Machine Learning in a Nutshell

15-488 Spring '20

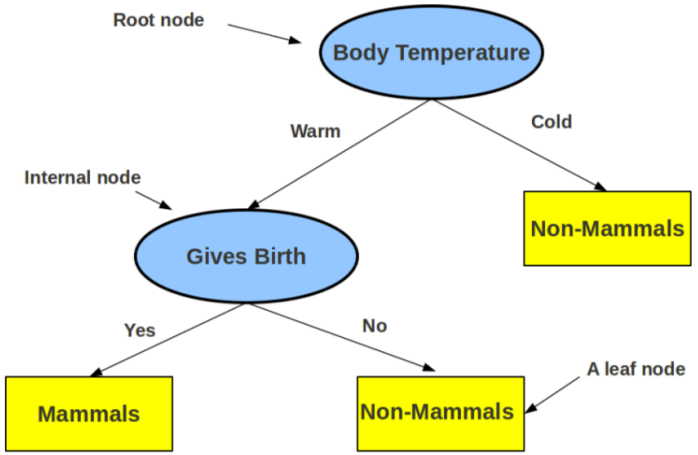
Lecture 26:
Decision Trees 3

Teacher:
Gianni A. Di Caro

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Decision Trees: Summary

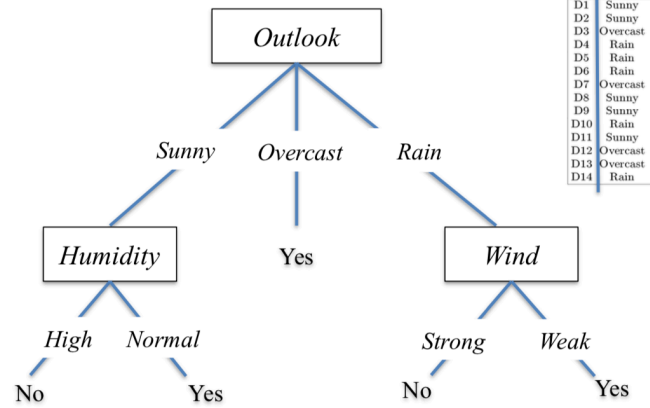
- A **decision tree** represents a function $f: X \rightarrow Y$, where X = attribute set, Y =decision set
- Tree Structure: Consists of a root node, internal nodes, and leaf nodes
  - ✓ **Root and internal nodes**: contain tests on attribute values
  - ✓ **Branches:** assign attribute values
  - ✓ **Leaf nodes**: define the predictions
- **Predict** input $\boldsymbol{x}$: traverse the tree from root to leaf, output value at leaf



- Build a tree:
  - Manually, from prior knoledge
  - ✓ Learning from data

f: <Outlook, Temperature, Humidity, Wind> → PlayTennis?

Day	Outlook	Temperature	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	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	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



➢ Finding the smallest consistent decision tree for given training dataset **is NP-Hard**

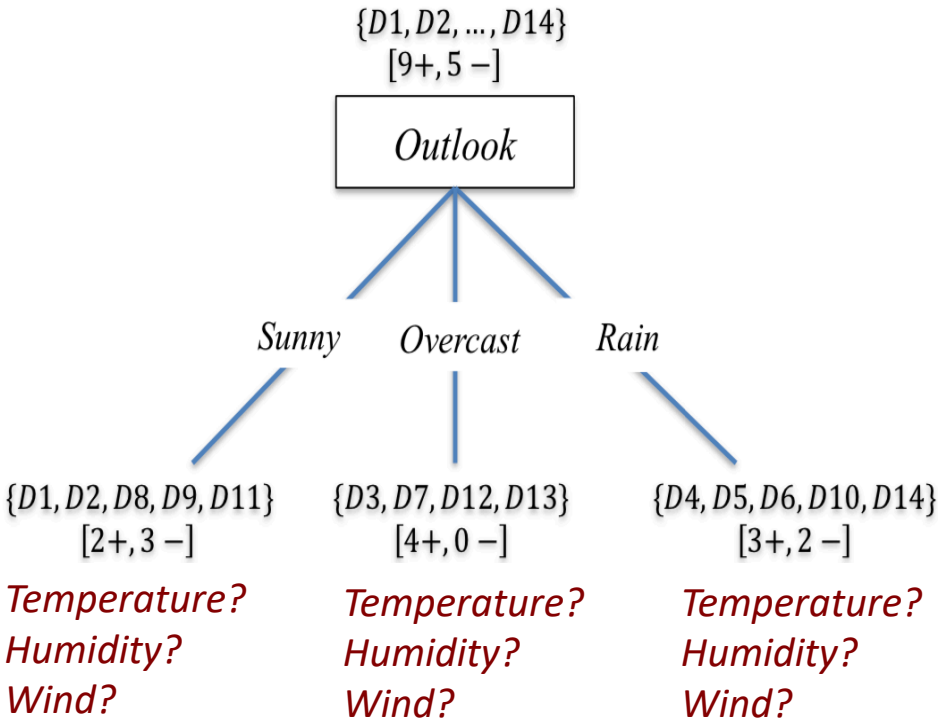
Decision Trees: Summary

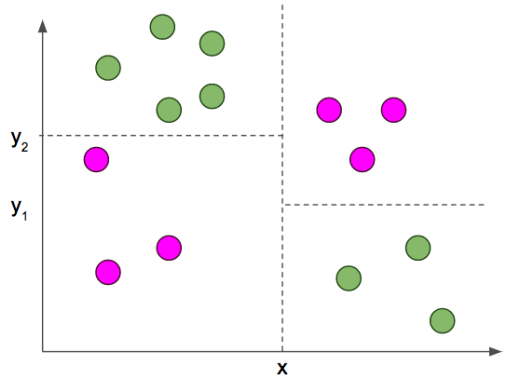
❖ Heuristics: **Top-down** tree construction, **greedy** stepwise selection of **attribute to split data**

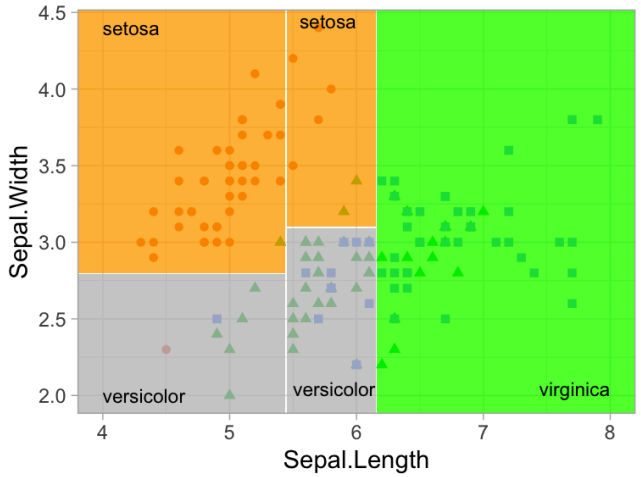
- Iteratively identify **partitions of the data set**, where inside a partition decisions are **consistent** with the training data → *Partitions are (possibly) homogeneous with respect to the labels*

✓ Popular: ID3, C4.5, CART

➢ **Key problem**: how to locally select the *best* attribute to split the data

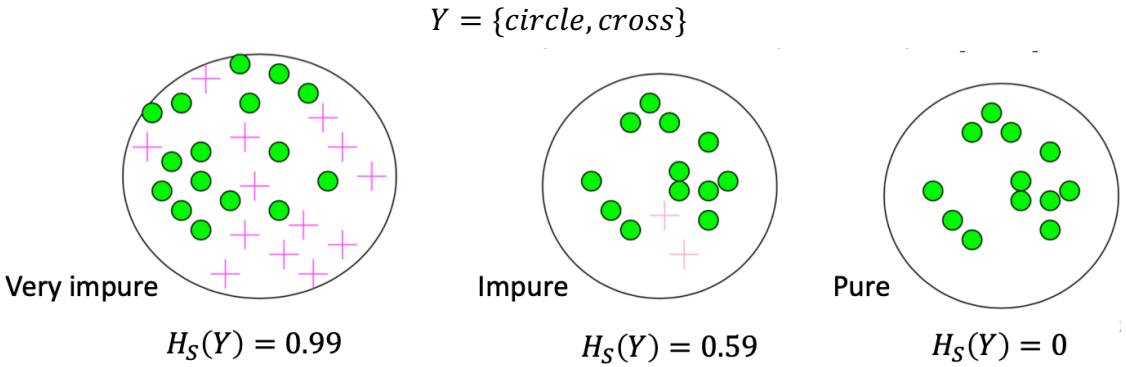






Decision Trees: Summary

- Classification: Greedy selection based on **purity** of generated partitions of the dataset (in terms of labels, Y)



- **Entropy** $H_S(Y)$ as measure of **purity** of a set S of labels Y

$$H_S(Y) = -\sum_{c=1}^{\#classes} P(Y = c) \ln P(Y = c)$$

$$P(Y = c) = \frac{\#examples\ of\ class\ c}{\#examples}$$

- **Information Gain**: Difference between $H_S(Y)$ and weighted average entropy of partitions in the data split generated by selecting attribute A; weights are proportional to the number of examples in each partition

$$Gain(S, A) = H_S(Y) - H_S(Y \mid A) \quad \rightarrow \quad Gain(S, A) = Entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- **ID3 Greedy selection**: The attribute A with the **largest Information Gain**

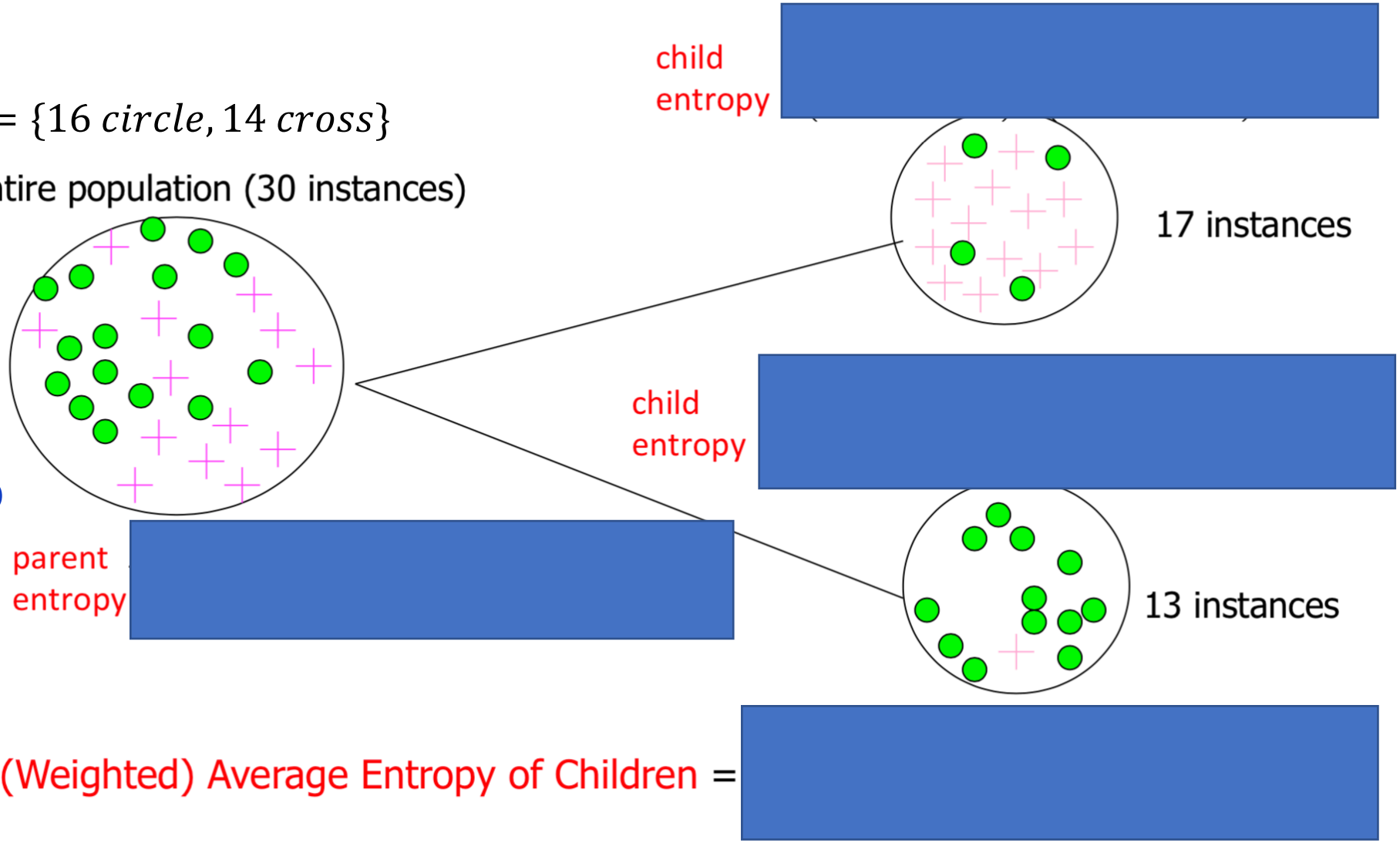
From entropy to information gain

Quiz: Compute Gains.

1. Parent H?
2. Child (up) H?
3. Child (down) H?
4. Information gain?

Information Gain = entropy(parent) – [average entropy(children)]

$S = \{16\ circle, 14\ cross\}$



$$H_S(Y) = -\sum_{c=1}^{\#classes} P(Y=c)\ln P(Y=c)$$

$$P(Y=c) = \frac{\#examples\ of\ class\ c}{\#examples}$$

Information Gain= 0.996 - 0.615 = 0.38

From entropy to information gain

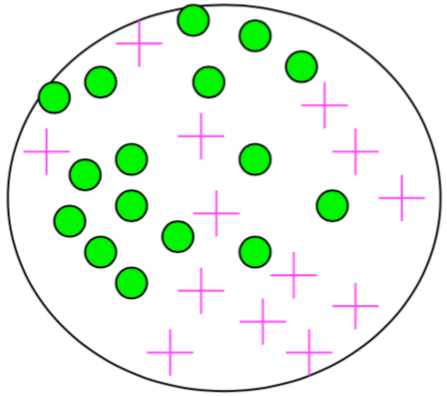
Quiz: Compute Gains.

1. Parent H?
2. Child (up) H?
3. Child (down) H?
4. Information gain?

Information Gain = entropy(parent) – [average entropy(children)]

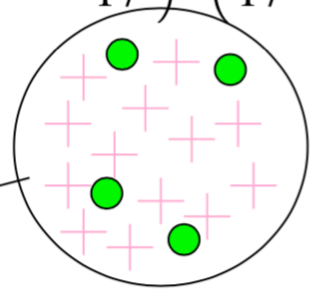
$S = \{16\ circle, 14\ cross\}$

Entire population (30 instances)



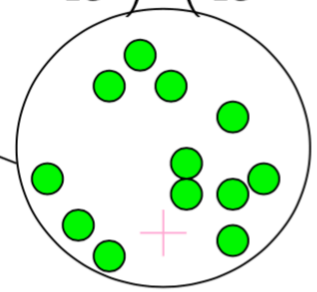
parent entropy $-\left(\frac{14}{30}\cdot\log_2\frac{14}{30}\right)-\left(\frac{16}{30}\cdot\log_2\frac{16}{30}\right)=0.996$

child entropy $-\left(\frac{13}{17}\cdot\log_2\frac{13}{17}\right)-\left(\frac{4}{17}\cdot\log_2\frac{4}{17}\right)=0.787$



17 instances

child entropy $-\left(\frac{1}{13}\cdot\log_2\frac{1}{13}\right)-\left(\frac{12}{13}\cdot\log_2\frac{12}{13}\right)=0.391$



13 instances

$$H_S(Y) = -\sum_{c=1}^{\#classes} P(Y=c)\ln P(Y=c)$$

$$P(Y=c) = \frac{\#examples\ of\ class\ c}{\#examples}$$

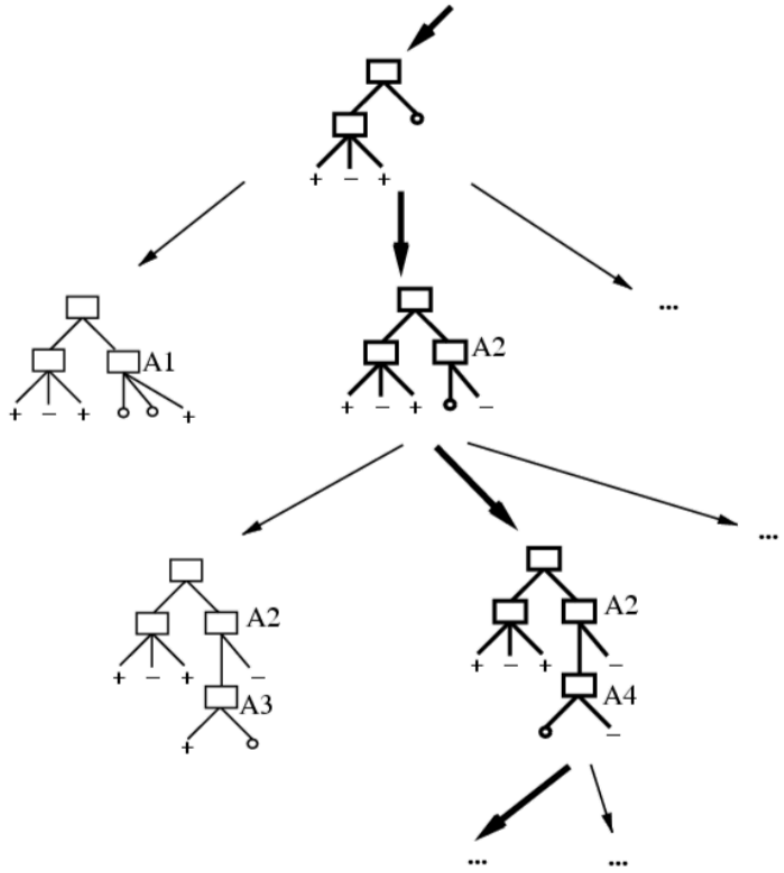
(Weighted) Average Entropy of Children = $\left(\frac{17}{30}\cdot 0.787\right)+\left(\frac{13}{30}\cdot 0.391\right)=0.615$

Information Gain= 0.996 - 0.615 = 0.38

What more about DTs?

- General **properties** of ID3?
- What about **overfitting / generalization**? → C4.5 improves ID3
- What about **continuous features**?
- What about **regression** tasks?
- What about other **measures of purity**?
- **Scikit-learn** methods?

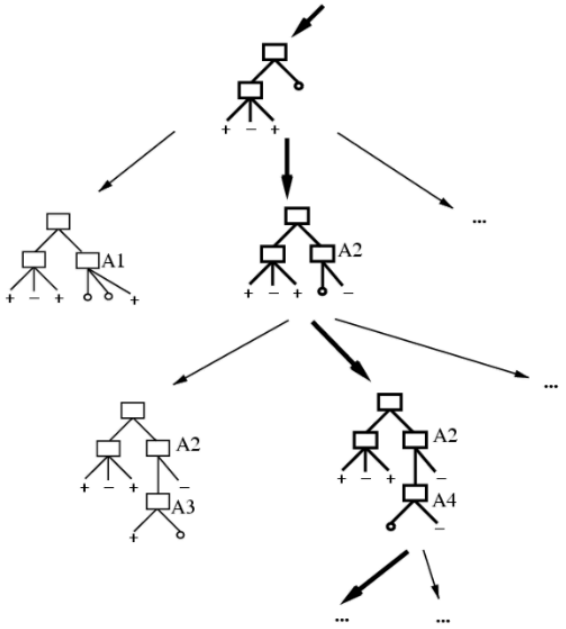
Properties of ID3



- ID3 performs heuristic search through space of decision trees
- It stops at smallest acceptable tree. Why?

Occam's razor: prefer the simplest hypothesis that fits the data

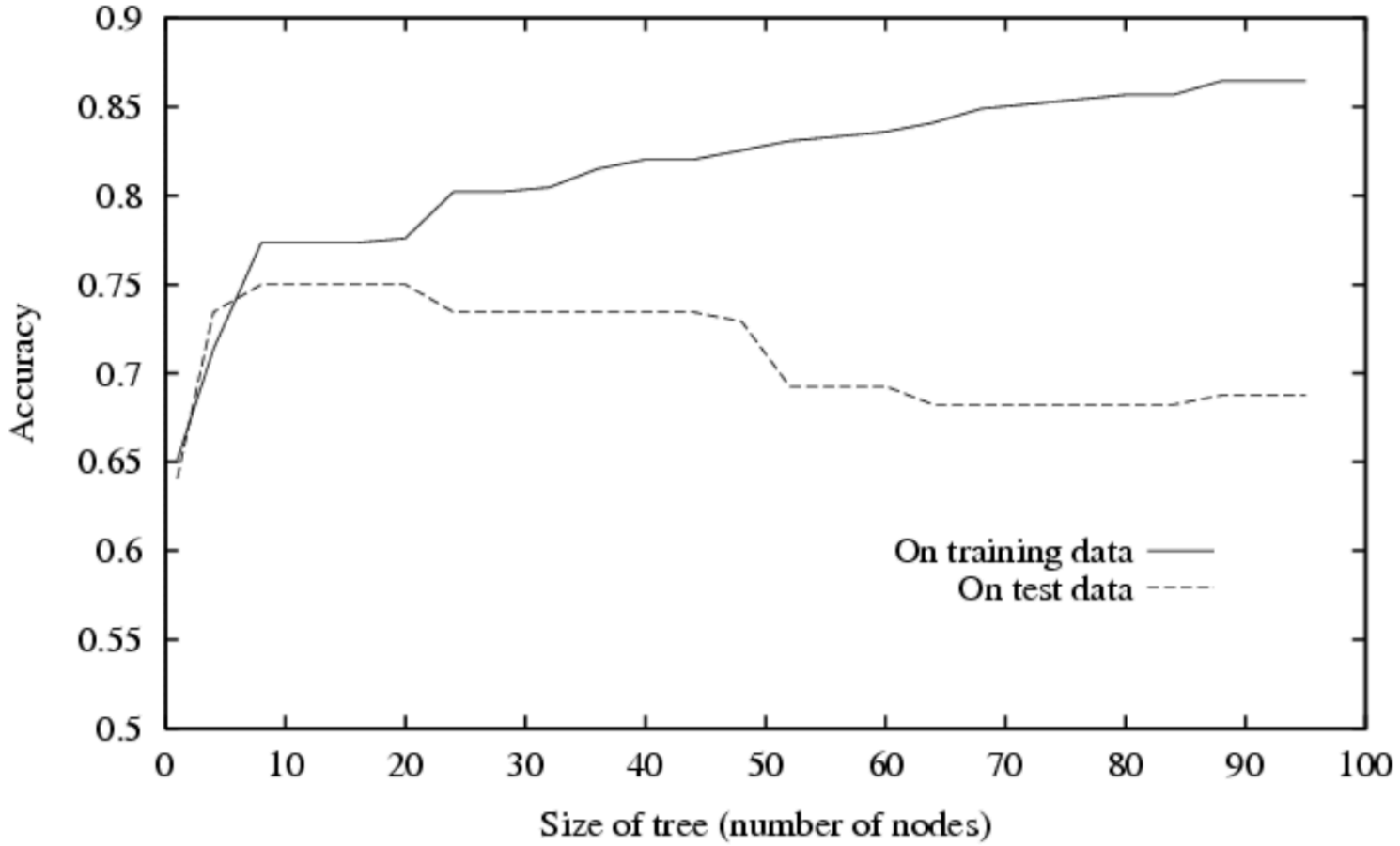
Properties of ID3



- ID3 performs heuristic search through space of decision trees

- It tends to have the right bias (output short decision trees), but it can still overfit.
- It might be beneficial to prune the tree by using a validation dataset.

Properties of ID3: Overfitting!



- Pruning by significance test
- Early stopping
- ….

Task: learning which medical patients have a form of diabetes.

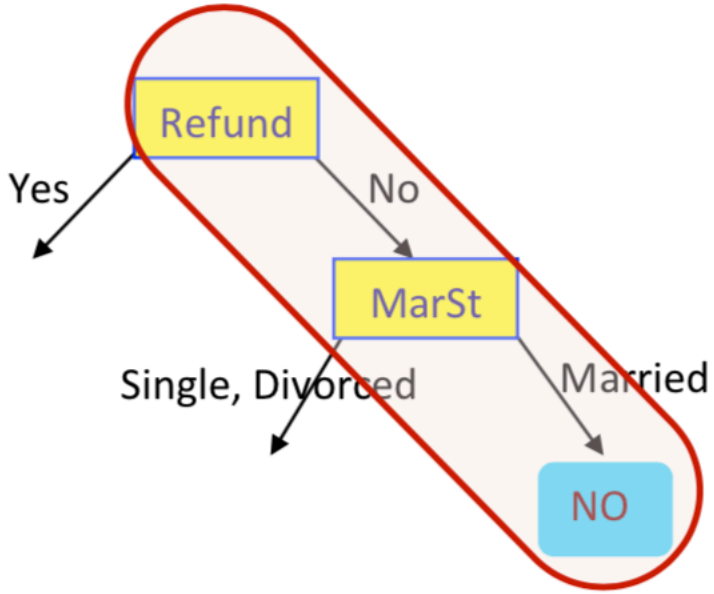
Overfitting: countermeasures (applies to all greedy approaches)

Two reasonable approaches:

- Optimize on the held-out (development) set
  - If growing the tree larger hurts performance, then stop growing
  - Requires a larger amount of data…
- Use statistical significance testing
  - Test if the improvement for any split it likely due to noise
  - If so, don't do the split!
  - Can also use this to prune the tree bottom-up

Overfitting: countermeasures (applies to all greedy approaches)

- Many strategies for picking simpler trees:
  - Pre-pruning
    - Fixed depth (e.g. ID3)
    - Fixed number of leaves
  - Post-pruning
    - Chi-square test
      - Convert decision tree to a set of rules
      - Eliminate variable values in rules which are independent of label (using chi-square test for independence)
      - Simplify rule set by eliminating unnecessary rules
  - Information Criteria: MDL(Minimum Description Length)



(Extras) Information Criteria (regularization approach)

- Penalize complex models by introducing cost

$$\hat{f} = \arg\min_T \left\{ \underbrace{\frac{1}{n}\sum_{i=1}^{n} \text{loss}(\hat{f}_T(X_i), Y_i)}_{\text{log likelihood}} + \underbrace{\text{pen}(T)}_{\text{cost}} \right\}$$

$$\begin{aligned} \text{loss}(\hat{f}_T(X_i), Y_i) &= (\hat{f}_T(X_i) - Y_i)^2 && \text{regression} \\ &= \mathbf{1}_{\hat{f}_T(X_i) \neq Y_i} && \text{classification} \end{aligned}$$

$\text{pen}(T) \propto |T|$ penalize trees with more leaves

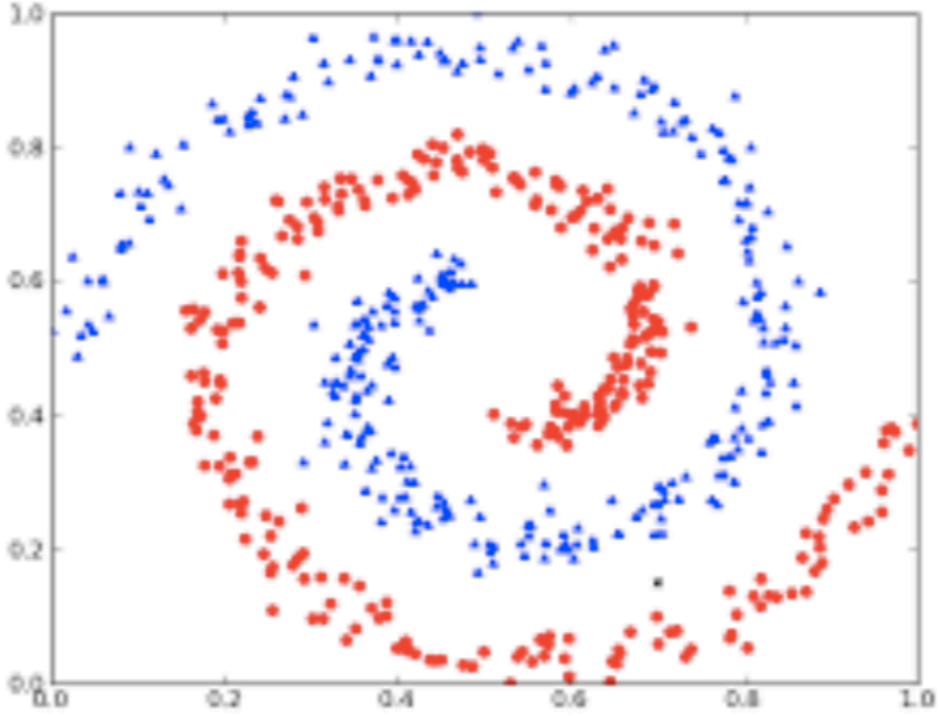
CART – optimization can be solved by dynamic programming

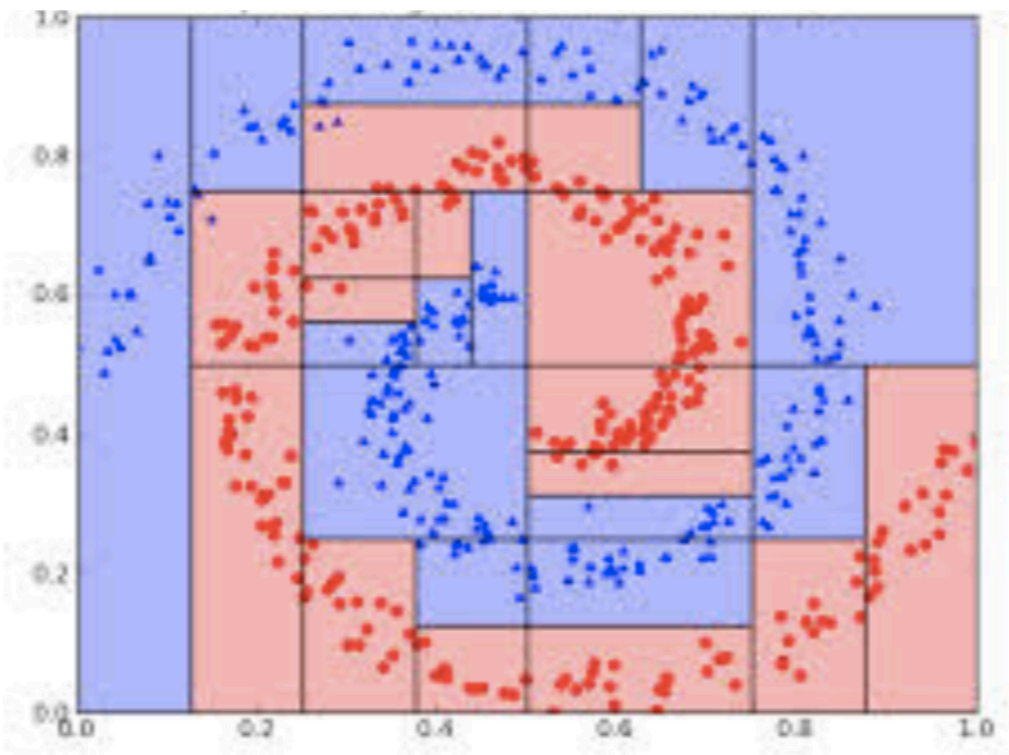
C4.5 (1992) improvement over ID3

Main loop:

1. $X \leftarrow$ the "best" decision feature for next *node*
2. Assign X as decision feature for *node*
3. For "best" split of X, create new descendants of *node*
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes
6. Prune back tree to reduce overfitting
7. Assign majority label to the leaf node

Example of a 2-feature DT classifier over complex data





Real-valued attributes

Acceleration (time for going from 0 to 100 Km/h) is a **real-valued attribute**

- Problem: Infinite number of possible split values ☹

$$acceleration \in [0, 50]$$

- ✓ Workaround: The dataset is necessarily ***finite***, such that **only a finite number of relevant splits matter** ☺

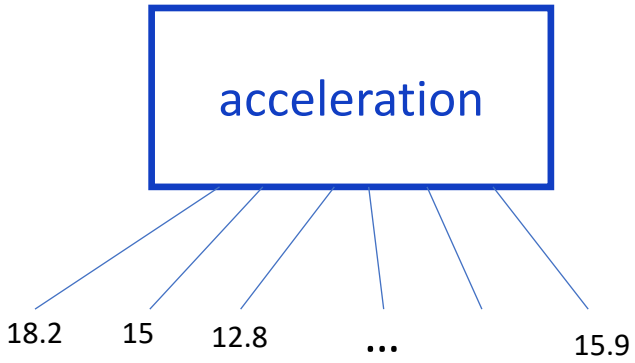
splits = $\{18.2, 15, 12.8, \ldots, 15.9\}$

mpg	cylinders	displacemen	horsepower	weight	acceleration	modelyear	maker
good	4	97	75	2265	18.2	77	asia
bad	6	199	90	2648	15	70	america
bad	4	121	110	2600	12.8	77	europe
bad	8	350	175	4100	13	73	america
bad	6	198	95	3102	16.5	74	america
bad	4	108	94	2379	16.5	73	asia
bad	4	113	95	2228	14	71	asia
bad	8	302	139	3570	12.8	78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
good	4	120	79	2625	18.6	82	america
bad	8	455	225	4425	10	70	america
good	4	107	86	2464	15.5	76	europe
bad	5	131	103	2830	15.9	78	europe

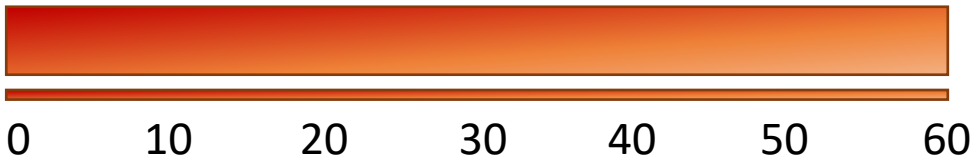
Real-valued attributes

- ✓ Workaround: The dataset is necessarily ***finite***, such that **only a finite number of relevant splits matter**

splits = {18.2, 15, 12.8, …, 15.9}



Split at acceleration will look like this



mpg	cylinders	displacemen	horsepower	weight	acceleration	modelyear	maker
good	4	97	75	2265	18.2	77	asia
bad	6	199	90	2648	15	70	america
bad	4	121	110	2600	12.8	77	europe
bad	8	350	175	4100	13	73	america
bad	6	198	95	3102	16.5	74	america
bad	4	108	94	2379	16.5	73	asia
bad	4	113	95	2228	14	71	asia
bad	8	302	139	3570	12.8	78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
good	4	120	79	2625	18.6	82	america
bad	8	455	225	4425	10	70	america
good	4	107	86	2464	15.5	76	europe
bad	5	131	103	2830	15.9	78	europe

Quiz: Is this fine?

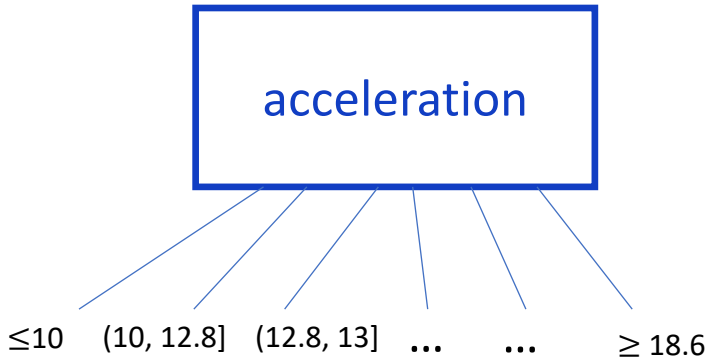
1. Yes, I don't see any problem
2. No, it wont' work (explain why)

It won't work: How do we answer a query for an input value of acceleration which is not the in the dataset?

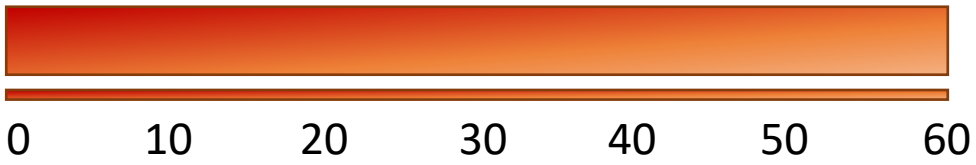
Real-valued attributes

- ✓ Workaround: Use **ranges** instead of single values?

splits = $\{\leq 10, (10, 12.8], (12.8, 13], \ldots, \geq 18.6\}$



Split at acceleration will look like this



mpg	cylinders	displacemen	horsepower	weight	acceleration	modelyear	maker
good	4	97	75	2265	18.2	77	asia
bad	6	199	90	2648	15	70	america
bad	4	121	110	2600	12.8	77	europe
bad	8	350	175	4100	13	73	america
bad	6	198	95	3102	16.5	74	america
bad	4	108	94	2379	16.5	73	asia
bad	4	113	95	2228	14	71	asia
bad	8	302	139	3570	12.8	78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
good	4	120	79	2625	18.6	82	america
bad	8	455	225	4425	10	70	america
good	4	107	86	2464	15.5	76	europe
bad	5	131	103	2830	15.9	78	europe

Quiz: Is this fine?

1. Yes, I don't see any problem
2. Yes, in principle but I see potential problems
3. No, it can't work (explain why)

It might work, but branching can be very large, making the tree grow a lot: computational issues + poor generalization

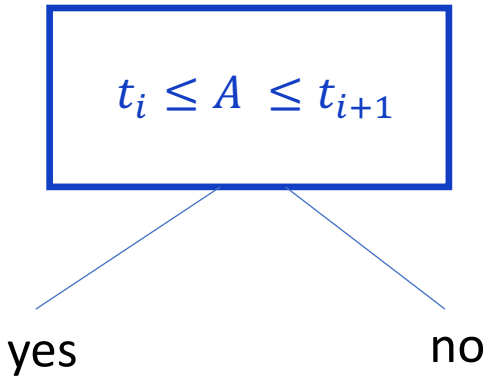
Real-valued attributes

✓ Workaround: Use **range attributes and binary splits (yes / no)**

splits = {Yes, No}

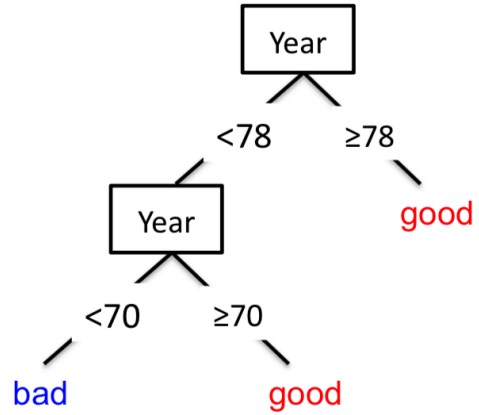
Continuous attribute, A, get transformed in n ranges:

A → $\{A < t_1, t_1 \leq A \leq t_2, \ldots, t_{n-1} \leq A \leq t_n\}$



Equivalently, we can use **open ranges**, $<$ or $\geq$

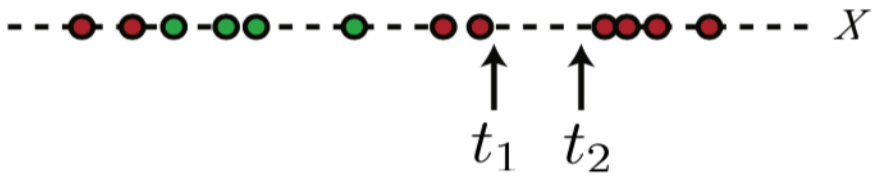
- **Binary tree:** split on attribute X at value t
  - One branch: X < t
  - Other branch: X ≥ t
- **Requires small change**
  - Allow repeated splits on same variable
  - How does this compare to "branch on each value" approach?



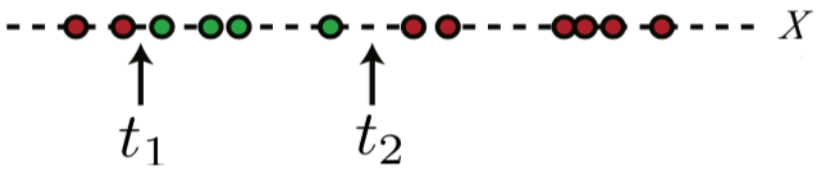
How do we define thresholds t?

Set of thresholds

- Binary tree, split on attribute X
  - One branch: $X < t$
  - Other branch: $X \geq t$
- Search through possible values of *t*
  - Seems hard!!!
- But only a finite number of *t*'s are important:



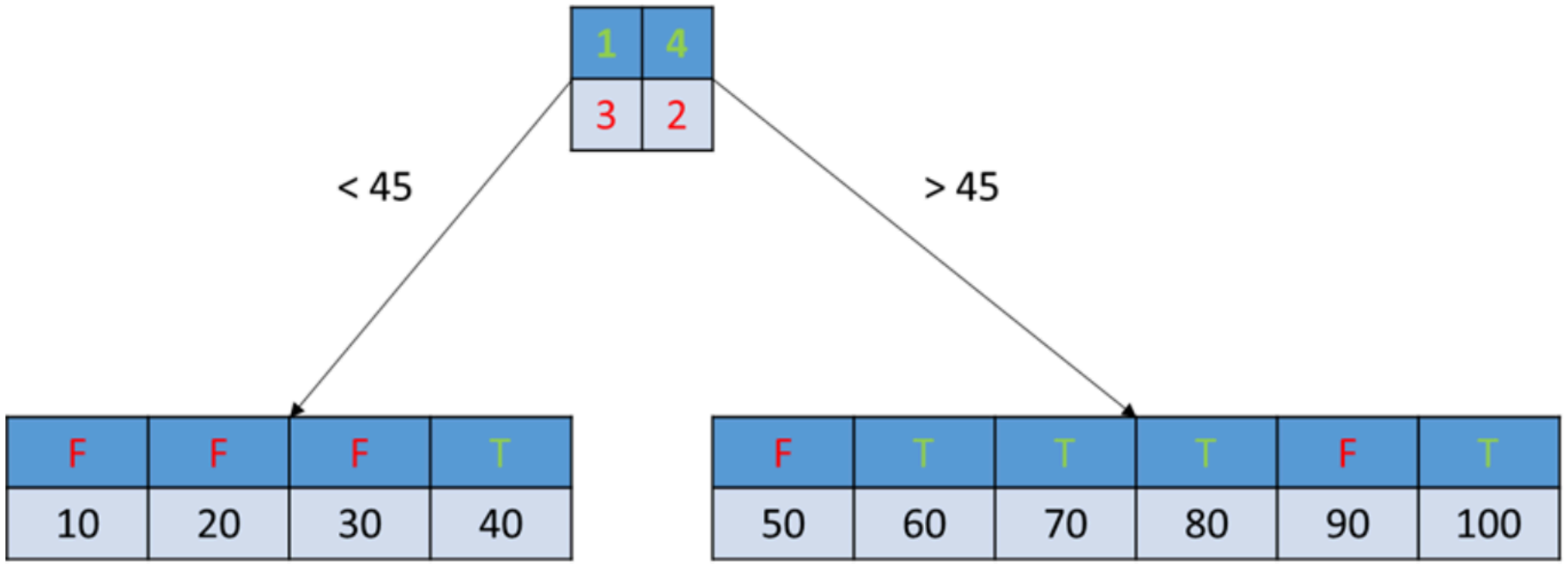
  - Sort data according to X into $\{x_1,\ldots,x_m\}$
  - Consider split points of the form $x_i + (x_{i+1} - x_i)/2$
  - Morever, only splits between examples of different classes matter!



(Figures from Stuart Russell)

Set of thresholds

T	F	F	F	T	F	T	F	T	T
40	10	30	90	80	50	60	20	70	100

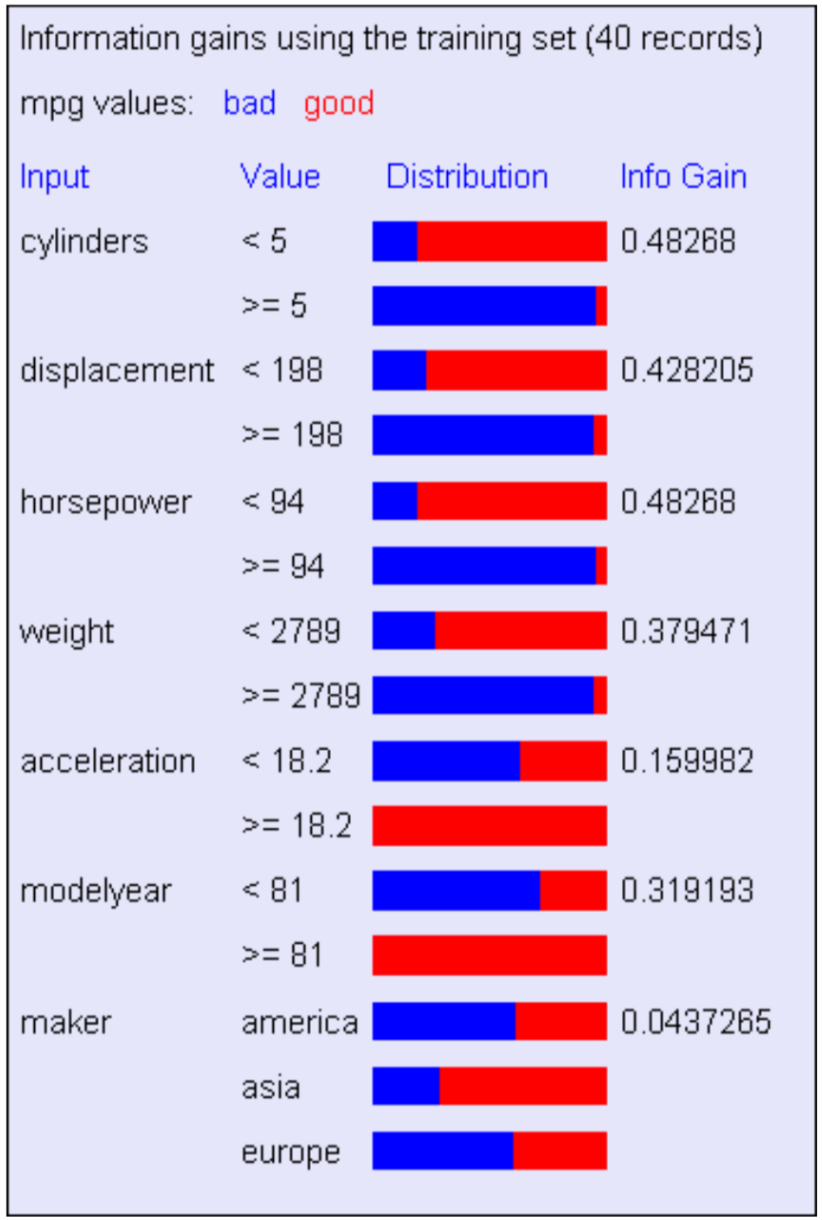


Picking the best threshold

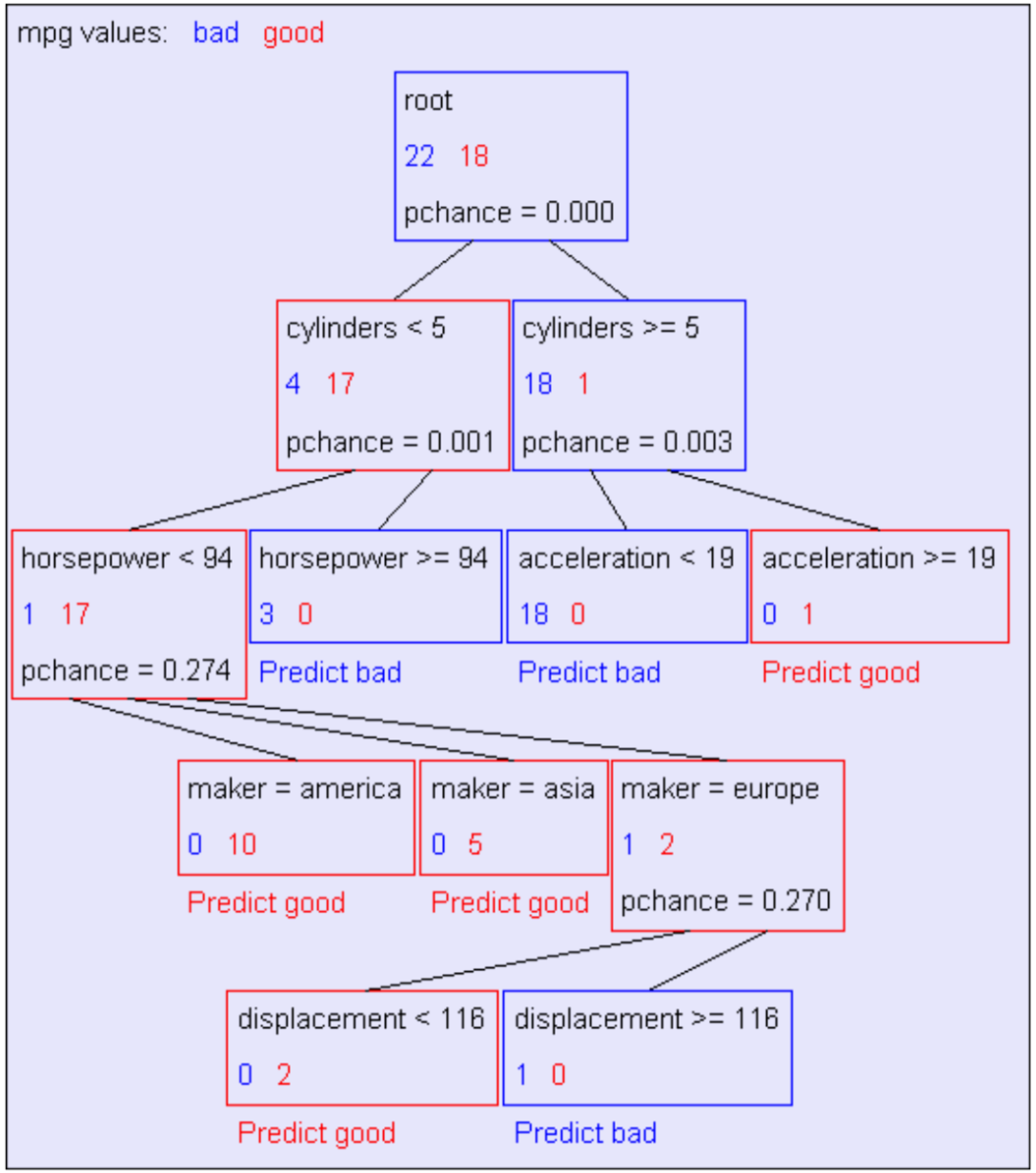
- Suppose *X* is real valued with threshold *t*
- Want **IG(Y | X:t)**, the information gain for Y when testing if *X* is greater than or less than *t*
- Define:
  - $H(Y|X:t) = p(X < t)\, H(Y|X < t) + p(X >= t)\, H(Y|X >= t)$
  - $IG(Y|X:t) = H(Y) - H(Y|X:t)$
  - $IG^*(Y|X) = \max_t IG(Y|X:t)$
- Use: $IG^*(Y|X)$ for continuous variables

Dataset with continuous attribute values

Example with MPG

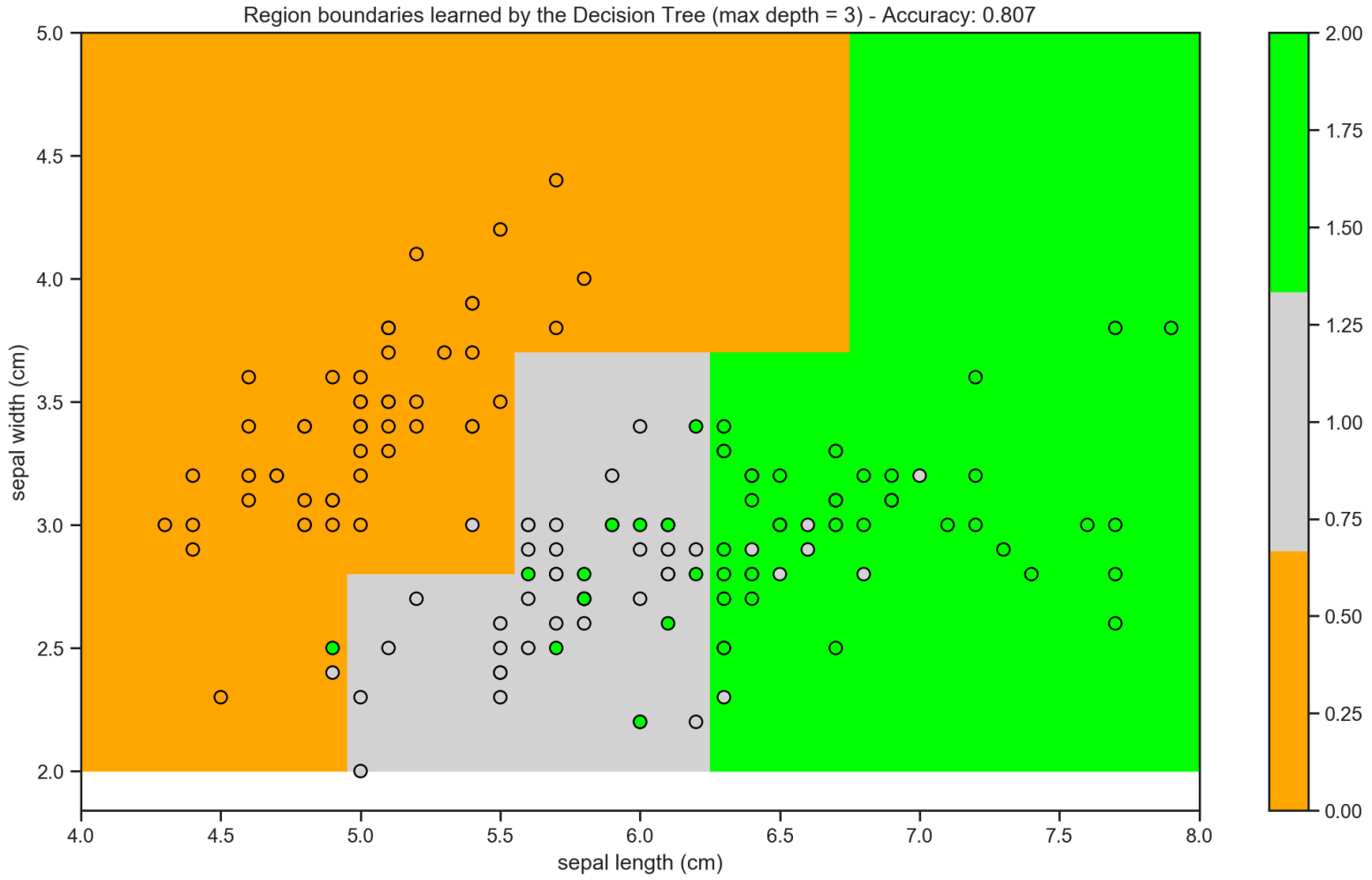


DT with continuous attributes

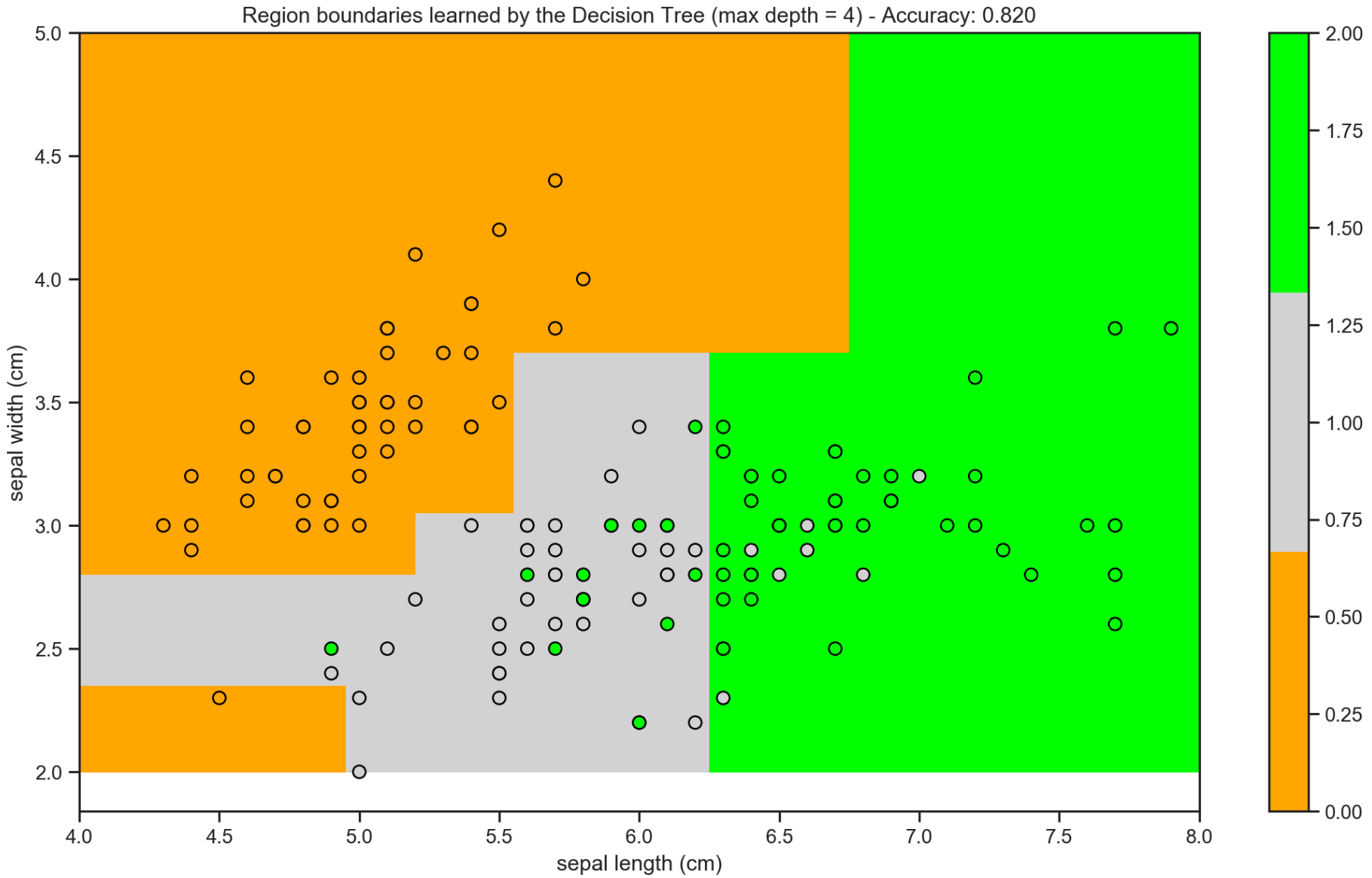


DT and Axis-aligned boundaries

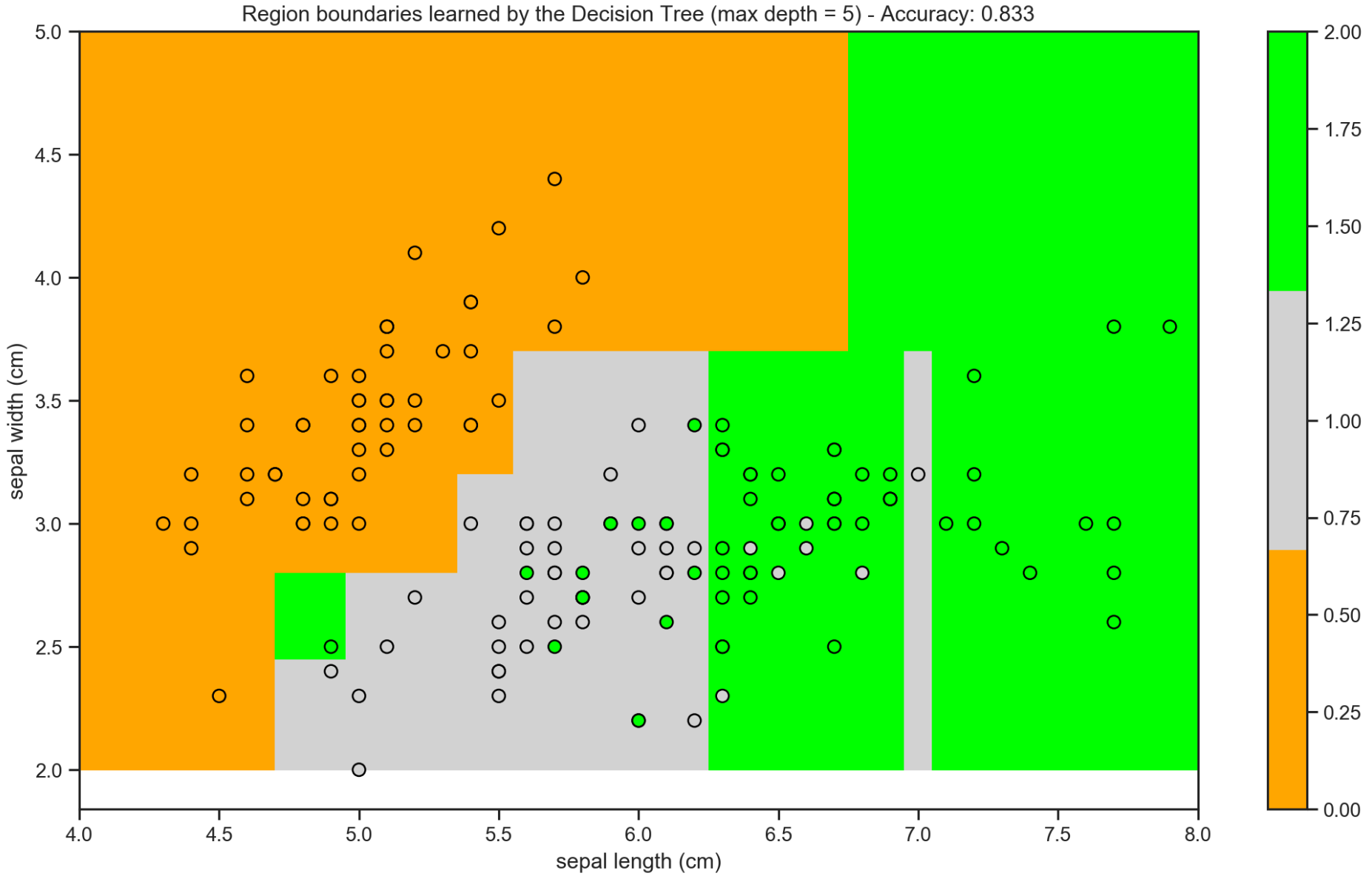
- Remember the Iris dataset? (lecture 9)
- Here we show the classification DT trained with only two features for the sake of visualization
- Changing the value the **max depth** constraint changes the amount of overfitting and the classification accuracy (on the training set)
- Note that **all features are continuous** in this case, and the $\geq$ and $\leq$ binary tests generate axis-parallel continuous boundaries
- Discrete features would generate a similar pattern but not continuous



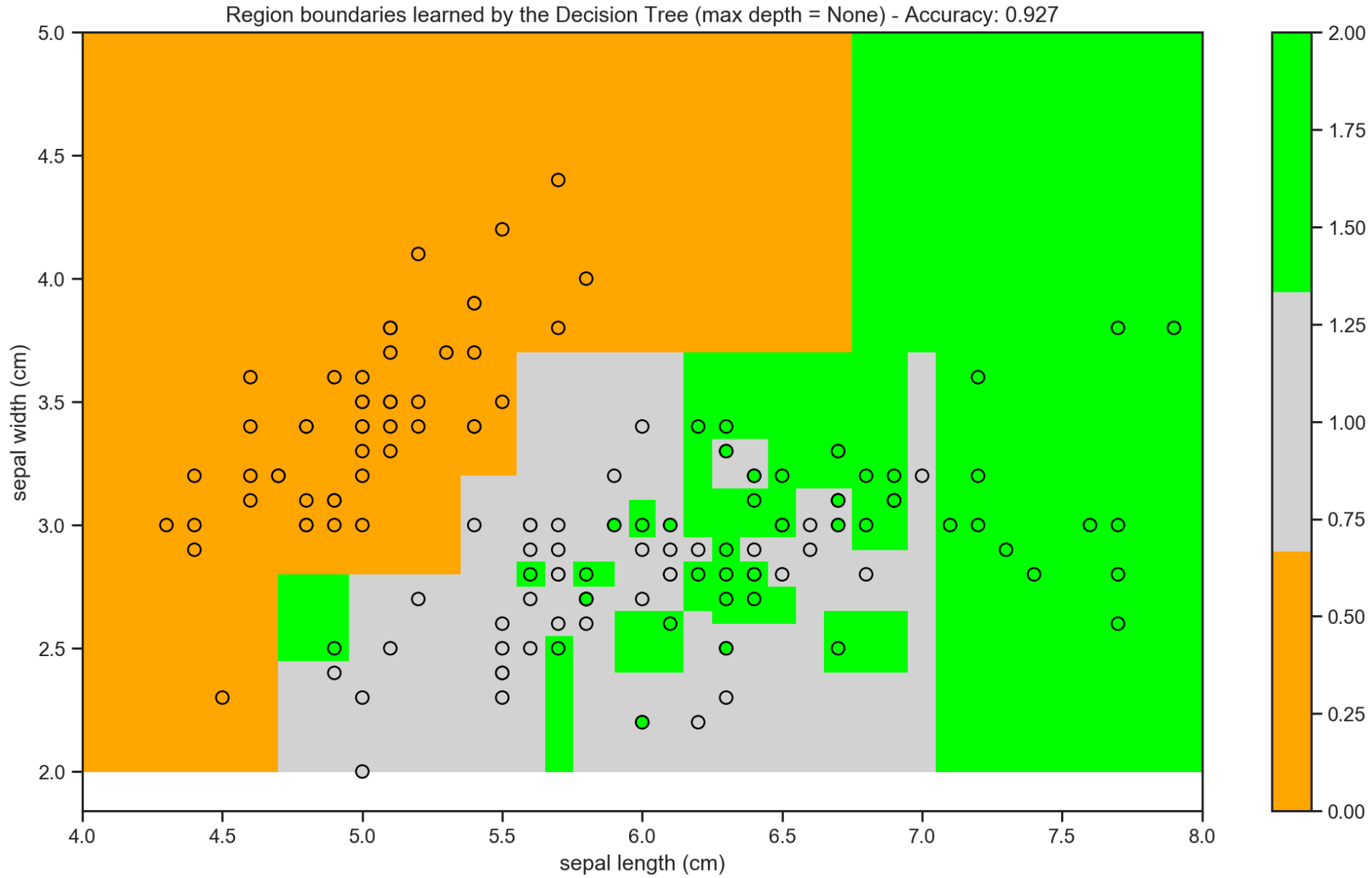
DT and Axis-aligned boundaries



DT and Axis-aligned boundaries

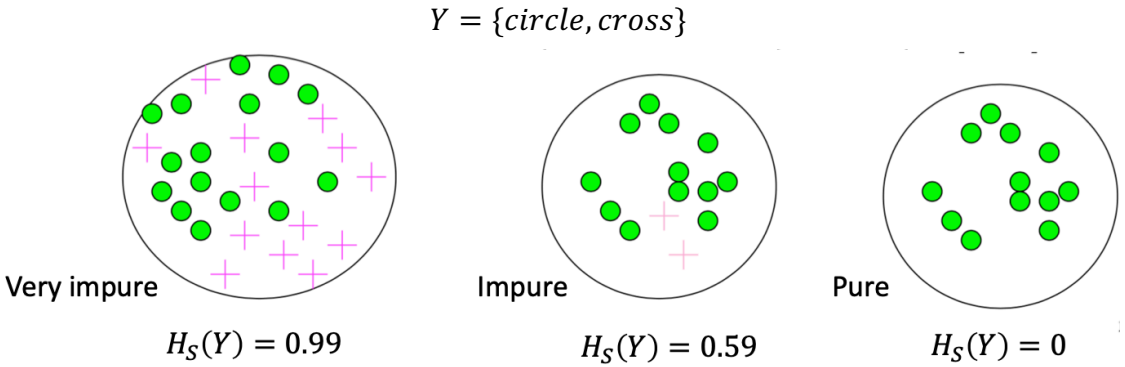


DT and Axis-aligned boundaries



DT for Regression Tasks: Explanatory notes

- The same as in classification, the only difference is in **how the selection of the best attribute is done**

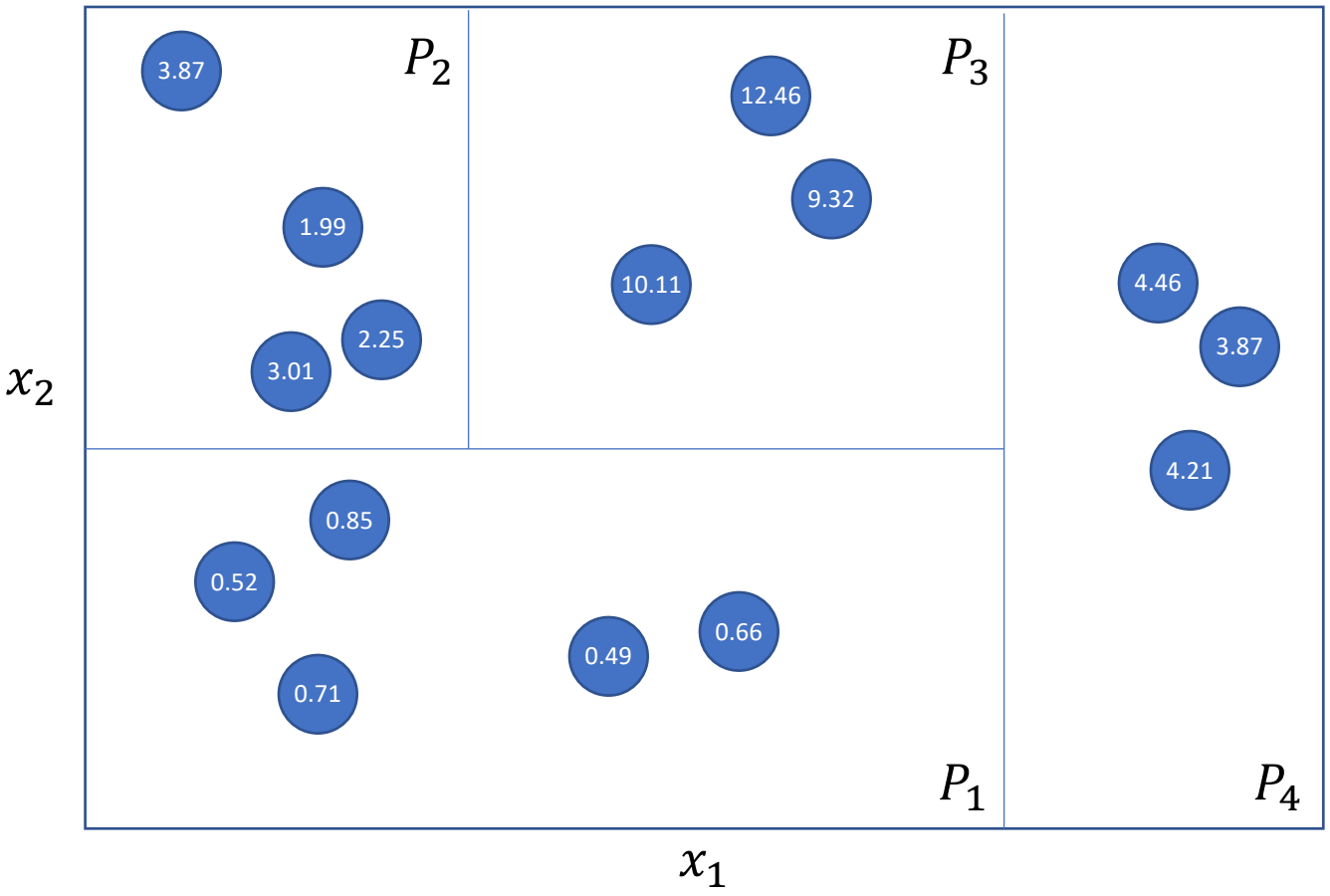


In classification, ***impurity*** of a partition indicates the certainty / reliability for issuing a decision out of a labeled dataset, and **entropy** is (commonly) used to measure the level of impurity

- Set purity is related to labels: a labeled set is *pur*e if all the entries in the set have the same label.
- If the elements in the set can take a discrete (a possibly small) number of labels, $Y = \{y_1, y_2, \cdots, y_n\}$, asserting the purity can be done by **counting the different labels and measuring their distribution**.

- In regression, since the *labels/values* take continuous values, $Y \in (y_{min}, y_{max})$, we can't really rely on counting the labels: most of the values/labels will be different from each other! But we can still reason on their distribution / spread over a continuous interval.
  - A set S whose elements have continuous labels / values, can be regarded as *pure* with respect to the labels/values when the labels are either **all the same** or **do not differ too much from each other**
- How do we quantify that *do not differ too much from each other*? → **Measures of dispersion** → *Variance*

DT for Regression Tasks: a numeric example

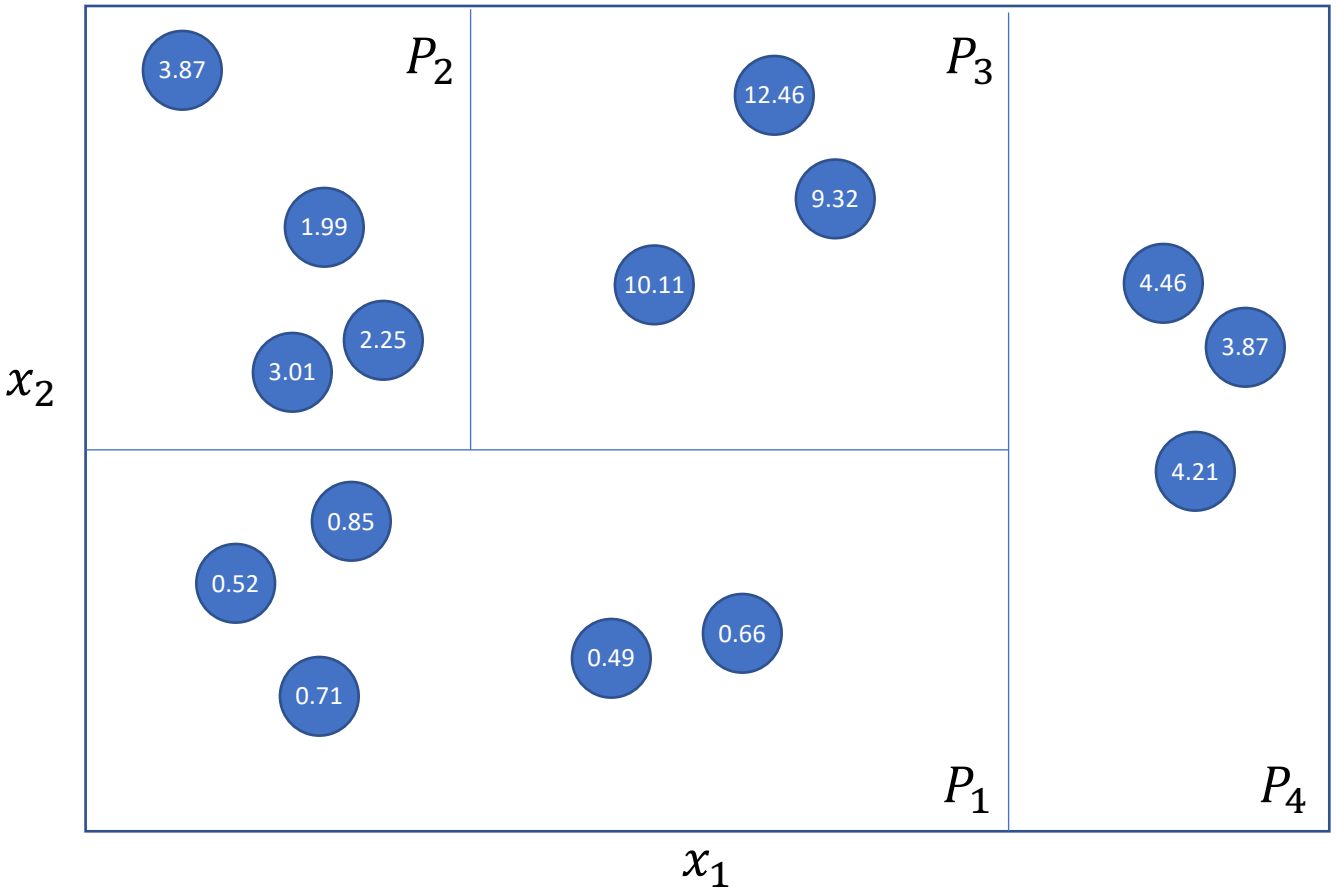


- **Regression problem**: $f: (x_1, x_2) \rightarrow Y \subseteq [0,15]$
- Circles represent **training data**
- Numbers in the circles are the **continuous labels/values of each example**
- The **axis-parallel boundaries** delimit decision regions learned by a regression decision tree
- Four partitions of the dataset have been learned, P_1, P_2, P_3, P_4
- **Each partition groups data whose labels are *homogeneous***, more or less: labels have *little spread* as measured by their **variance**

$$\mu(P_1) = \frac{(0.85 + 0.52 + 0.71 + 0.49 + 0.66)}{5} = 0.65$$

$$\sigma^2(P_1) = \frac{(\mu(P_1) - 0.85)^2 + (\mu(P_1) - 0.52)^2 + (\mu(P_1) - 0.71)^2 + (\mu(P_1) - 0.49)^2 + (\mu(P_1) - 0.66)^2}{4} = 0.017$$

DT for Regression Tasks: a numeric example



$\mu(P_1) = 0.65 \quad \sigma^2(P_1) = 0.017$

$\mu(P_2) = 2.78 \quad \sigma^2(P_2) = 0.536$

$\mu(P_3) = 10.63 \quad \sigma^2(P_3) = 1.778$

$\mu(P_4) = 4.18 \quad \sigma^2(P_4) = 0.058$

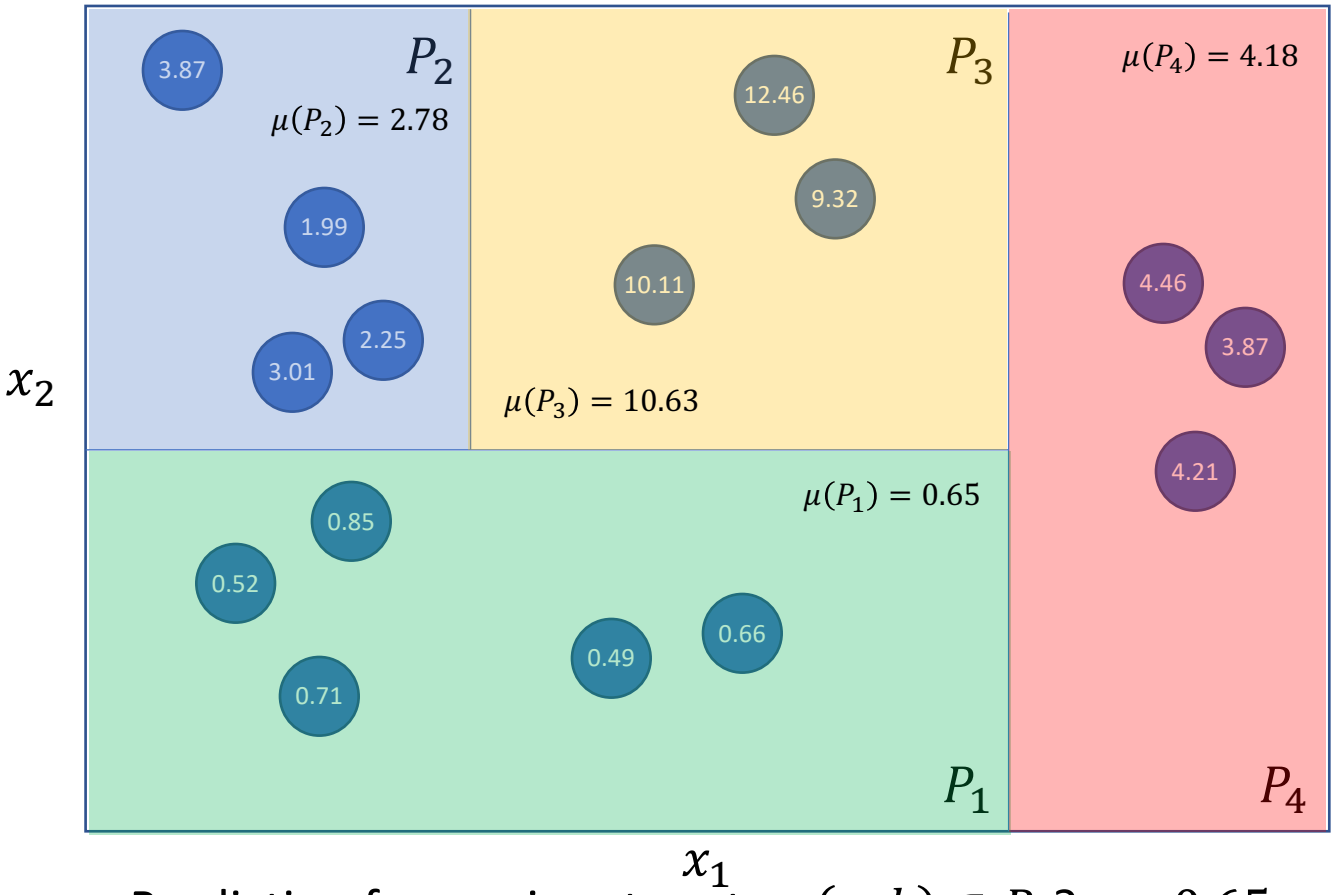
✓ **The variance in each partition is quite low**: labels are quite homogeneous about their mean

E.g., the partition $P_1 \cup \{10.11\}$ would result in: $\sigma^2\left(P_1 \bigcup \{10.11\}\right) = 12.45$

→ High variance, not a pure set in terms of continuous labels

Variance (or other measures of spread) are good measures of set purity for continuos labels

DT for Regression Tasks: a numeric example



- Prediction for *any* input vector $(a, b) \in P_1$? → 0.65
- Prediction for *any* input vector $(a, b) \in P_2$? → 2.78
- Prediction for *any* input vector $(a, b) \in P_3$? → 10.63
- Prediction for *any* input vector $(a, b) \in P_4$? → 4.18

✓ **How *predictions* are made in the decision tree?**

➢ The rationale guiding tree construction was to define decision boundaries based on partitioning training data in low-variance groups

✓ It is *sound* to use the **sample mean** of the training data in a region as predicted output

○ Because of *low variance,* the sample mean is a good representative of the set!

Note: Other summary statistics such as the **median** or the **mode** can be used for predictions

DT for Regression Tasks: Dispersion to measure label purity

❖ **DT for Regression, the algorithms:**

- ✓ Follow the same greedy top-down approaches described in the case of classification
- ➢ Use measures of **value dispersion** for the labels (e.g., **variance**) to measure the purity of a partition and in turn assert **quality / gain** of an attribute split
- ➢ Use the measured gain to guide the **greedy decisions** for selecting the best attribute for local splitting

In practice:

- *Replace Entropy* as a measure of set purity with a measure of dispersion for continuous values
- Select the attribute that generates partitions with **largest reduction in dispersion** (on average)

Reference measure of dispersion: Variance / Standard Deviation

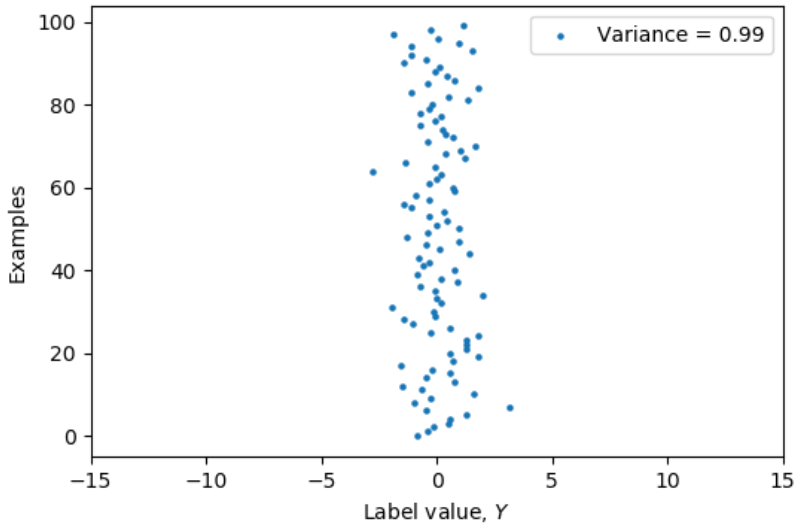
✓ **Variance** – Given a set Y of possible continuous labels, $Y \subseteq \mathbb{R}$, and given a set S of m labeled examples, $S = \{y_1, y_2, \cdots, y_n\}$, $y_i \in Y$, the sample variance $\hat{\sigma}^2(S)$, of S wrt the label values:

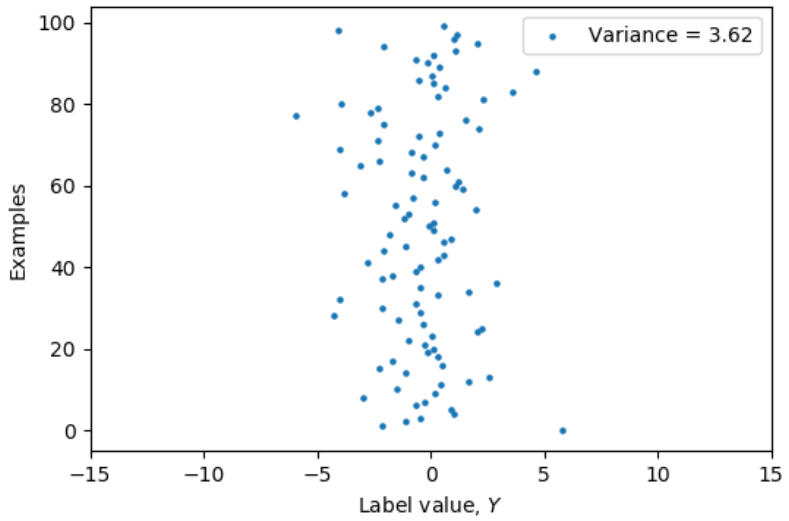
$$\hat{\sigma}^2 = \frac{1}{(n-1)} \sum_{i=1}^{n} (\bar{y}_S - y_i)^2 \qquad \bar{y}_S = \frac{1}{n} \sum_{i=1}^{n} y_i$$

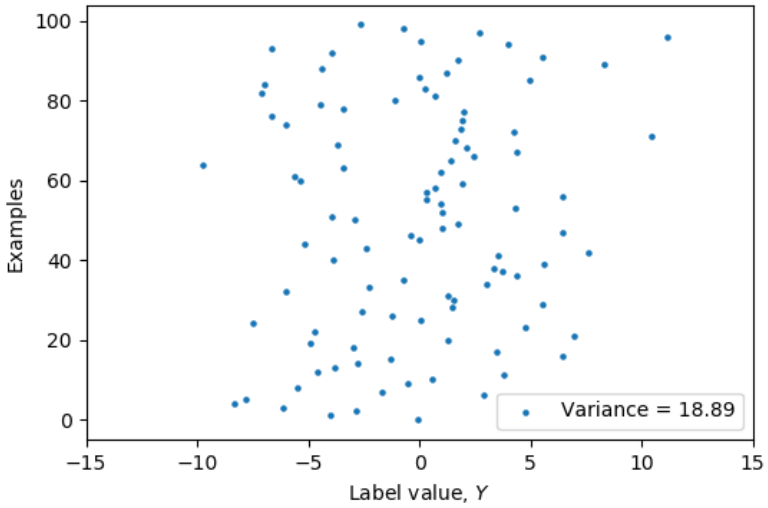
✓ **Standard Deviation** works equally well / better: $\hat{\sigma} = \sqrt{\hat{\sigma}^2}$

(linear in a scale transformation of the data)

❖ If all samples have the same label, $\hat{\sigma} = 0$







When to stop in the quest for purity?

- **When to stop splitting a node**? Is a partition ***pure enough*** or not?
  - Classification:
    - ✓ **If labels are all the same**, the partition is pure, decision will be consistent with the training data, no need to split further.
    - ❑ **If labels are mixed**, we can still decide to stop (e.g., to avoid the tree growing / overfitting) when the entropy of the set is below a certain threshold (e.g., since we know that for n possible labels, entropy ranges from 0 to $-\ln\frac{1}{n}$ we can set the threshold to some fraction of the maximum value, or above the minimum)

When to stop in the quest for purity?

- **When to stop splitting a node**? Is a partition ***pure enough*** or not?

  - Regression:

    - **Labels won't be all the same** (in general), since they are continuous values!

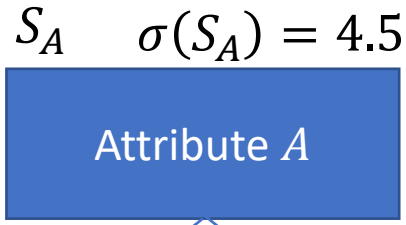
    - We *must* set a threshold on the value of the used measure of purity (e.g., standard deviation, σ) to decide if / when to declare a node a leaf.

    - ✓ For standard deviation, since σ's value depends on the scale of the labels in the set S, the **coefficient of variation** of the set is often used as an indicator for determining a threshold about splitting further or not:

$$CoV = \frac{\sigma(S)}{|\mu(S)|} \times 100\%$$

Note that $\sigma(S)$ and $\mu(S)$ have the same *units*

When to stop in the quest for purity?



$S_A^T \subseteq S_A$ $S_A^F \subseteq S_A$ $S_A^T \cup S_A^F = S_A$

$\sigma(S_A^T) = 0.15$ $\sigma(S_A^F) = 1.5$

Is this a leaf?
Should be stop here?
Is the set S_A^T *pure* enough?

It doesn't look as a leaf, maybe

$\mu(S_A^F) = 4.02$ → CoV = 37%

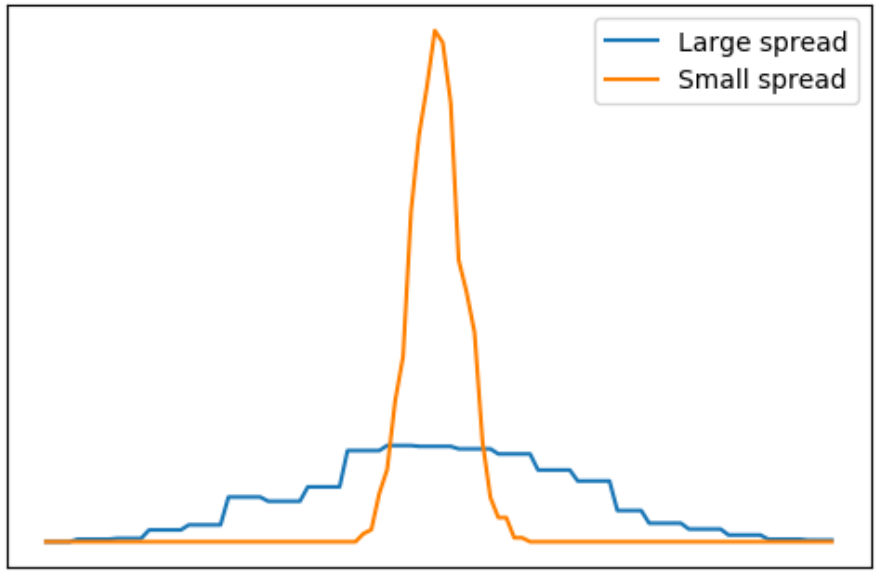
$\mu(S_A^T) = 1.65$ → CoV = 9%

This will be expanded

E.g., if the threshold was set to 10%, this node won't be expanded

(Extras) Measures of dispersion for continuous-valued variables

- There are various **measures of statistical dispersion**, in addition to *Standard Deviation*, they can be more or less robust to outliers and to deviations from normality assumptions in the data



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

- Standard deviation
- Interquartile range (IQR)
- Range
- Mean absolute difference (also known as Gini mean absolute difference)
- Median absolute deviation (MAD)
- Average absolute deviation (or simply called average deviation)
- Distance standard deviation

Note: *(Differential) Entropy* can be defined and used for real-valued random variables, but as some issues ...

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

(Extras) Notes on Measures of dispersion for discrete-valued labels

- In addition to *Entropy* and *Information Gain*, other measures of purity / dispersion can and have been employed in the case of classification tasks (discrete-valued labels)
- Two popular measures are **Gain Ratio** and **Gini Index**, that try to tackle some issues of Entropy and IG

❖ **Gain Ratio** (introduced in C4.5)

- Motivation: *IG doesn't make any difference between attributes that split over a large number of partitions vs. attributes that are more parsimonious in branching*. IG only measures the overall weighted average of the entropy of the partition set. However, attributes with large number of children nodes (branching) make the tree growing more, and, therefore, are more likely to *overfit rather than generalize*. We can see an attribute with many possible labels as being a very *specialized* attribute, which might have the tendency to overfit when chosen.
  - If an attribute A can take many values, it will split its data in multiple, possibly *specialized* partitions, where each partition has likely a low entropy, such that overall, the IG will be high.
  - E.g., A = *Credit card number*, splitting on A generates large number of partitions, one for each possible card number, each possibly identifying a very specific data example/person. Entropy of each partition will be low and the overall IG(A) will be high. However, DT will grow large and it'll likely overfit training data since it is using person-specific information → Not expected to generalize well.

(Extras) Notes on Measures of dispersion for discrete-valued labels

❖ **Gain Ratio** (introduced in C4.5)

- Idea: In order to minimize the risk of overfitting, bias the growth of the the decision tree *against* choosing attributes with large number of distinct (highly specialized) values.
- Implementation: Measure the entropy of the overall splitting (*SplitInfo*) and use it to penalize splits with high split entropy, that correspond to the case of many specialized partitions, each with a low entropy

If S is a labeled set, Y is the set of possible labels, A is a candidate attribute to split S on, and n is the number of partitions that would be generated by A's split:

$$Gain\ Ratio = \frac{Information\ Gain}{SplitInfo} = \frac{H_S(Y) - H_S(Y|A)}{H\ (Split(A))} = \frac{H_S(Y) - H_S(Y|A)}{\sum_{i=1}^{n} w_i \ln w_i}$$

$$w_i = \frac{\#\ examples\ in\ partition\ i}{\#\ examples\ in\ S}$$

(Extras) Notes on Measures of dispersion for discrete-valued labels

❖ **Gini Index** (used in CART, it can be used both for classification and regression)

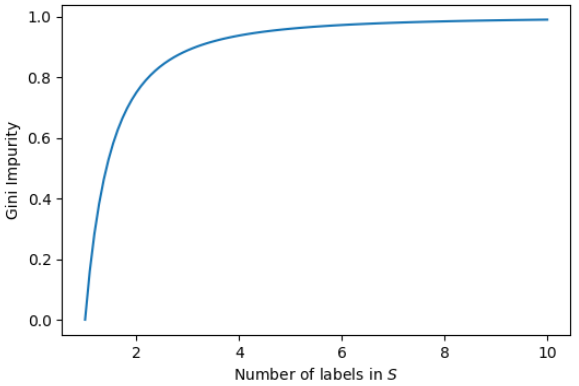
Gini Index is based on Gini's measure of impurity of a labeled set S, for a set of possible labels Y

- Gini Impurity: Measure of impurity of a labeled set S defined as one minus the sum of the squared class probabilities in the set (measured as frequencies)

$$Gini\ Impurity\ (S, Y) = 1 - \sum_{c=1}^{\#classes\ in\ S} p_c^2 \qquad p_c = \frac{\#\ examples\ of\ class\ c\ in\ S}{\#\ examples\ in\ S}$$

Gini impurity is a measure of how often a randomly chosen element from S would be incorrectly labeled if it was labeled according to the distribution of labels in the subset.

- If only one label / class is present in S, Gini Impurity is 0. Then, it grows with the growth of the number of classes and based on how uniform they are in S
- If all classes/labels are equally spread in the set S, $p_c = \frac{1}{\#classes}$, and Gini's Impurity grows with the number of classes as shown in the figure



(Extras) Notes on Measures of dispersion for discrete-valued labels

❖ **Gini Index** (used in CART)

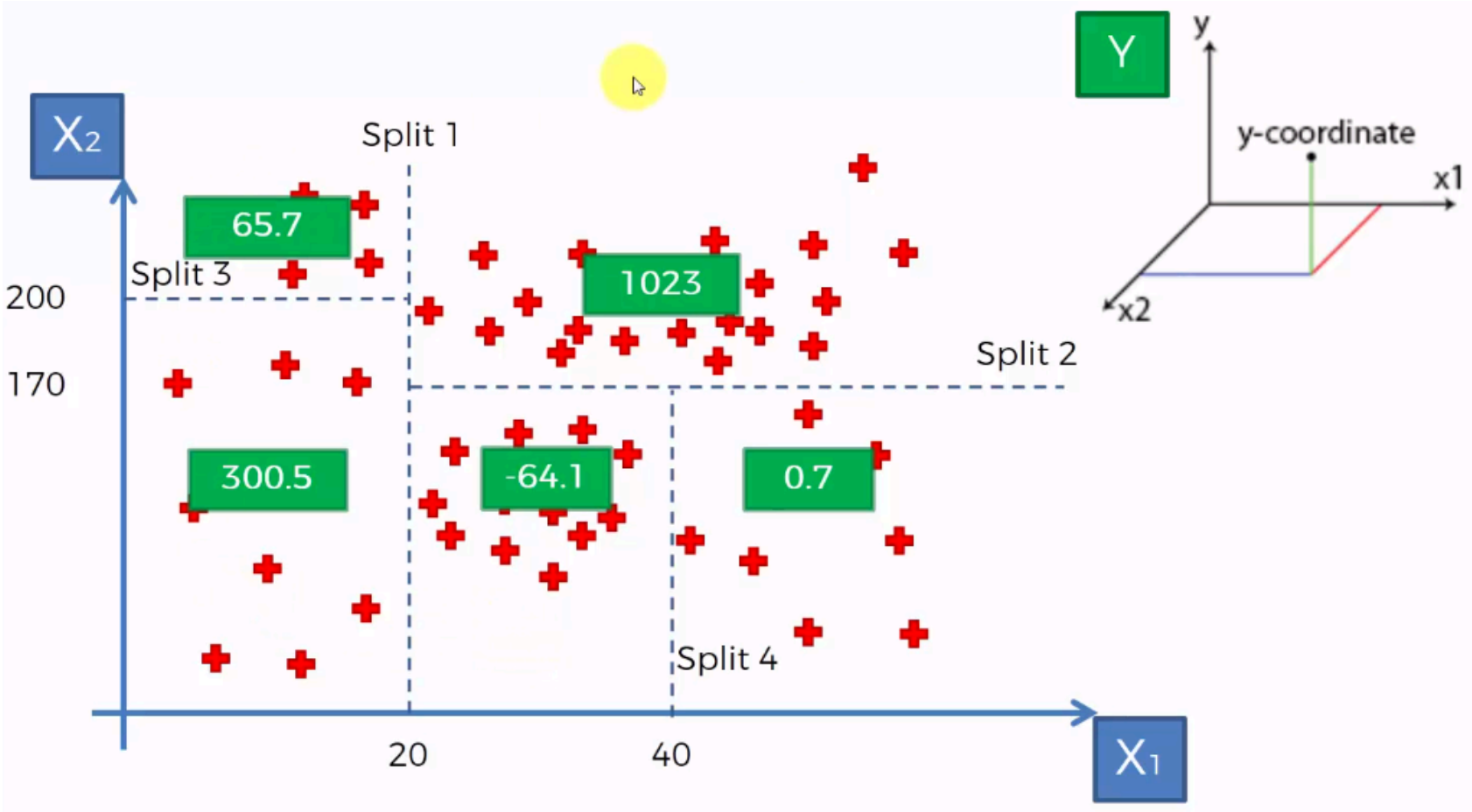
- Implementation: *Gini Index* is defined as the **weighted sum of the Gini Impurity of the different partitions of S after a split on an attribute A**, where each portion is weighted by the ratio of the size of the partition with respect to the size of the parent dataset S. Y is the set of discrete labels.

$$Gini\ Index(S, A) = \sum_{\pi=1}^{\#partitions} w_\pi\ Gini\ Impurity\ (\pi, Y) \qquad w_\pi = \frac{\#\ examples\ in\ partition\ \pi}{\#\ examples\ in\ S}$$

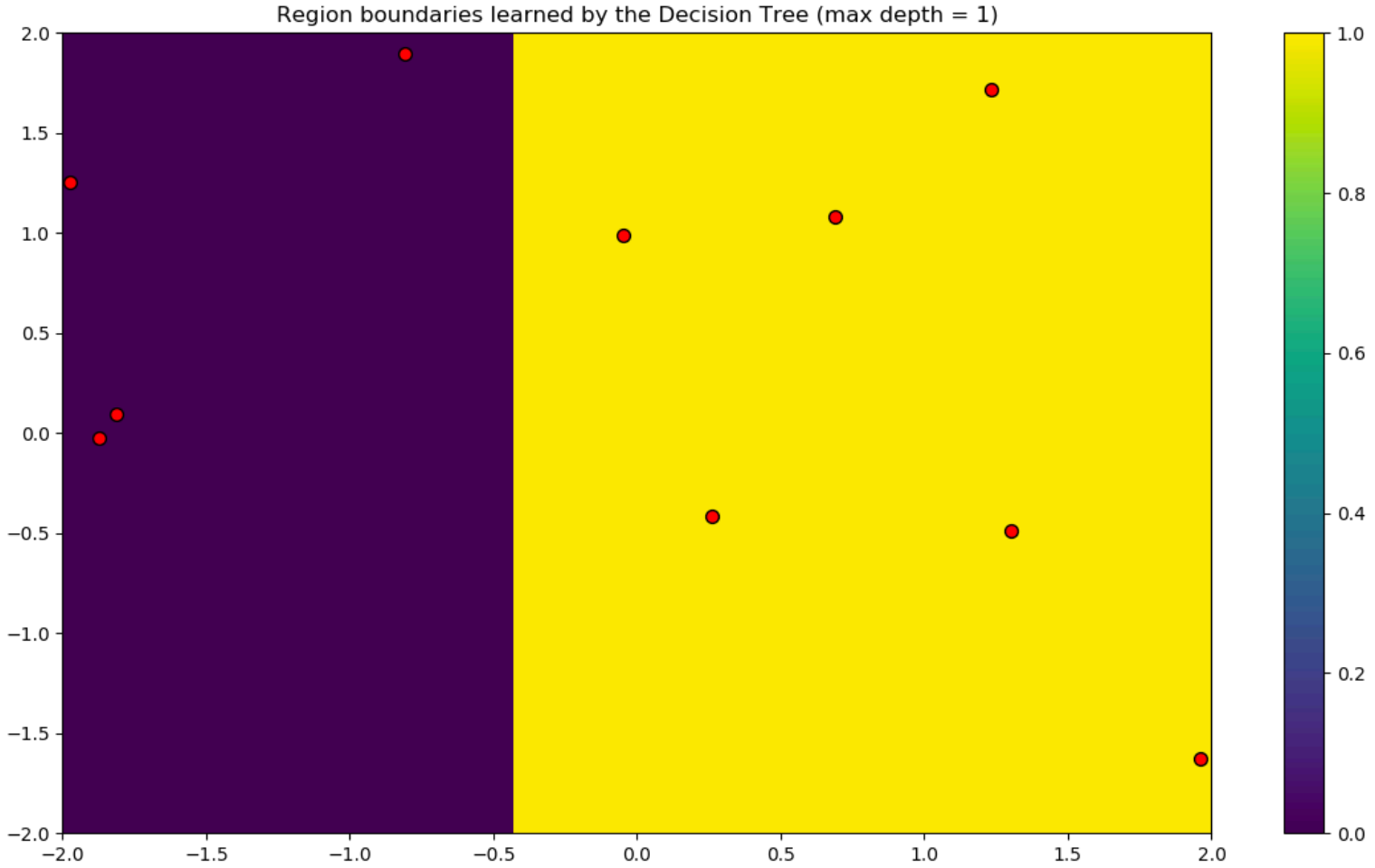
For a dataset S with two classes, Gini's Index ranges from 0 to 0.5: 0 if the dataset is pure, and 0.5 if the two labels are distributed equally among the data.

→ The attribute with the *lowest* Gini Index is used as the next splitting attribute

DT for Regression Tasks



DT for Regression Tasks vs. Max depth

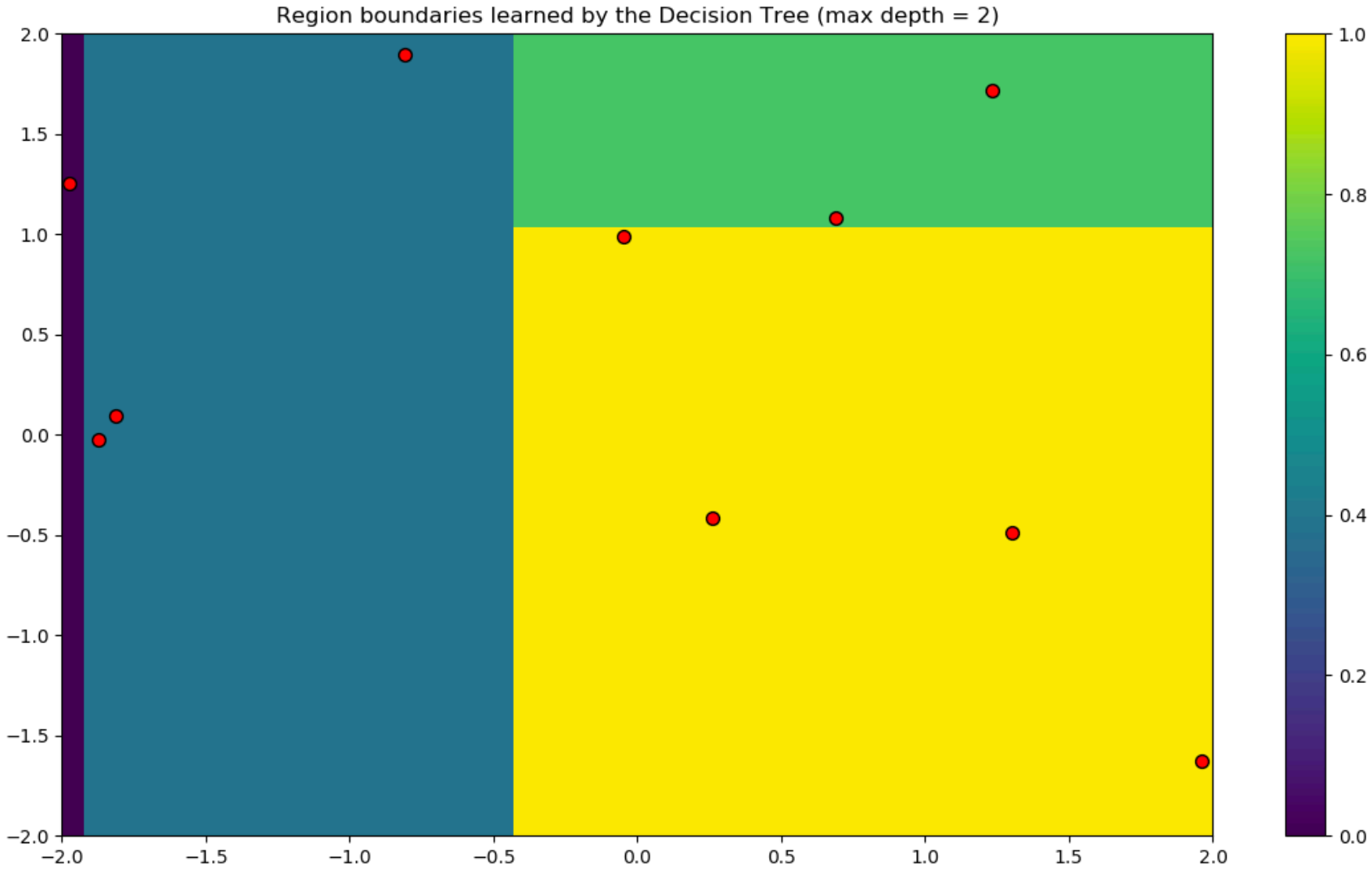


- Number of partitions in the feature space depend on the **max depth** parameter, that constrains the growth of the tree
- Dataset D consists of 10 points → it <u>easy to overfit</u> if we don't constrain the DT
- The prediction in each region is the *average of the values Y of the training points in the region*

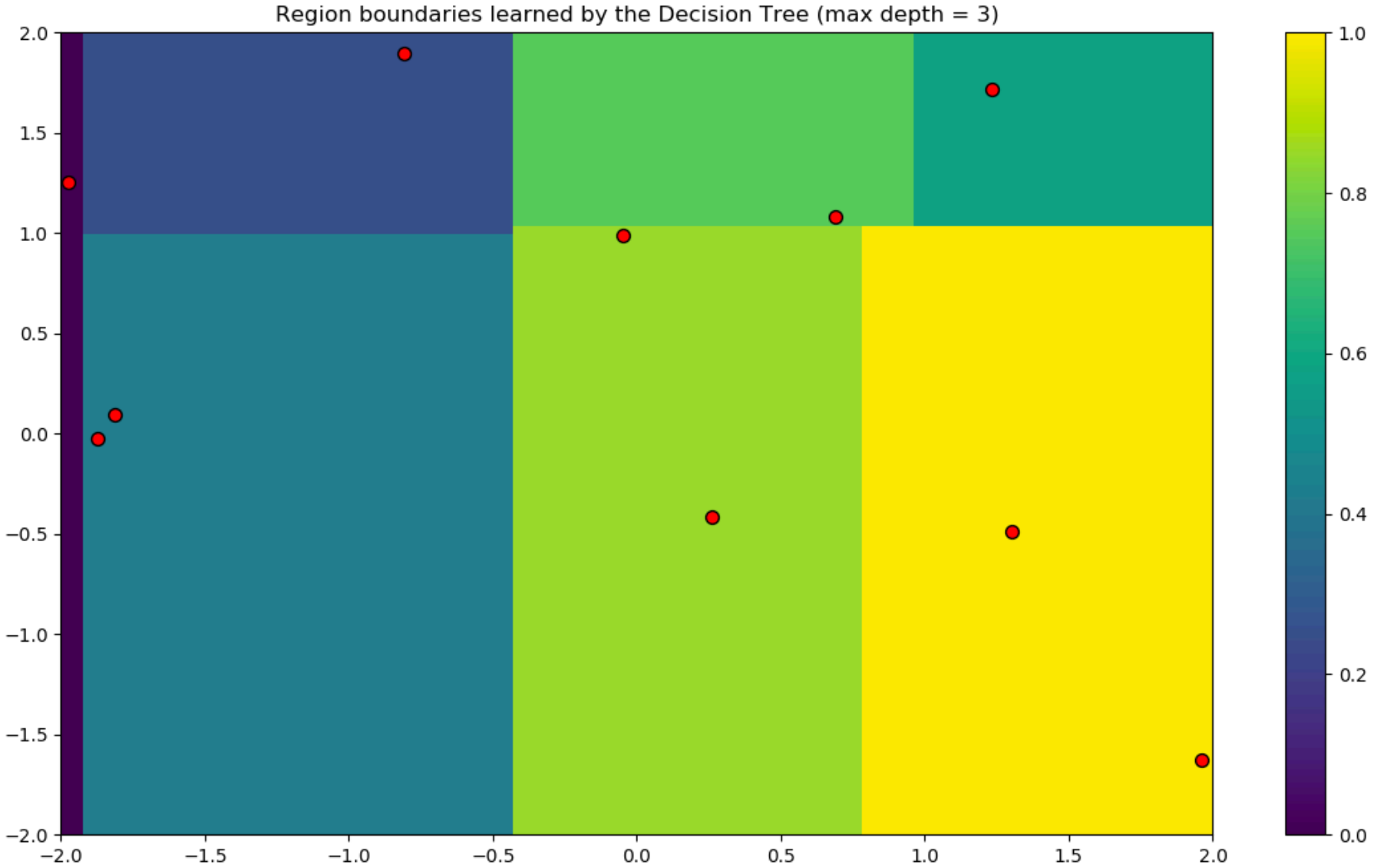
D = { [0.69, 1.08], [-0.05, 0.99], [1.3 , -0.49], [-1.87, -0.02], [1.23, 1.72], [0.26, -0.42], [-0.81, 1.9], [-1.81, 0.1], [1.96, -1.63], [-1.97, 1.25] }

Y = { -7.3 , 1.68, 11.33, -25.33, -16.59, -5.12, -35.6 , -26.26, 2. , -49.47 }

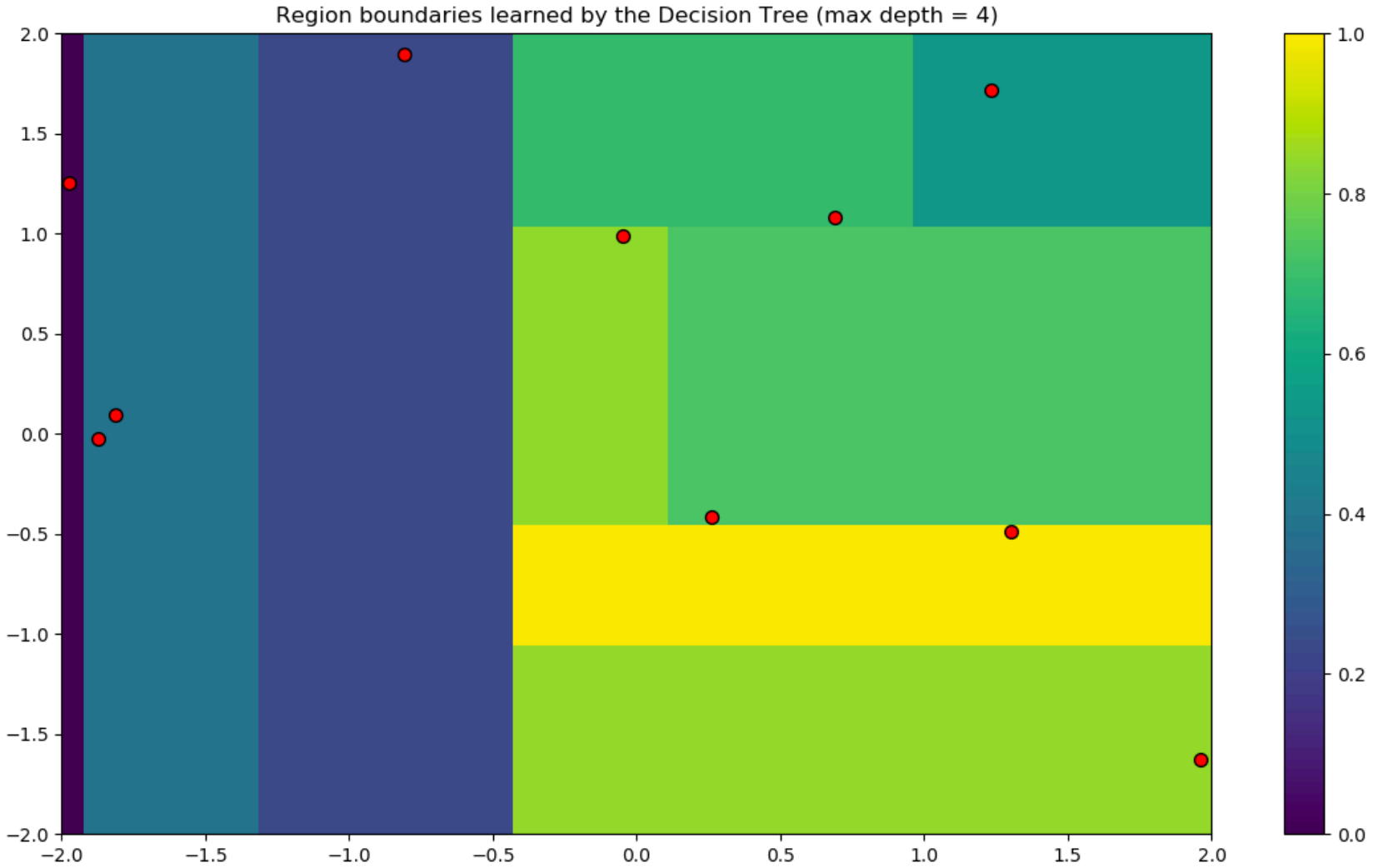
DT for Regression Tasks vs. Max depth



DT for Regression Tasks vs. Max depth

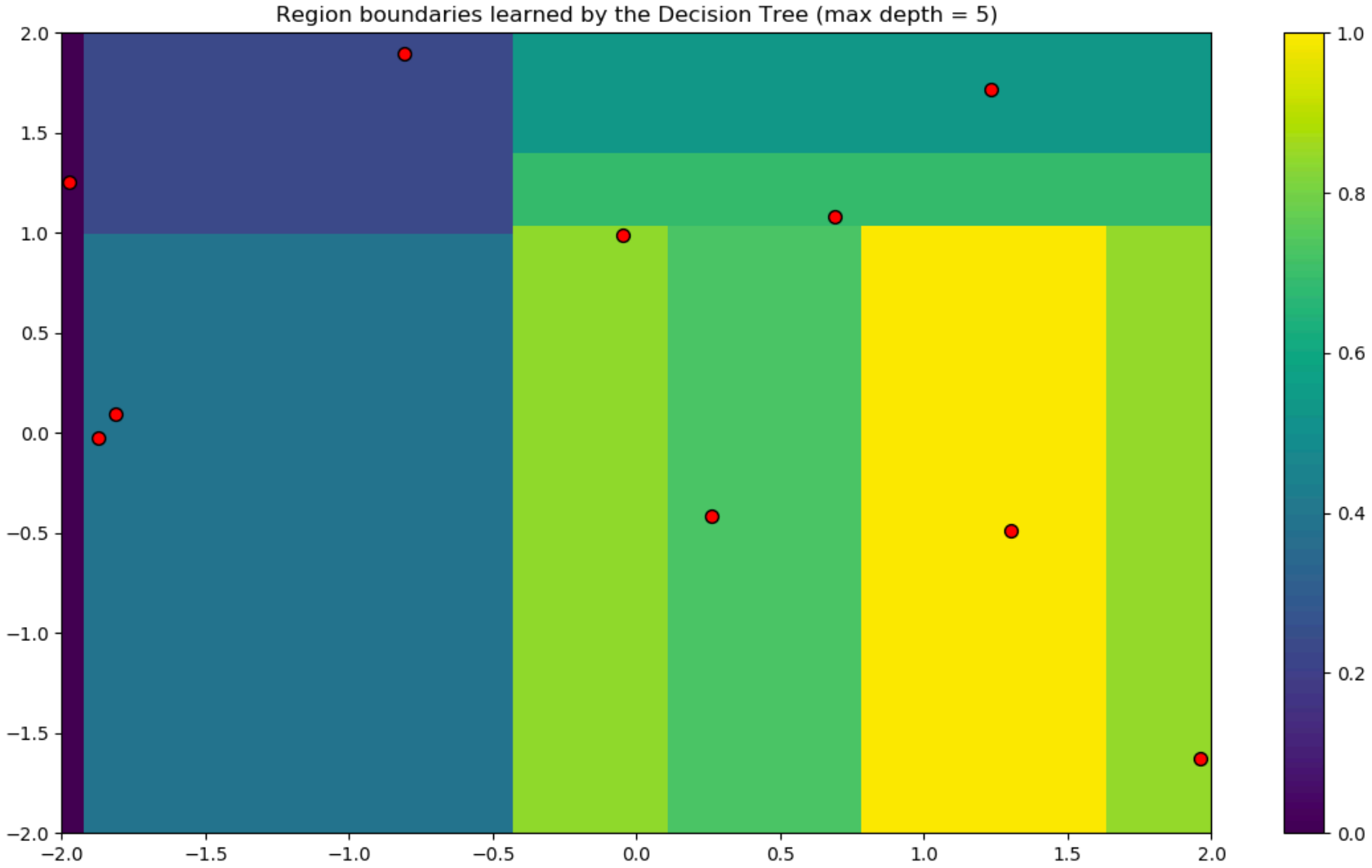


DT for Regression Tasks vs. Max depth



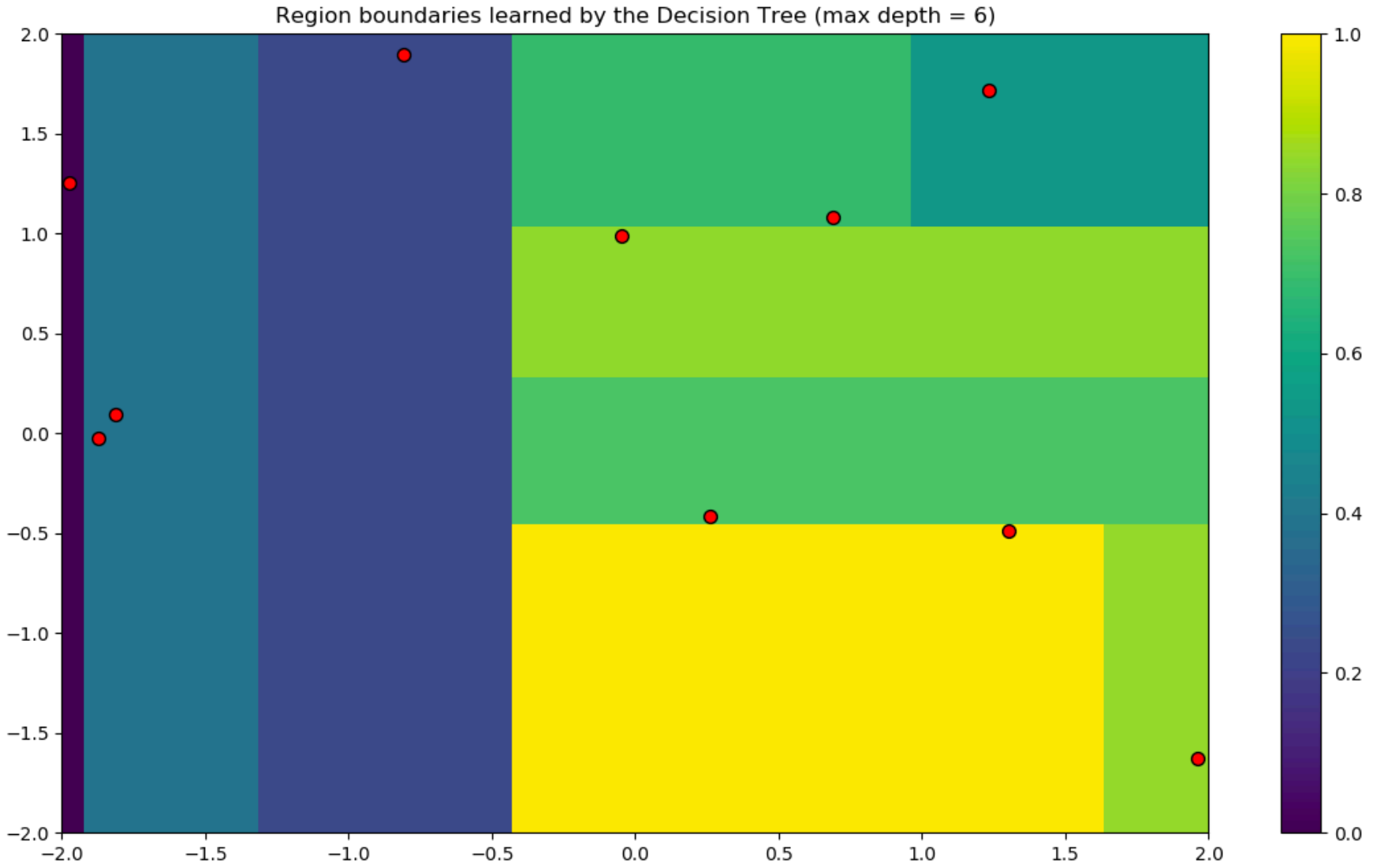
9 partitions out of 10 data points → Overfitting!

DT for Regression Tasks vs. Max depth



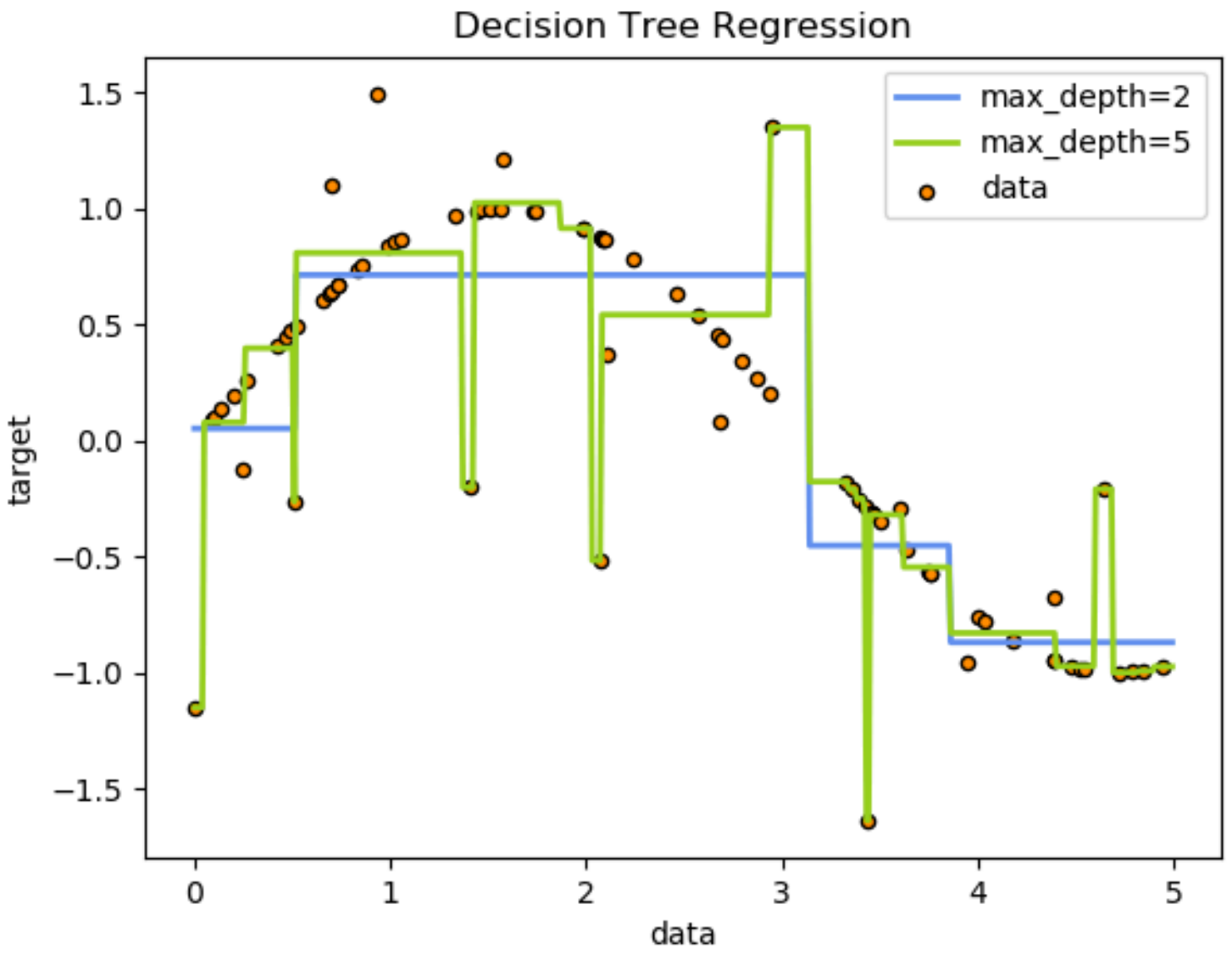
10 partitions out of 10 data points → Overfitting!

DT for Regression Tasks vs. Max depth



10 (different) partitions out of 10 data points → Overfitting!

DT for Regression Tasks: 1D example



- Training data points (the orange circles) have been generated out of a periodic function adding some noise to target values.
- Green and blue regions indicate the partitions found by the DT with different max depths

Do-it-yourself: Construct the classification decision tree

Day	Outlook	Temperature	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	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	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

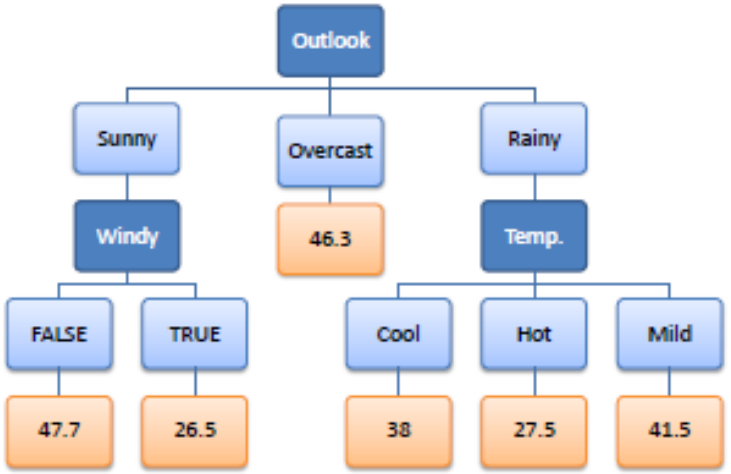
- ❖ Given the training dataset for the play tennis problem, construct by hand the full decision tree for the classification QuAM
  - ○ Use the Information gain to guide greedy attribute selection for splitting

1. Start by computing the entropy of the initial dataset, and proceed by computing the IG for all the attributes
2. Make the greedy selection and proceed until you reach leaf nodes that cannot or be expanded further
3. Draw the resulting decision tree with the decisions at the leafs
4. Report all numerical calculations

- ✓ Suggestion: implement a simple python function that makes the compution of the entropies / IG for you ☺
- ✓ The final tree consists of five leafs and two internal nodes

Do-it-yourself: Construct the regression decision tree

Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	45
Sunny	Cool	Normal	False	52
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	35
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	46
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



- Given the training dataset for the play tennis problem, let's consider the number of hours played, that now we want to use for prediction, making the problem a regression one.

- Use the Standard Deviation and the Coefficient of Variation to guide (and stop) greedy attribute selection for splitting

1. Start by computing the STD of the initial dataset, and proceed by computing the STD for all the attributes
2. Make the greedy selection and proceed until you reach leaf nodes that cannot be expanded further based on CoV at 10%
3. Draw the resulting decision tree with the decisions at the leafs
4. Report all numerical calculations

✓ Suggestion: implement a simple python function that makes the compution of STD and CoV for you ☺

✓ The final tree should look like the one shown to the left