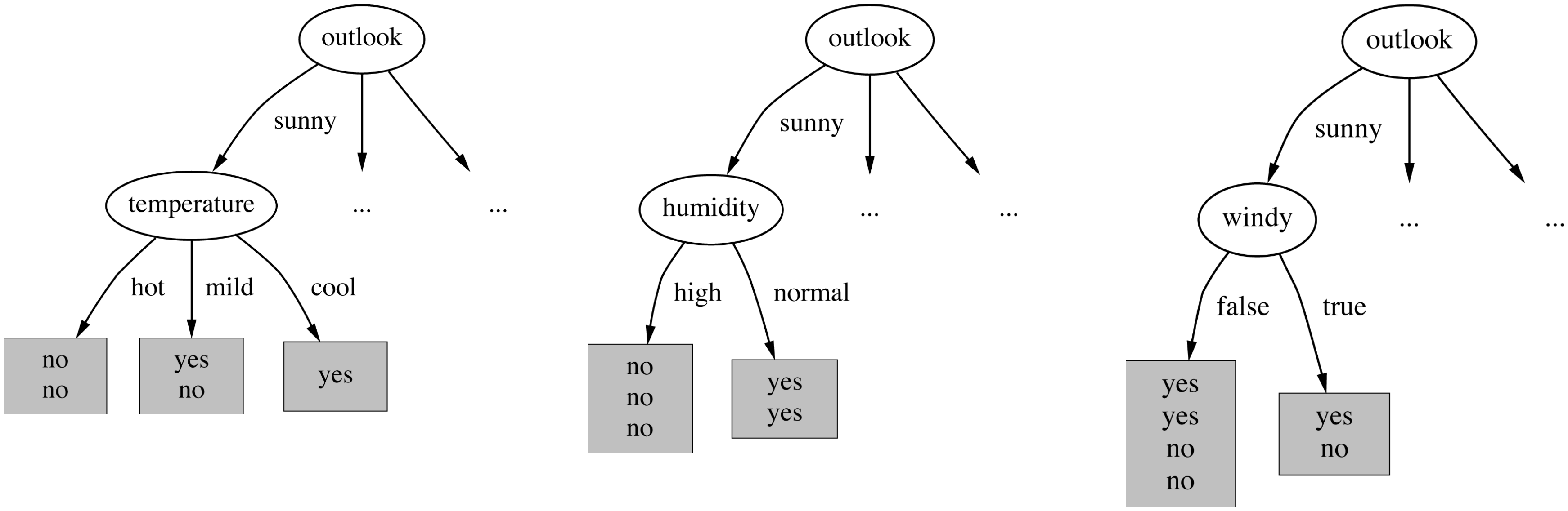
Outlook is selected as the root note



further splitting necessary

Outlook = overcast contains only examples of class **yes**

Example (Ctd.)



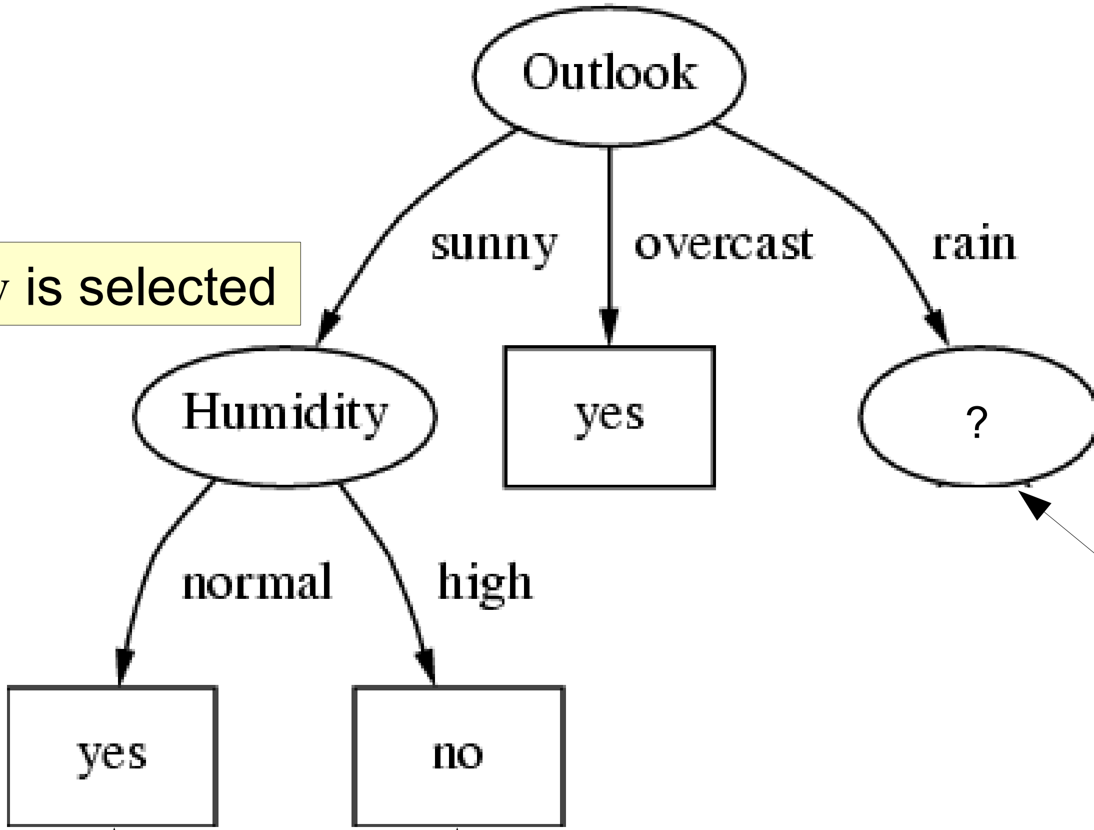
$$\begin{aligned} \text{Gain}(\mathit{Temperature}\,) &= 0.571 \text{ bits} \\ \text{Gain}(\mathit{Humidity}\,) &= 0.971 \text{ bits} \\ \text{Gain}(\mathit{Windy}\,) &= 0.020 \text{ bits} \end{aligned}$$

Humidity is selected

Example (Ctd.)

TECHNISCHE UNIVERSITÄT DARMSTADT

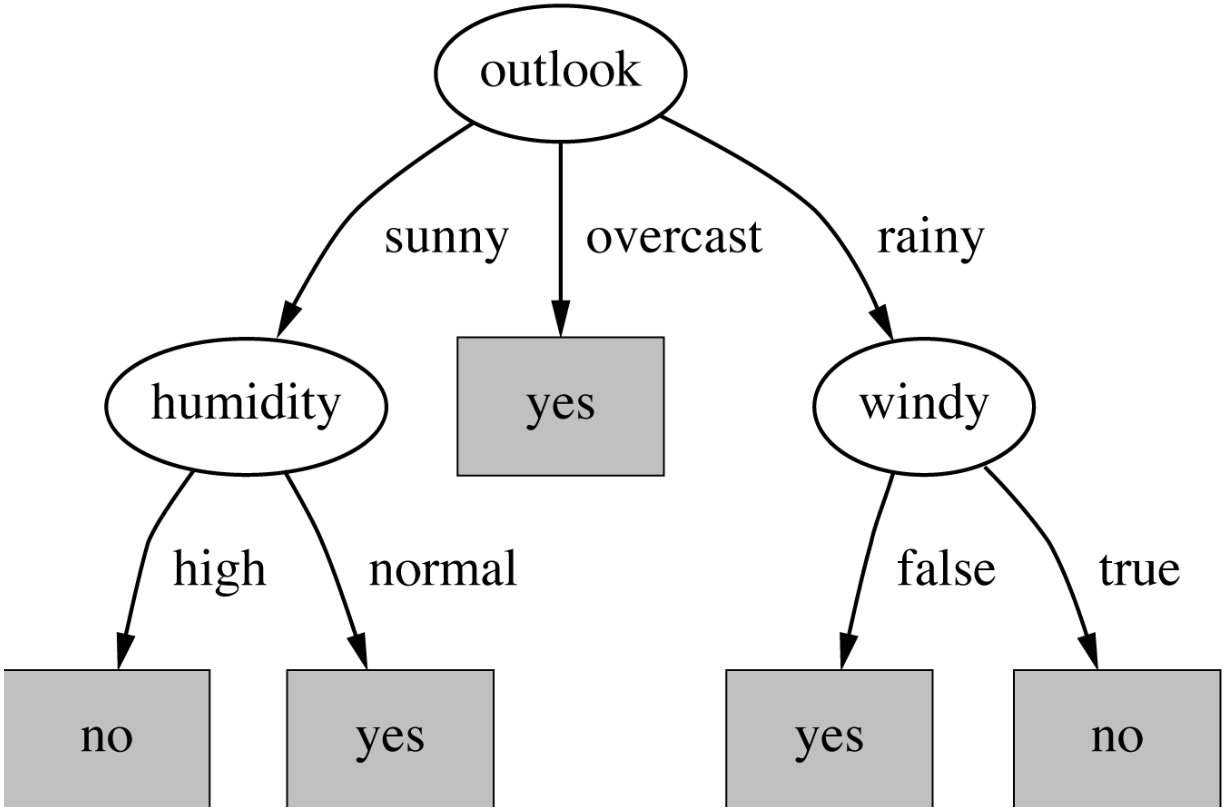
Humidity is selected



further splitting necessary

Pure leaves
→ No further expansion necessary

Final decision tree



Gini Index

- Many alternative measures to Information Gain
- Most popular altermative: Gini index
  - used in e.g., in CART (Classification And Regression Trees)
  - impurity measure (instead of entropy)

$$Gini(S) = \sum_i p_i \cdot (1 - p_i) = 1 - \sum_i p_i^2$$

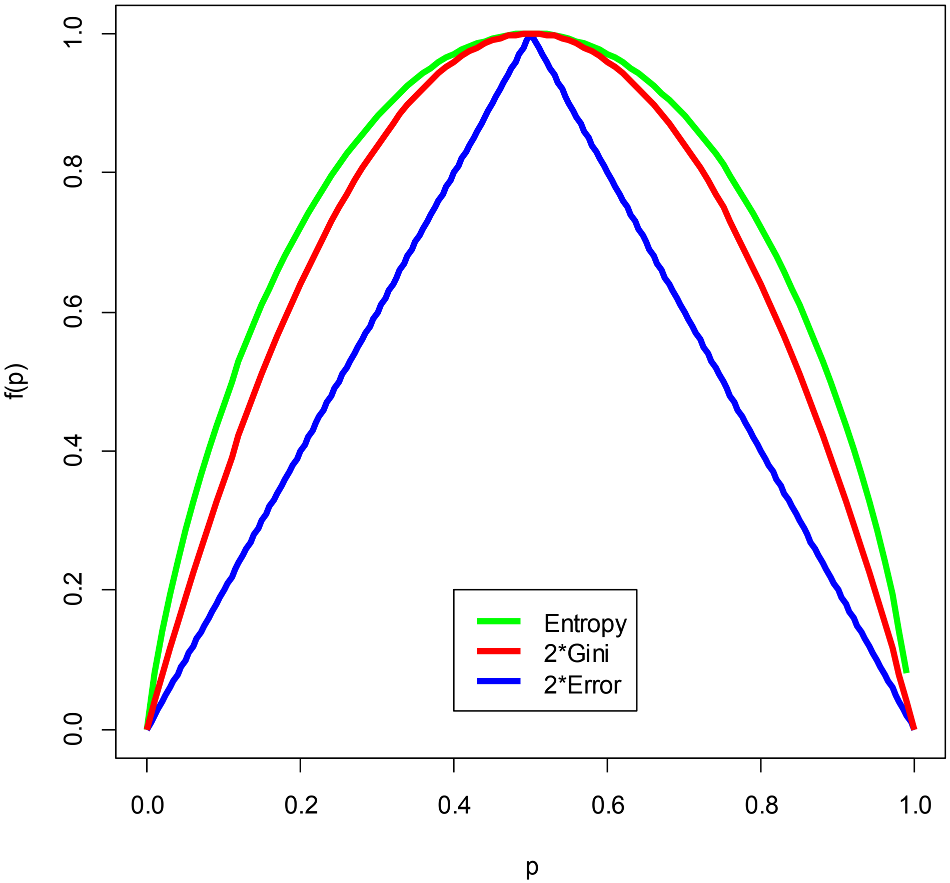
  - average Gini index (instead of average entropy / information)

$$Gini(S, A) = \sum_i \frac{|S_i|}{|S|} \cdot Gini(S_i)$$

- Gini Gain
  - could be defined analogously to information gain
  - but typically averageGini index is minimized instead of maximizing Gini gain

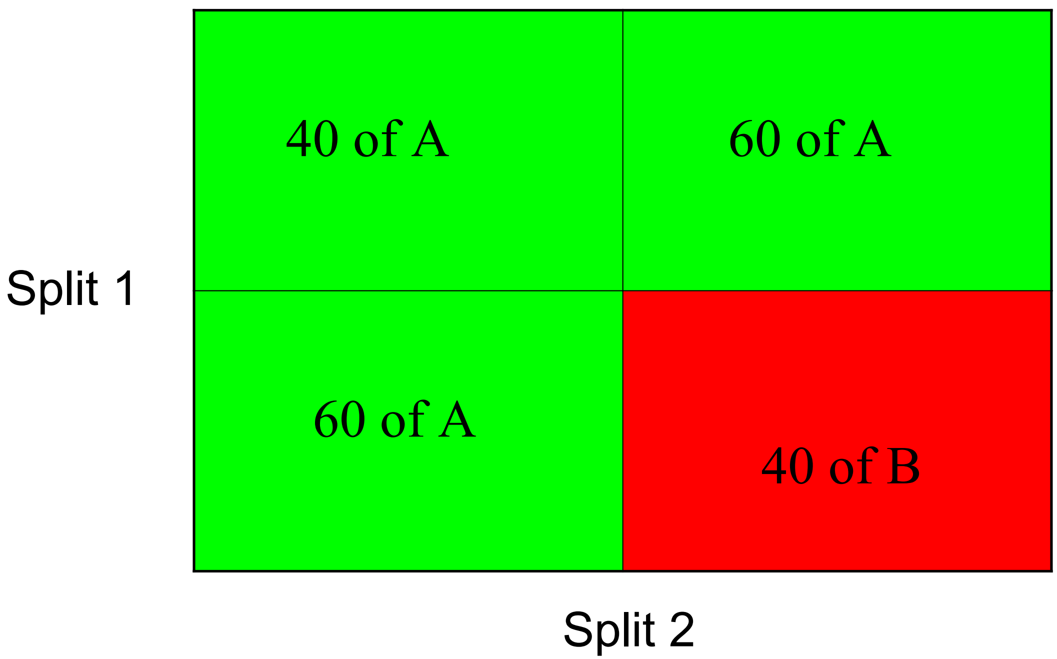
Comparison of Splitting Criteria

For a 2-class problem:



Why not use Error as a Splitting Criterion?

- Reason:
  - The bias towards pure leaves is not strong enough
- Example 1: Data set with 160 Examples A, 40 Examples B
  - → Error rate without splitting is 20%



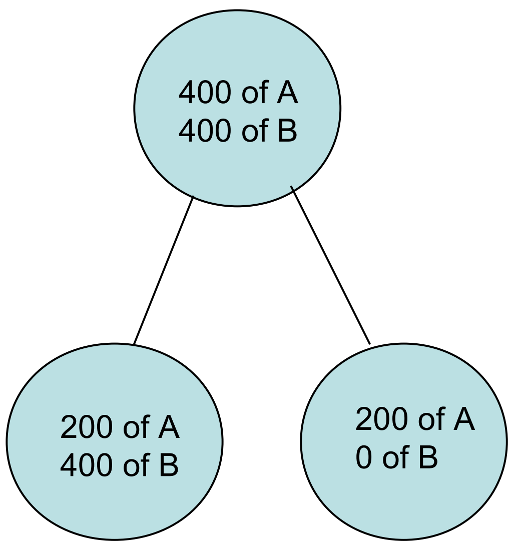
For each of the two splits, the total error after splitting is also (0% + 40%)/2 = 20% → no improvement

However, together both splits would give a perfect classfier.

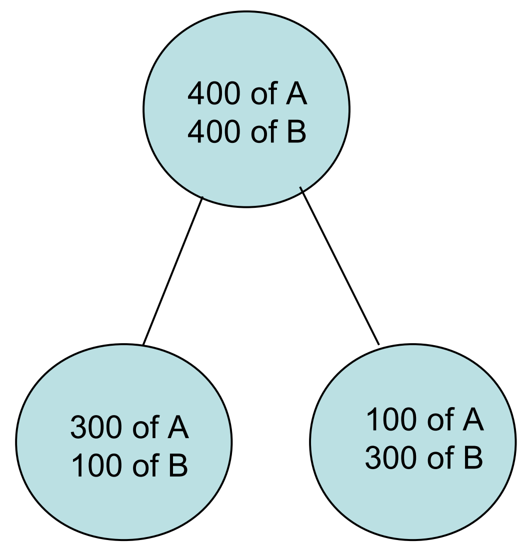
Based on a slide by Richard Lawton

Why not use Error as a Splitting Criterion?

- Reason:
  - The bias towards pure leaves is not strong enough
- Example 2:
  - Dataset with 400 examples of class A and 400 examples of class B



Error rate = 25%



Error rate = 25%

Based on a slide by Richard Lawton

Industrial-strength algorithms

- For an algorithm to be useful in a wide range of real-world applications it must:
  - Permit numeric attributes
  - Allow missing values
  - Be robust in the presence of noise
  - Be able to approximate arbitrary concept descriptions (at least in principle)

→ ID3 needs to be extended to be able to deal with real-world data

- Result: **C4.5**
  - Best-known and (probably) most widely-used learning algorithm
    - original C-implementation at http://www.rulequest.com/Personal/
  - Re-implementation of C4.5 Release 8 e.g. in Weka: J4.8
  - Commercial successor: C5.0
    - freely available e.g. in R

Missing values

- Examples are classified as usual
  - if we are lucky, attributes with missing values are not tested by the tree
- If an attribute with a missing value needs to be tested:
  - split the instance into fractional instances (*pieces*)
  - one piece for each outgoing branch of the node
  - a piece going down a branch receives a weight proportional to the popularity of the branch
  - weights sum to 1
- Info gain or gain ratio work with fractional instances
  - use sums of weights instead of counts
- during classification, split the instance in the same way
  - Merge probability distribution using weights of fractional instances

Numeric attributes

- Standard method: binary splits
  - E.g. temp < 45
- Unlike nominal attributes, every attribute has many possible split points
- Solution is straightforward extension:
  - Evaluate info gain (or other measure) for every possible split point of attribute
  - Choose “best” split point
  - Info gain for best split point is info gain for attribute

→ Computationally more demanding than splits on discrete attributes

Example

- Assume a numerical attribute for Temperature
- First step:
  - Sort all examples according to the value of this attribute
  - Could look like this:

64	65	68	69	70	71	72	72	75	75	80	81	83	85
Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No

Temperature < 71.5: yes/4, no/2

Temperature ≥ 71.5: yes/5, no/3

$$I(\text{Temperature} \ @ \ 71.5) = \frac{6}{14} \cdot E(\text{Temperature} < 71.5) + \frac{8}{14} E(\text{Temperature} \geq 71.5) = 0.939$$

- Split points can be placed between values or directly at values
- Has to be computed for all pairs of neighboring values

Efficient Computation

- Efficient computation needs only one scan through the values!
  - Linearly scan the sorted values, each time updating the count matrix and computing the evaluation measure
  - Choose the split position that has the best value

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Taxable Income									
Sorted Values →	60	70	75	85	90	95	100	120	125	220

Efficient Computation

- Efficient computation needs only one scan through the values!
  - Linearly scan the sorted values, each time updating the count matrix and computing the evaluation measure
  - Choose the split position that has the best value

	Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
		Taxable Income									
Sorted Values →		60	70	75	85	90	95	100	120	125	220

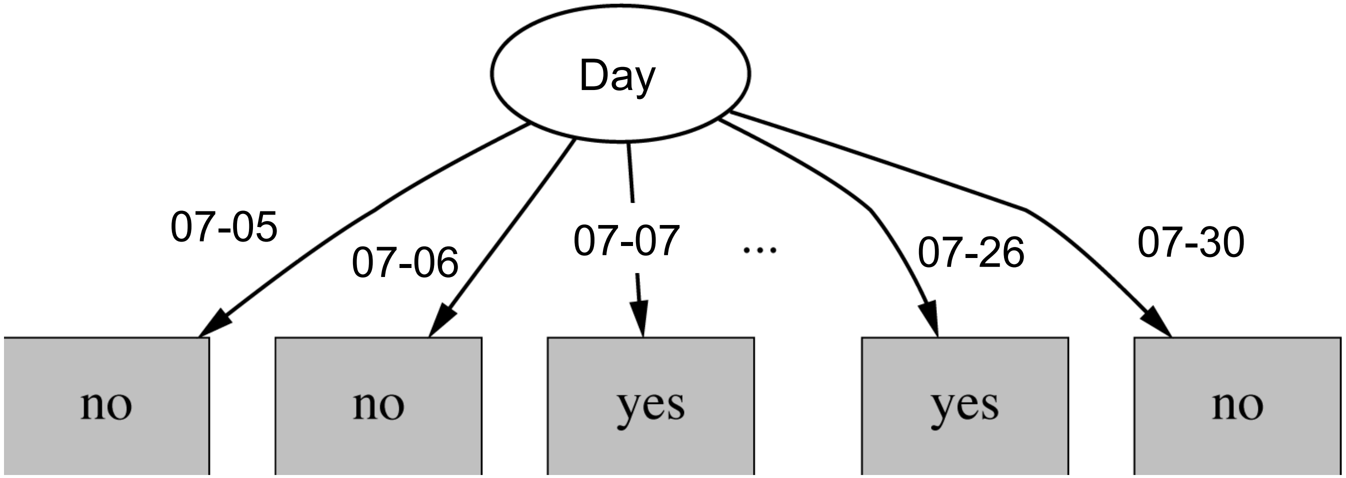
Split Positions →	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		*<u>0.300</u>*		0.343		0.375		0.400		0.420	

Highly-branching attributes

Problematic: attributes with a large number of values

- extreme case: each example has its own value
  - e.g. example ID; Day attribute in weather data
- Subsets are more likely to be pure if there is a large number of different attribute values
  - Information gain is biased towards choosing attributes with a large number of values
- This may cause several problems:
  - Overfitting
    - selection of an attribute that is non-optimal for prediction
  - Fragmentation
    - data are fragmented into (too) many small sets

Decision Tree for Day attribute



- Entropy of split:

$$I(\text{Day}) = \frac{1}{14}\left(E([0,1]) + E([0,1]) + \ldots + E([0,1])\right) = 0$$

  - Information gain is maximal for Day (0.940 bits)

Split Encoding

Categorical Encoding

- use at it is
- #branches=#categories

Numeric Encoding

- enumerate categories
- ordering might be aleatoric
- DT treats feature as numeric

Categorical Feature		Numeric
Louise	=>	1
Gabriel	=>	2
Emma	=>	3
Adam	=>	4
Alice	=>	5
Raphael	=>	6
Chloe	=>	7
Louis	=>	8
Jeanne	=>	9
Arthur	=>	10

https://medium.com/data-design/
visiting-categorical-features-and-encoding-in-decision-trees-53400fa65931

Split Encoding

One-Hot Encoding

- one binary feature for each category
- also very popular representation e.g. for neural networks
- #features=#categories

Categorical Feature		f1	f2	f3	f4	f5	f6	f7	f8	f9	f10
Louise	=>	1	0	0	0	0	0	0	0	0	0
Gabriel	=>	0	1	0	0	0	0	0	0	0	0
Emma	=>	0	0	1	0	0	0	0	0	0	0
Adam	=>	0	0	0	1	0	0	0	0	0	0
Alice	=>	0	0	0	0	1	0	0	0	0	0
Raphael	=>	0	0	0	0	0	1	0	0	0	0
Chloe	=>	0	0	0	0	0	0	1	0	0	0
Louis	=>	0	0	0	0	0	0	0	1	0	0
Jeanne	=>	0	0	0	0	0	0	0	0	1	0
Arthur	=>	0	0	0	0	0	0	0	0	0	1

Binary Encoding

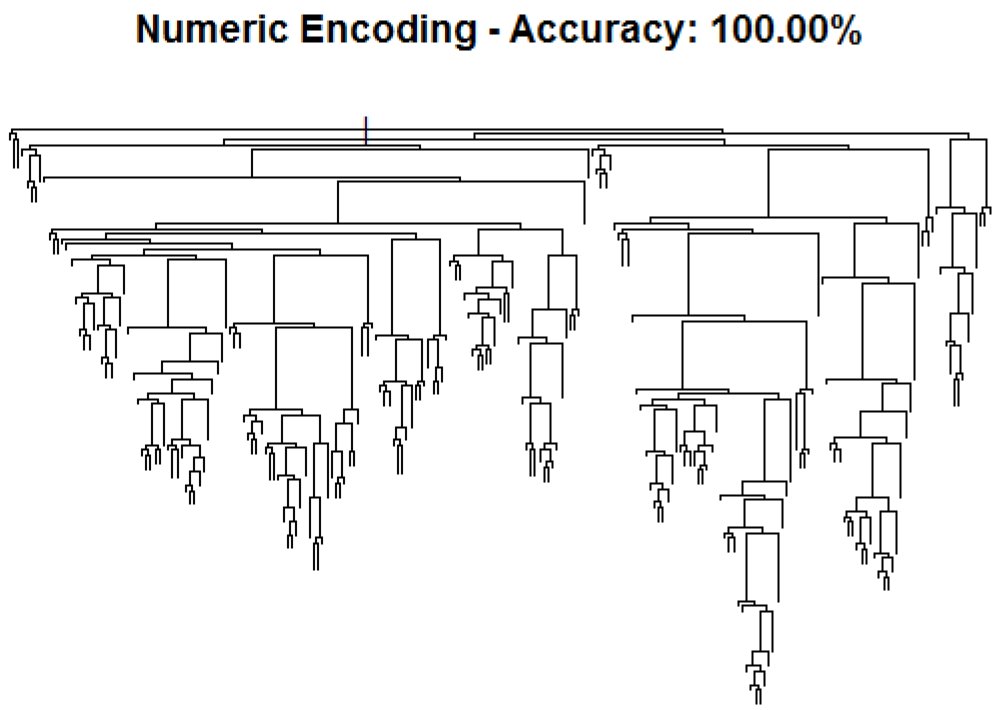
- unique binary encoding for each category
- #features=log2(#categories+1)

			Binary Encoded			
Categorical Feature		=	x1	x2	x4	x8
Louise	=>	1	1	0	0	0
Gabriel	=>	2	0	1	0	0
Emma	=>	3	1	1	0	0
Adam	=>	4	0	0	1	0
Alice	=>	5	1	0	1	0
Raphael	=>	6	0	1	1	0
Chloe	=>	7	1	1	1	0
Louis	=>	8	0	0	0	1
Jeanne	=>	9	1	0	0	1
Arthur	=>	10	0	1	0	1

https://medium.com/data-design/visiting-categorical-features-and-encoding-in-decision-trees-53400fa65931

Split Encoding

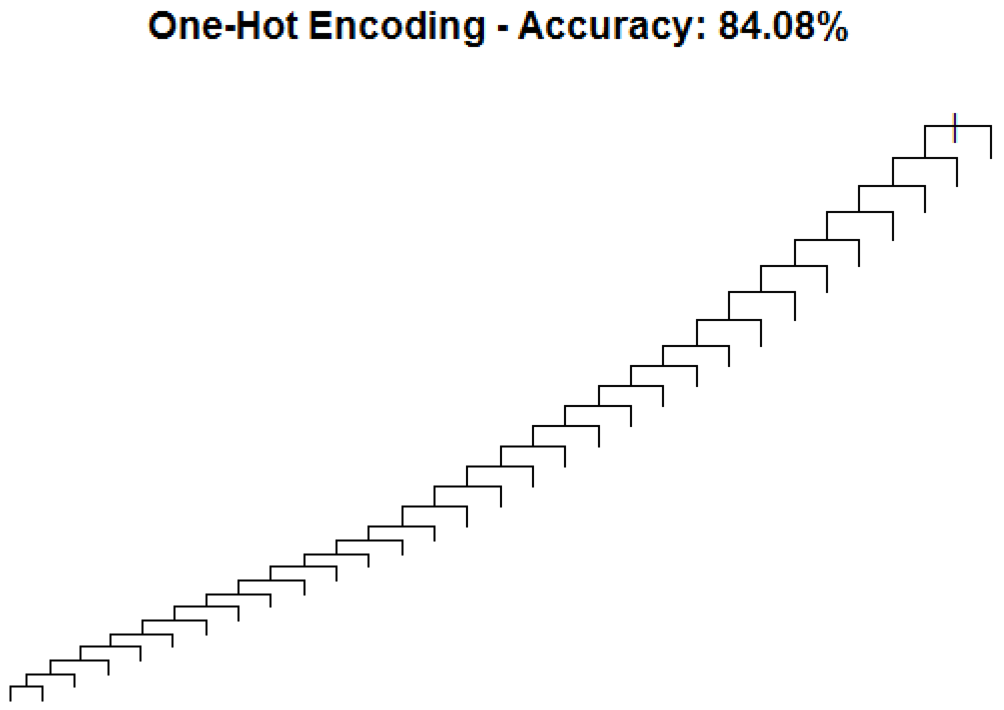
1024 categories and 25% positive labels



https://medium.com/data-design/
visiting-categorical-features-and-encoding-in-decision-trees-53400fa65931

Split Encoding

1024 categories and 25% positive labels

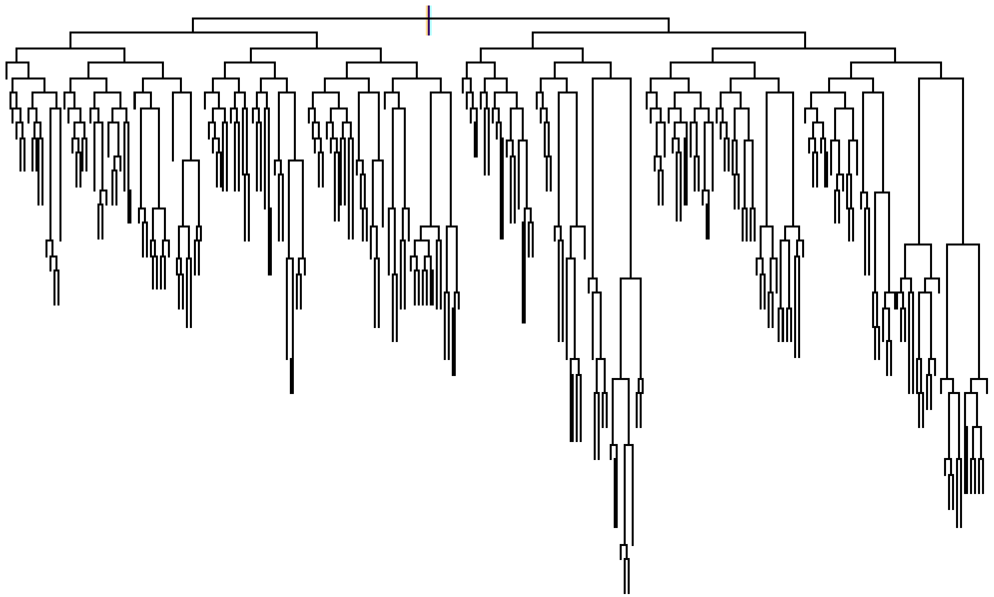


https://medium.com/data-design/
visiting-categorical-features-and-encoding-in-decision-trees-53400fa65931

Split Encoding

1024 categories and 25% positive labels

Binary Encoding - Accuracy: 99.61%

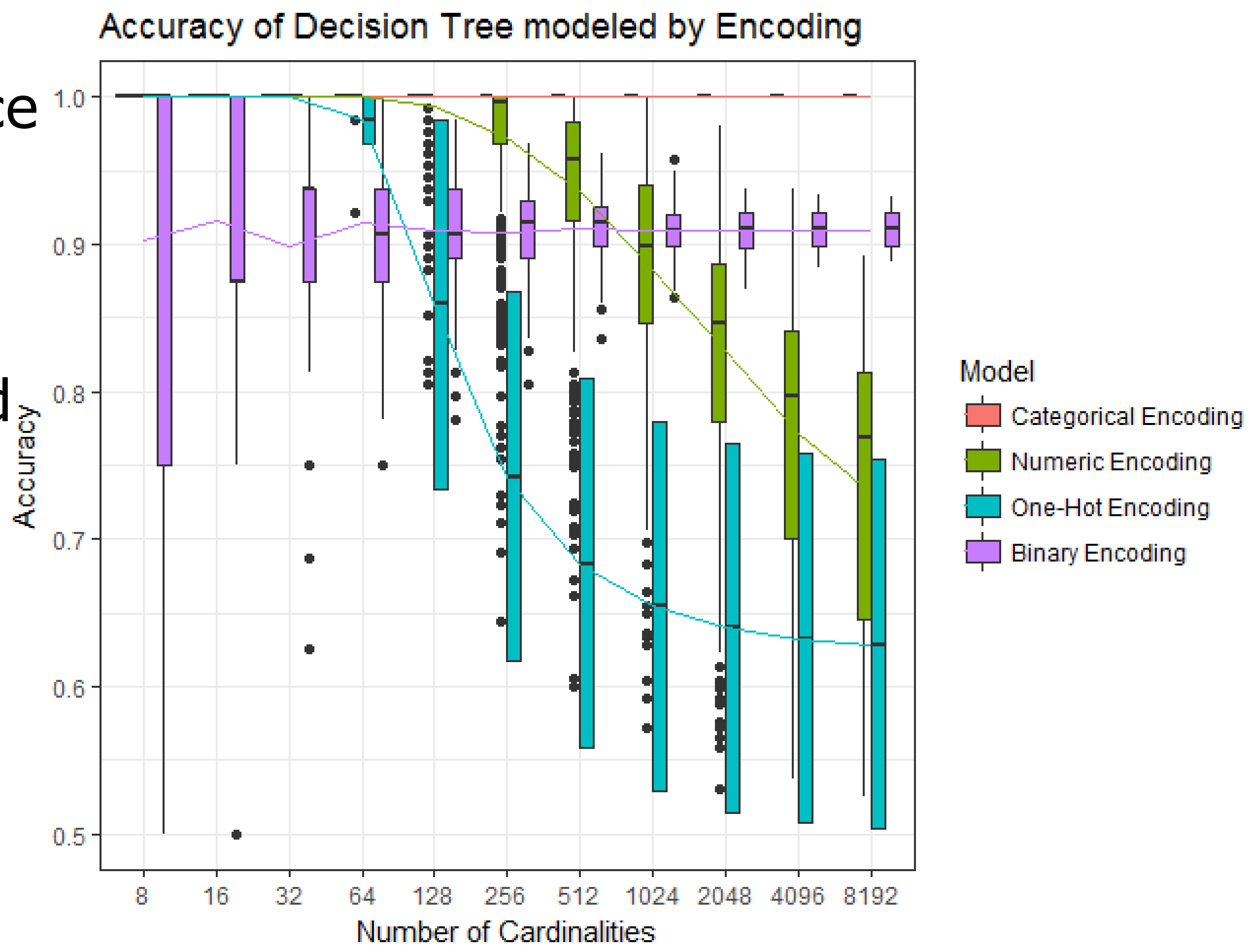


https://medium.com/data-design/visiting-categorical-features-and-encoding-in-decision-trees-53400fa65931

Split Encoding Example Results

- categorical encoding achieves best performance
  - but, in addition to overfitting and fragmentation problem, might be difficult to read
- one-hot encoding not beneficial for DT learning
  - less accurate
  - more computationally expensive for high number of categories

What other possibilities would make sense?



https://medium.com/data-design/visiting-categorical-features-and-encoding-in-decision-trees-53400fa65931

Overfitting and Pruning

TECHNISCHE UNIVERSITÄT DARMSTADT

- The smaller the complexity of a concept, the less danger that it overfits the data
  - A polynomial of degree n can always fit $n+1$ points
- Thus, learning algorithms try to keep the learned concepts simple
  - Note a „perfect“ fit on the training data can always be found for a decision tree! (except when data is contradictory)

Pre-Pruning:

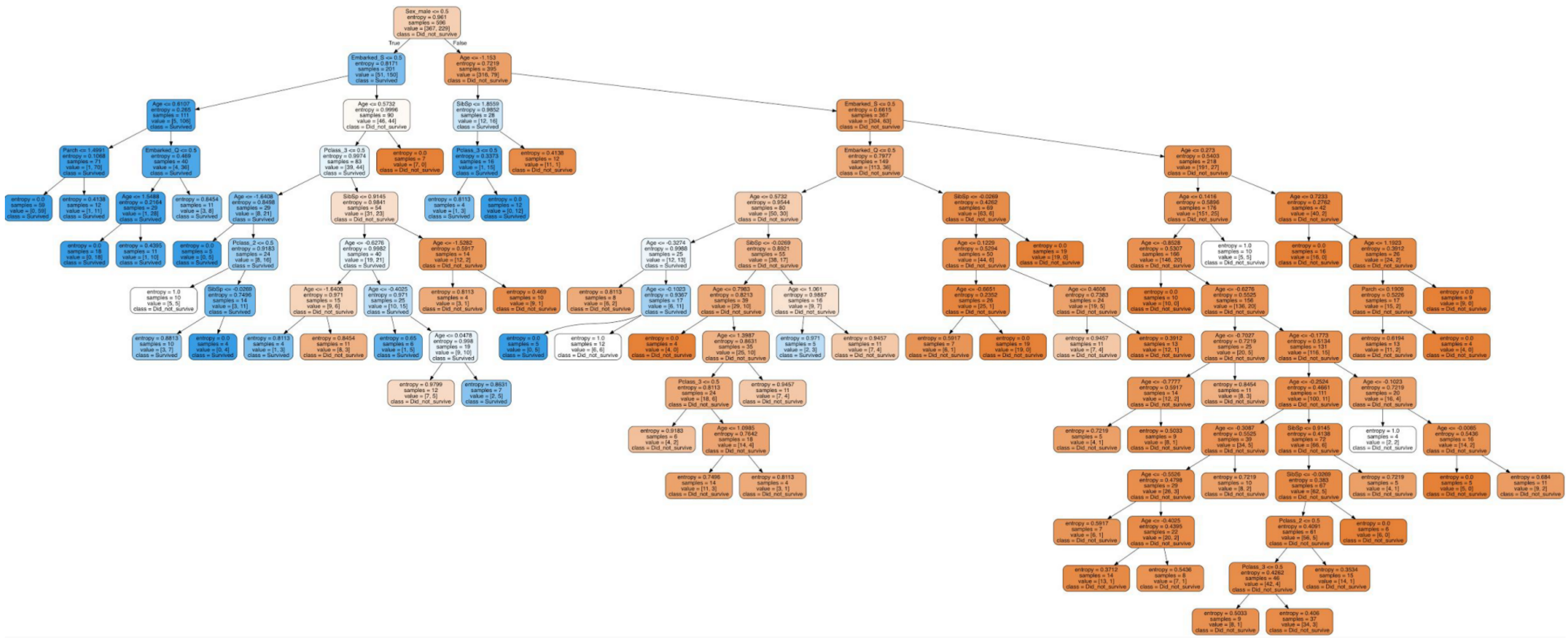
- stop growing a branch when information becomes unreliable

Post-Pruning:

- grow a decision tree that correctly classifies all training data
- simplify it later by replacing some nodes with leafs

- Post-pruning preferred in practice—pre-pruning can “stop early”

Overfitting and Pruning



Pre-pruning

- Based on statistical significance test
  - Stop growing the tree when there is no *statistically significant* association between any attribute and the class at a particular node
- Most popular test: *chi-squared test*
- ID3 used chi-squared test in addition to information gain
  - Only statistically significant attributes were allowed to be selected by information gain procedure

- C4.5 uses a simpler strategy
  - but combines it with → post-pruning
  - parameter -m: (default value $m=2$)
    each node above a leave must have
    - at least two successors
    - that contain at least m examples

Early Stopping

- Pre-pruning may stop the growth process prematurely: *early stopping*
- Classic example: XOR/Parity-problem
  - No individual attribute exhibits any significant association to the class

	a	b	class
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

  → In a dataset that contains XOR attributes a and b, and several irrelevant (e.g., random) attributes, ID3 can not distinguish between relevant and irrelevant attributes

  → Pre-pruning won't expand the root node

  - Structure is only visible in fully expanded tree
- But:
  - XOR-type problems rare in practice
  - pre-pruning is faster than post-pruning

Post-pruning

TECHNISCHE UNIVERSITÄT DARMSTADT

- basic idea
  - first grow a full tree to capture all possible attribute interactions
  - later remove those that are due to chance

1. learn a complete and consistent decision tree that classifies all examples in the training set correctly
2. as long as the performance increases
   - try simplification operators on the tree
   - evaluate the resulting trees
   - make the replacement that results in the best estimated performance
3. return the resulting decision tree

Post-pruning

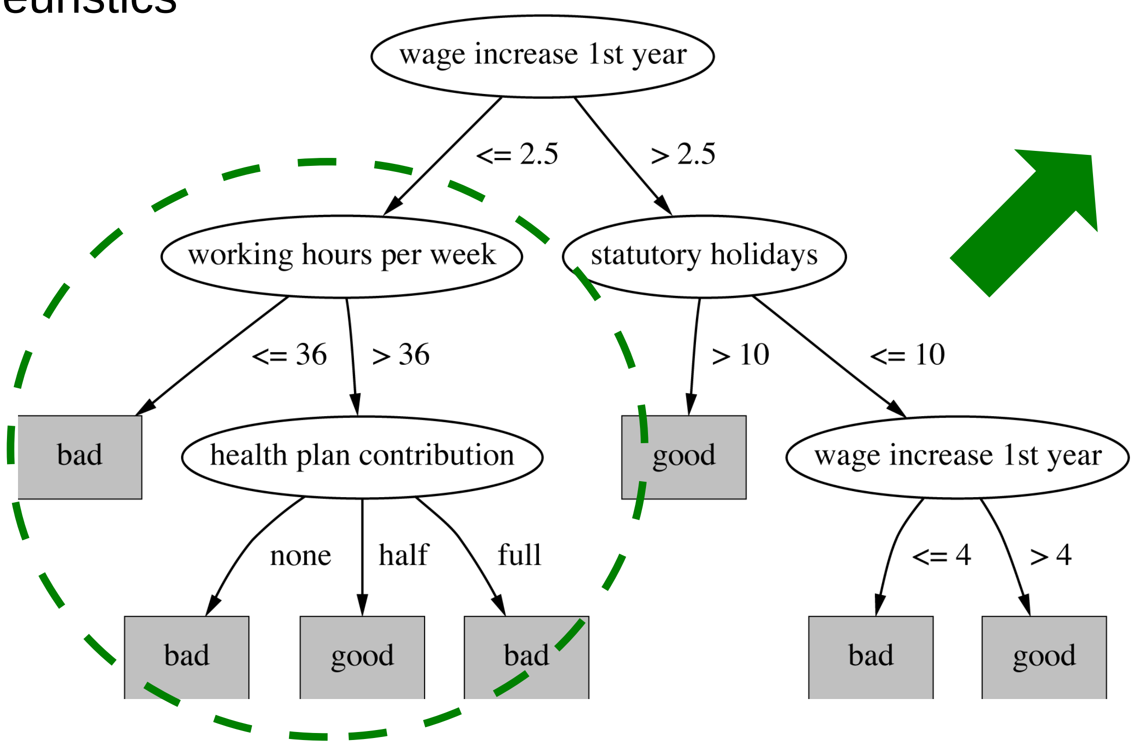
- Two subtree simplification operators
  - Subtree replacement
  - Subtree raising

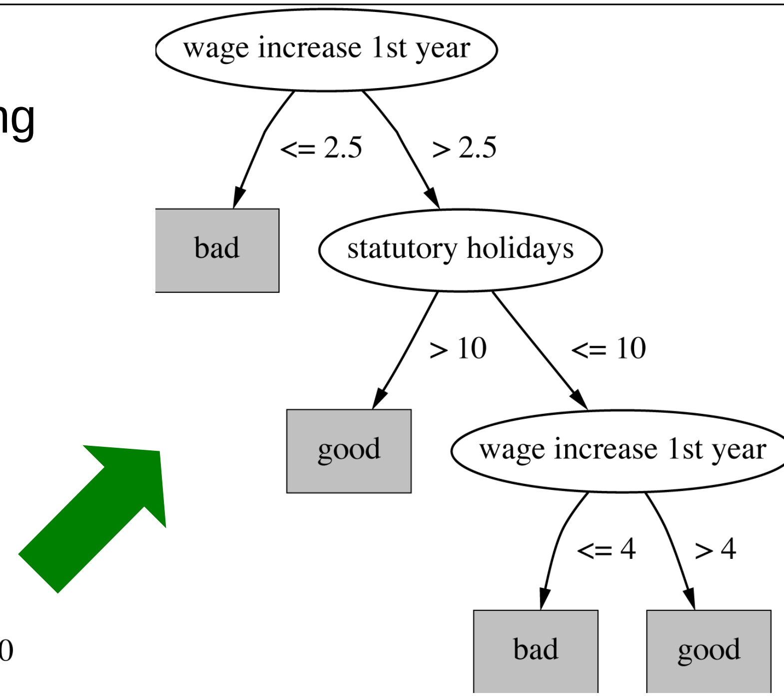
- Possible performance evaluation strategies
  - error estimation
    - on separate pruning set („reduced error pruning“)
    - with confidence intervals (C4.5's method)
  - significance testing
  - MDL principle

Subtree Replacement

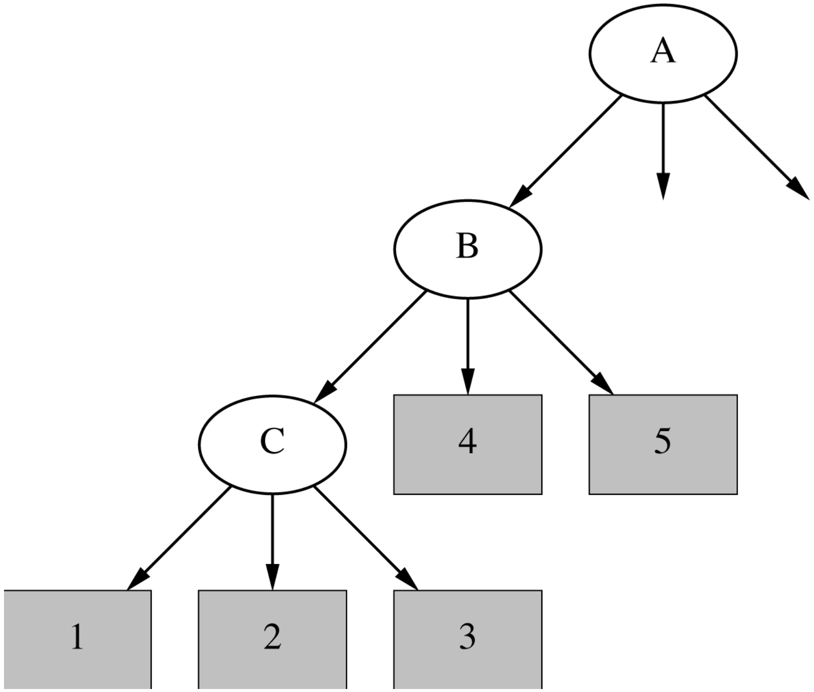
Proceeds Bottom-up:

- consider replacing a tree only after considering all its subtrees
- may make a difference for complexity-based heuristics

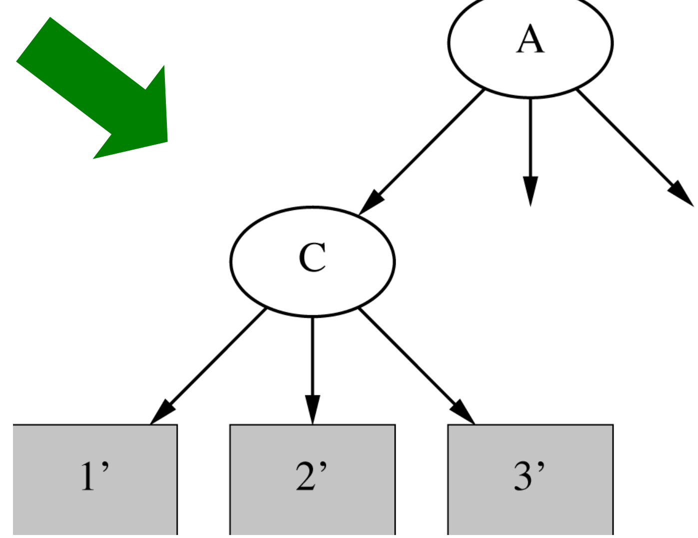




Subtree Raising



- Delete node B
- Redistribute instances of leaves 4 and 5 into C



Estimating Error Rates

- Prune only if it does not increase the estimated error
  - Error on the training data is NOT a useful estimator (would result in almost no pruning)
- Reduced Error Pruning
  - Use hold-out set for pruning
  - Essentially the same as in rule learning
    - only pruning operators differ (subtree replacement)
- C4.5's method
  - Derive confidence interval from training data
    - with a user-provided confidence level
  - Assume that the true error is on the upper bound of this confidence interval (pessimistic error estimate)

Reduced Error Pruning

Basic Idea

- optimize the accuracy of a decision tree on a separate pruning set

1. split training data into a growing and a pruning set
2. learn a complete and consistent decision tree that classifies all examples in the growing set correctly
3. as long as the error on the pruning set does not increase
   - try to replace each node by a leaf (predicting the majority class)
   - evaluate the resulting (sub-)tree on the pruning set
   - make the replacement that results in the maximum error reduction
4. return the resulting decision tree

Regression Trees

Differences to Decision Trees (Classification Trees)

- Leaf Nodes:
  - Predict the average value of all instances in this leaf
- Splitting criterion:
  - Minimize the variance of the values in each subset S_i
  - Standard deviation reduction

$$SDR(A,S) = SD(S) - \sum_i \frac{|S_i|}{|S|} SD(S_i)$$

- Termination criteria:

  Very important! (otherwise only single points in each leaf)

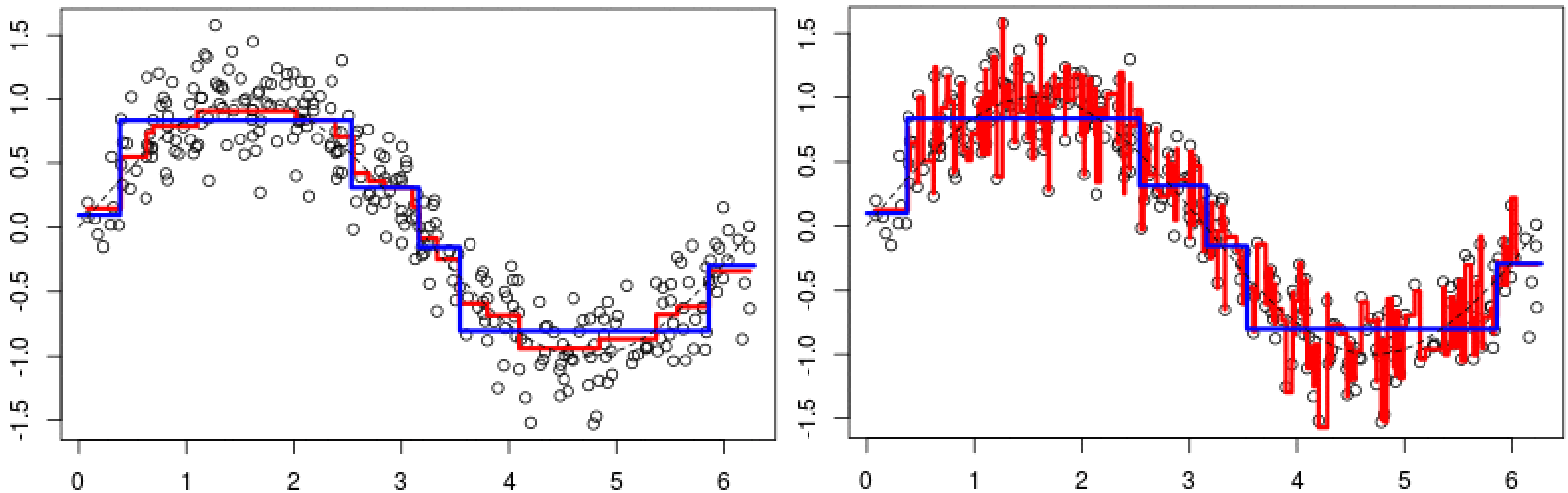
  - lower bound on standard deviation in a node
  - lower bound on number of examples in a node
- Pruning criterion:
  - size of tree, or numeric error measures, e.g. mean-squared error

Regression Trees

Differences to Decision Trees (Classification Trees)

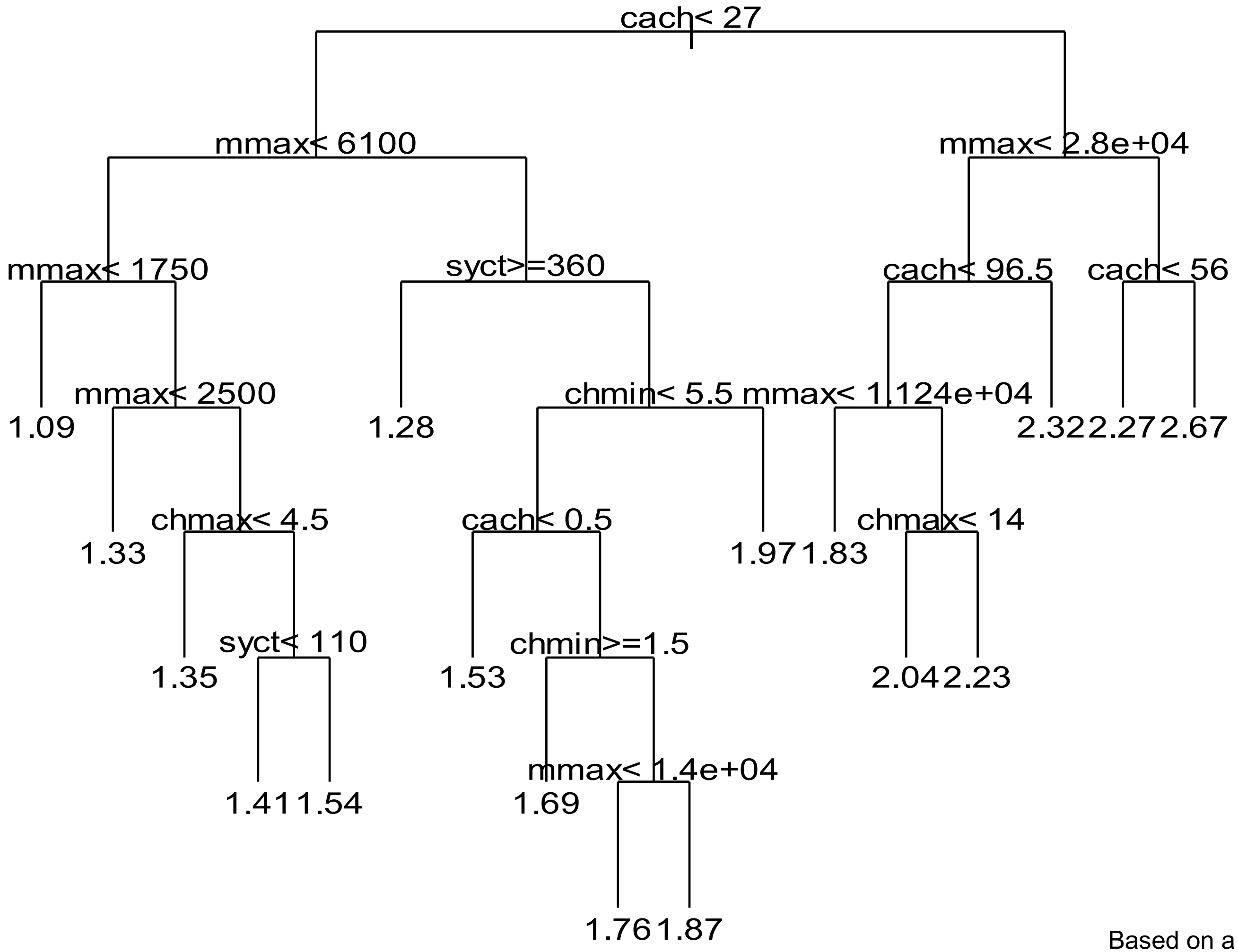
- Leaf Nodes:
  - Predict the average value of all instances in this leaf
- Splitting criterion:
  - Minimize the variance of the values in each subset S_i
  - Standard deviation reduction

$$SDR(A,S)=SD(S)-\sum_i \frac{|S_i|}{|S|} SD(S_i)$$



https://www.r-bloggers.com/an-attempt-to-understand-boosting-algorithms/

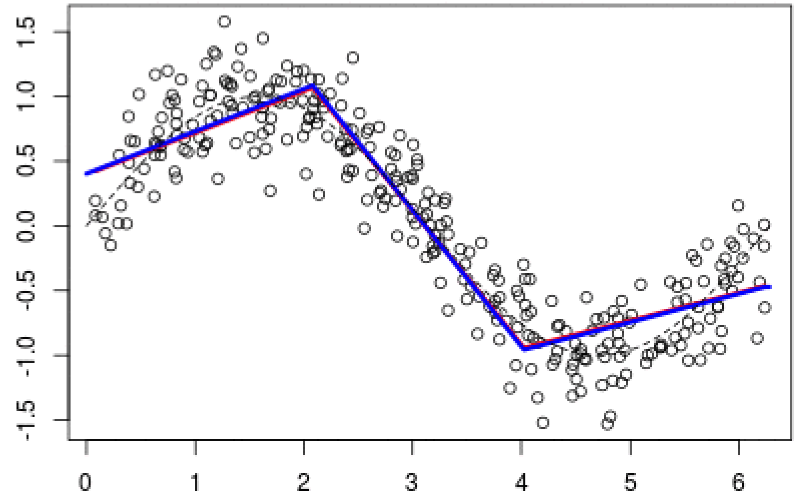
Regression Tree

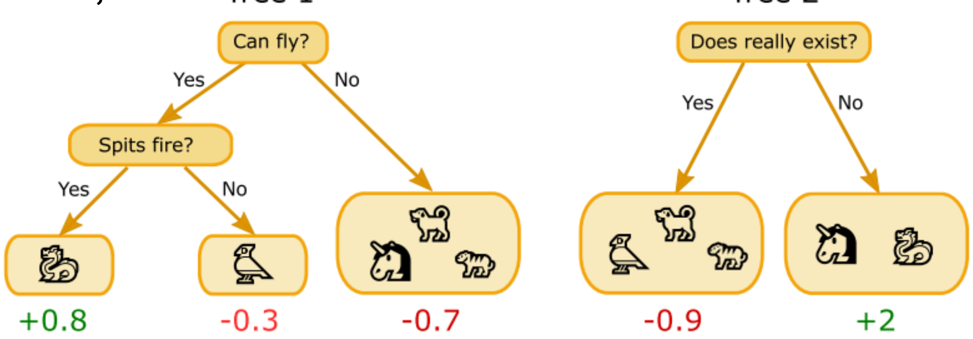


Based on a slide by Richard Lawton

Other Trees

- Model Trees
  - use a linear model for making the predictions
  - growing of the tree is as with Regression Trees
  - mapping function gets piecewise linear
- Gradient Boosting trees
  - ensemble of trees, where each subsequent tree corrects previous predictions (→ Boosting)
  - can use aleatory losses like MSE, logistic loss, ...
    - splits determined by gain on gradient statistics
  - regularization via tree size and model parameters in objective function





f( ) = 0.8 + 2 = 2.8 f( ) = -0.7 - 0.9 = -1.6

$$L(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_c)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2$$

https://xgboost.readthedocs.io/en/latest/tutorials/model.html
Simon Peter Bohlender, Dynamic Multilabel Classification using Gradient Boosted Trees

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Classification Problems require the prediction of a discrete target value
  - can be solved using decision tree learning
  - iteratively select the best attribute and split up the values according to this attribute
- Regression Problems require the prediction of a numerical target value
  - can be solved with regression trees and model trees
  - difference is in the models that are used at the leafs
  - are grown like decision trees, but with different splitting criteria
- Some Advantages of decision trees
  - non-linearity → fast predictors
  - natural support for categorical data
  - interpretability: comprehensible by humans
  - robustness: due to good heuristics and by using ensembles
- Overfitting is a serious problem!
  - simpler, seemingly less accurate trees are often preferable
  - evaluation has to be done on separate test sets

Tools

- Online tool for exercising: http://www.aispace.org/exercises/exercise7-a-1.shtml