

# MACHINE LEARNING BASIC CONCEPTS

# SUPERVISED MACHINE LEARNING CLASSIFICATION TASK.

**T = training set**

**future data**

Years	Amount	Salary	Own house?	Overdue accounts?	Repay loan
10	50000	3000	Yes	0	??

**Attributes, features, predictors,**  
Input variables, Independent  
variables, explanatory variables

Label, **class**, output variable, dependent  
variable, **response**, predictand, target

**Instances, examples, data points**

Years	Amount	Salary	Own house?	Overdue accounts?	Repaid loan
15	60000	1900	Yes	2	No
2	30000	3500	Yes	0	Yes
9	9000	1700	Yes	1	No
15	18000	3000	No	0	Yes
10	24000	2100	No	0	No
...	...	...	...	...	...

Algorithm

**Model**

**IF OA > 0 THEN NO**

**IF OA == 0 AND  
S > 2500 THEN Yes**

Repaid loan = yes

# DEFINITIONS: ATTRIBUTES AND RESPONSE

- **Attributes / features / input variables**

- A feature is an individual measurable property of an instance
- Types:
  - Numeric: (real numbers or integers):
    - 0, 1, 2
    - 1.3, 7.9, 10.798, ...
  - Categorical: yes / no ; red / green / yellow
    - Many ML algorithms cannot deal with them: they must be encoded into numbers (one-hot-encoding, ...)
  - Ordinal: cold, lukewarm, hot
    - Most ML cannot deal with them, they are encoded as integer numbers

- **Response / output variable**

- If categorical (e.g. cancer / no cancer) => classification problem
  - If it has two values: binary classification, otherwise, multi-class problem
- If real / integer number => regression problem

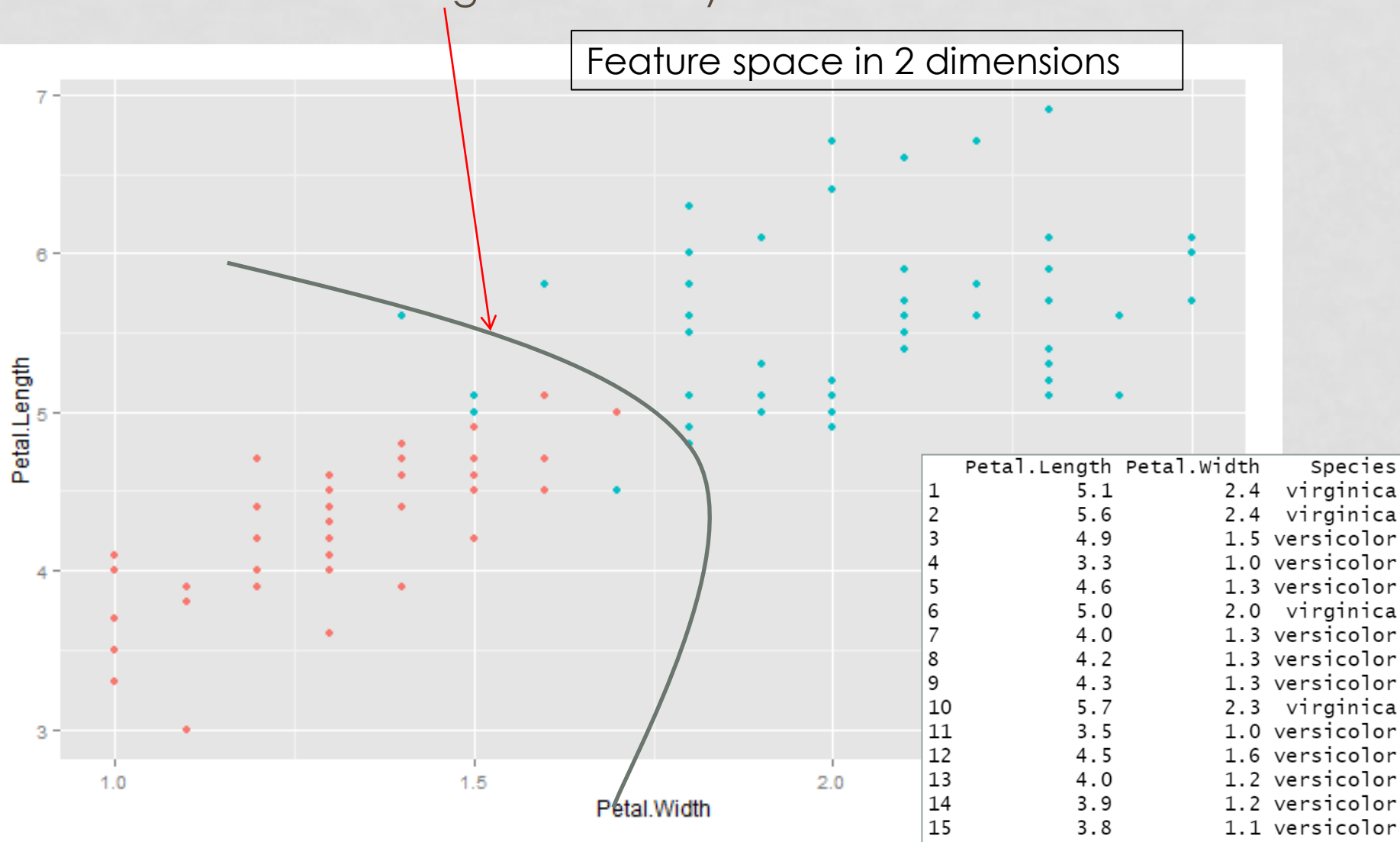
# DEFINITIONS: FEATURE SPACE (INSTANCE SPACE)

- Instances, examples = tuples
  - In general, they inhabit a d-dimensional space (instance space)
  - (input, output) =  $(\mathbf{x}_i, y) = (x_{i1}, x_{i2}, \dots, x_{id}, y) \in (\mathbf{R}^d, Y)$ 
    - Note: **boldface** means vector
    - This instance has 5 inputs and 1 output. It inhabits a 5-dimensional space

Years	Amount	Salary	Own house?	Overdue accounts?	Repaid loan
15	60000	1900	Yes	2	No

- In 2-dimensions (2 attributes), each instance is a point in instance space

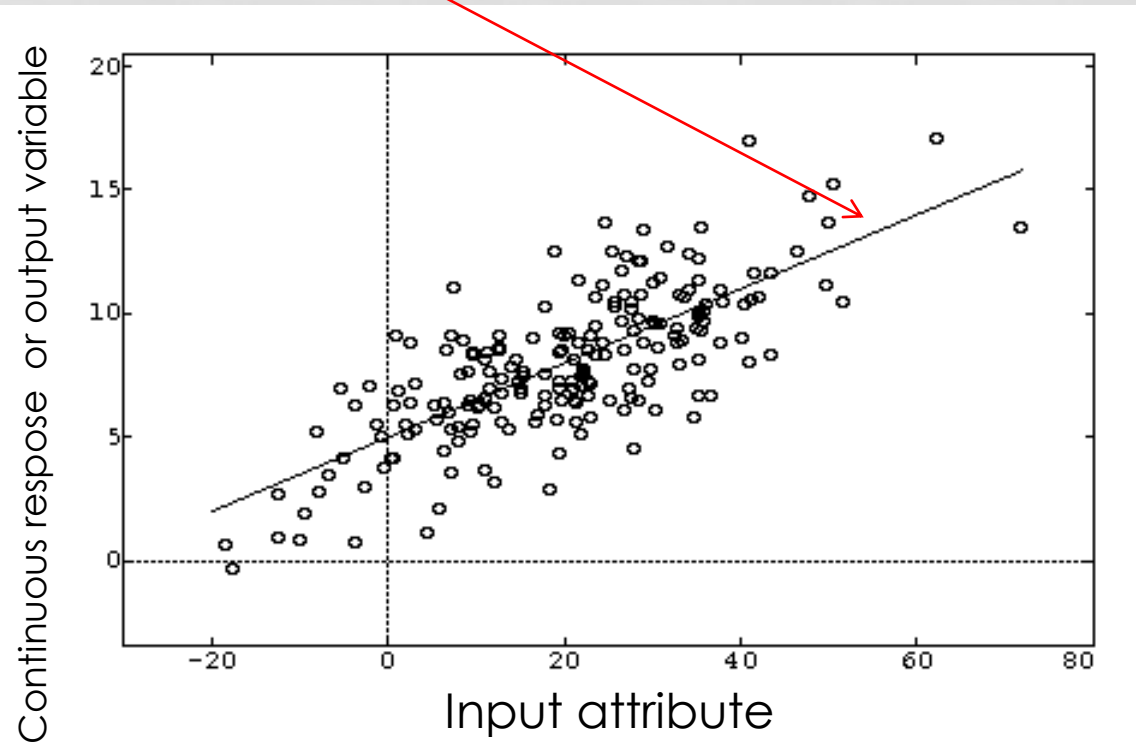
- Example: classify plants into two classes ("versicolor" / red vs. "virginica" / blue)
- 2 attributes = (Petal.Width, Petal.Length) = 2 dimensions
- Classification = finding a boundary between the classes



# FEATURE SPACE IN REGRESSION

- In the following case, there is one input variable and the output variable / response (continuous “label”)
- Regression = finding a function that transforms the inputs into the output

input	output
rooms	HousingPrice
6.575	24.0
6.421	21.6
7.185	34.7
6.998	33.4
7.147	36.2
6.430	28.7
6.012	22.9
6.172	27.1



# SUPERVISED MACHINE LEARNING CLASSIFICATION TASK. AN EXAMPLE:

**T = training set**

**future data**

Years	Amount	Salary	Own house?	Overdue accounts?	Repay loan
10	50000	3000	Yes	0	??

**Attributes, features, predictors,**  
Input variables, Independent  
variables, explanatory variables

Label, **class**, output variable, dependent  
variable, **response**, predictand, target

**Instances, examples, data points**

Years	Amount	Salary	Own house?	Overdue accounts?	Repaid loan
15	60000	1900	Yes	2	No
2	30000	3500	Yes	0	Yes
9	9000	1700	Yes	1	No
15	18000	3000	No	0	Yes
10	24000	2100	No	0	No
...	...	...	...	...	...

Algorithm

**Model**

**IF OA > 0 THEN NO**

**IF OA == 0 AND  
S > 2500 THEN Yes**

Repay loan = yes

# SUPERVISED MACHINE LEARNING CLASSIFICATION TASK. AN EXAMPLE:

**T = training set**

**future data**

Years	Amount	Salary	Own house?	Overdue accounts?	Repay loan
10	50000	3000	Yes	0	??

**Attributes, features, predictors,**  
Input variables, Independent  
variables, explanatory variables

Label, **class**, output variable, dependent  
variable, **response**, predictand, target

**Instances, examples, data points**

Years	Amount	Salary	Own house?	Overdue accounts?	Repaid loan
15	60000	1900	Yes	2	No
2	30000	3500	Yes	0	Yes
9	9000	1700	Yes	1	No
15	18000	3000	No	0	Yes
10	24000	2100	No	0	No
...	...	...	...	...	...

Algorithm

**Model**

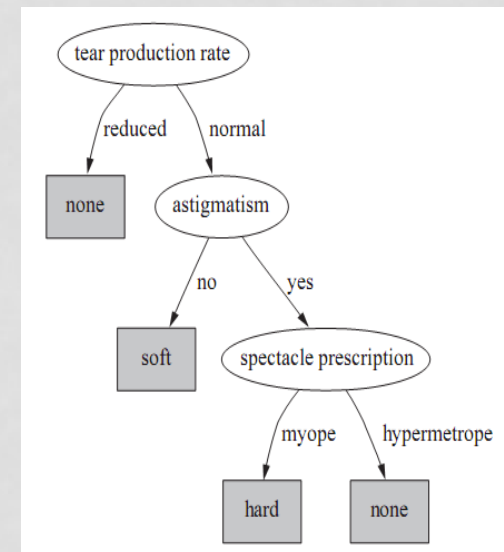
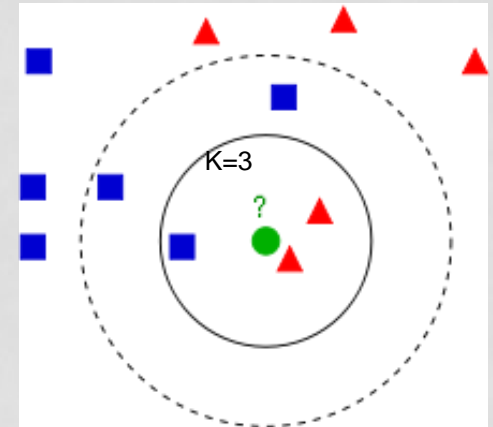
- **K-Nearest neighbors (KNN)**
- Trees, rules
- Ensembles (bagging, boosting, stacking, ...)
- Functions: neural networks, support vector machines, deep learning. ...

Repay loan = yes

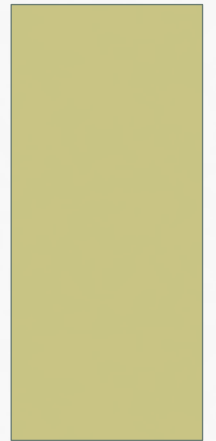


# SYLLABUS

1. Introduction to Machine Learning: tasks, algorithms & models
2. Basic methods for training classification and regression models:
  - **K Nearest Neighbours (KNN)**
  - Classification / regression trees & rules
3. Methodology
4. Methods for preprocessing
5. Advanced training methods based on ensembles of models
6. Large Scale Machine Learning. Big Data
7. Advanced topics
8. Software tools



# KNN: K-NEAREST NEIGHBOURS



KNN is a lazy method, because the "model" is the training data

Training data

Years	Amount	Salary	Own house?	Overdue accounts?	Repaid loan
15	60000	1900	Yes	2	No
2	30000	3500	Yes	0	Yes
9	9000	1700	Yes	1	No
15	18000	3000	No	0	Yes
10	24000	2100	No	0	No
...	...	...	...	...	...

KNN

Years	Amount	Salary	Own house?	Overdue accounts?	Repay loan
10	50000	3000	Yes	0	??



Model

Years	Amount	Salary	Own house?	Overdue accounts?	Repaid loan
15	60000	1900	Yes	2	No
2	30000	3500	Yes	0	Yes
9	9000	1700	Yes	1	No
15	18000	3000	No	0	Yes
10	24000	2100	No	0	No
...	...	...	...	...	...

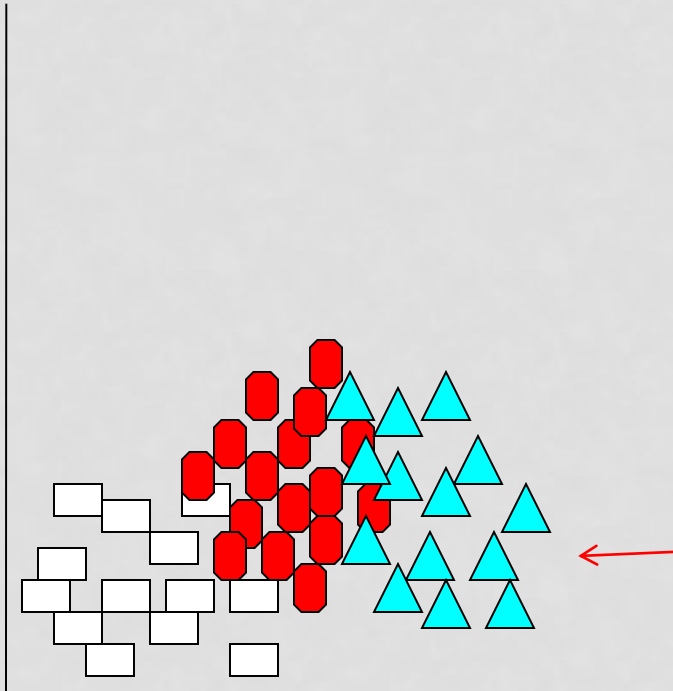


Yes

# K-NEAREST NEIGHBORS (KNN): training

Training

Height



□ Child

■ Adult

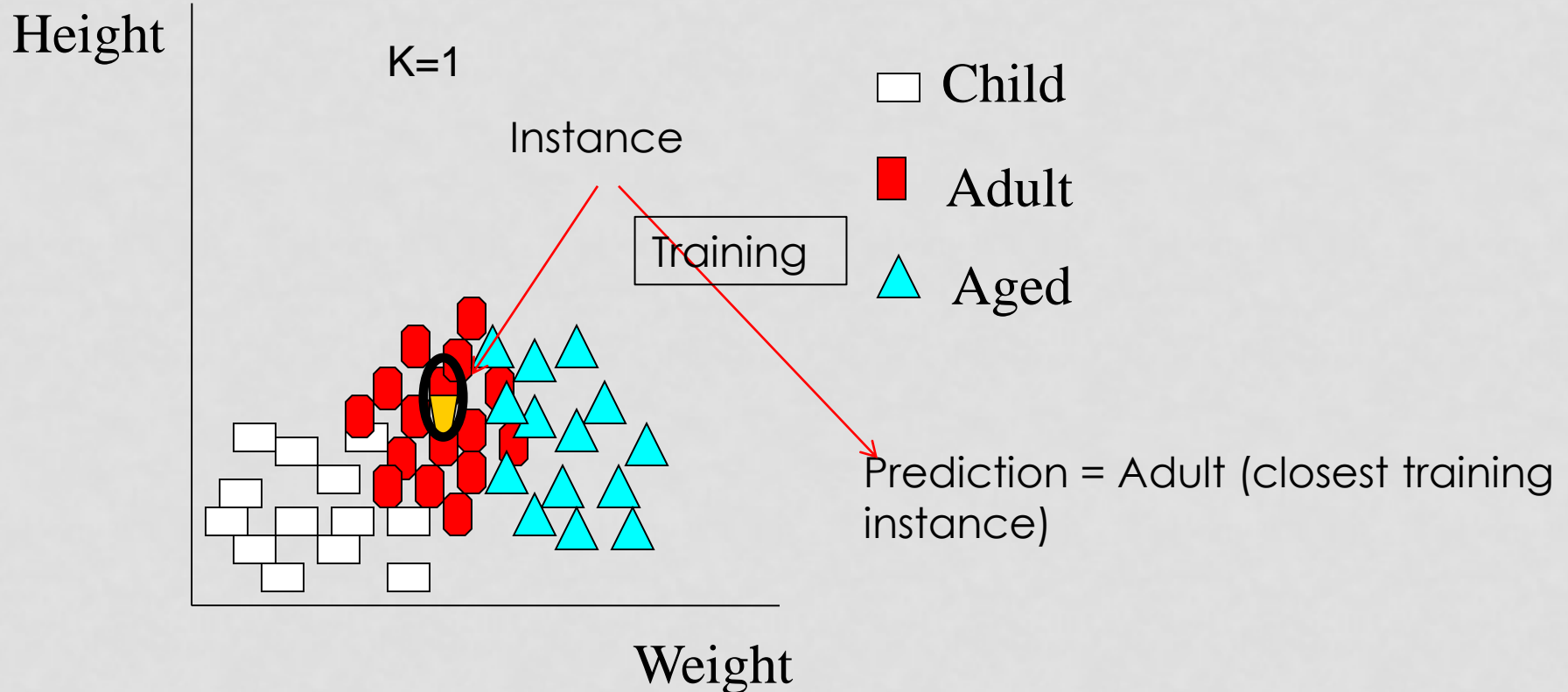
▲ Aged

Model = all training instances  
are stored

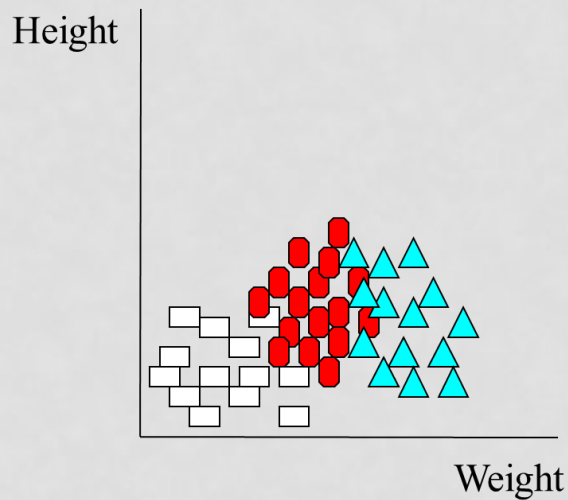
Weight

# K-NEAREST NEIGHBORS (KNN): prediction

## Prediction (inference)



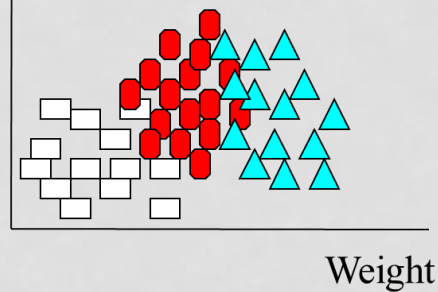
## Training data



Training

KNN

## Model



Adult

Height

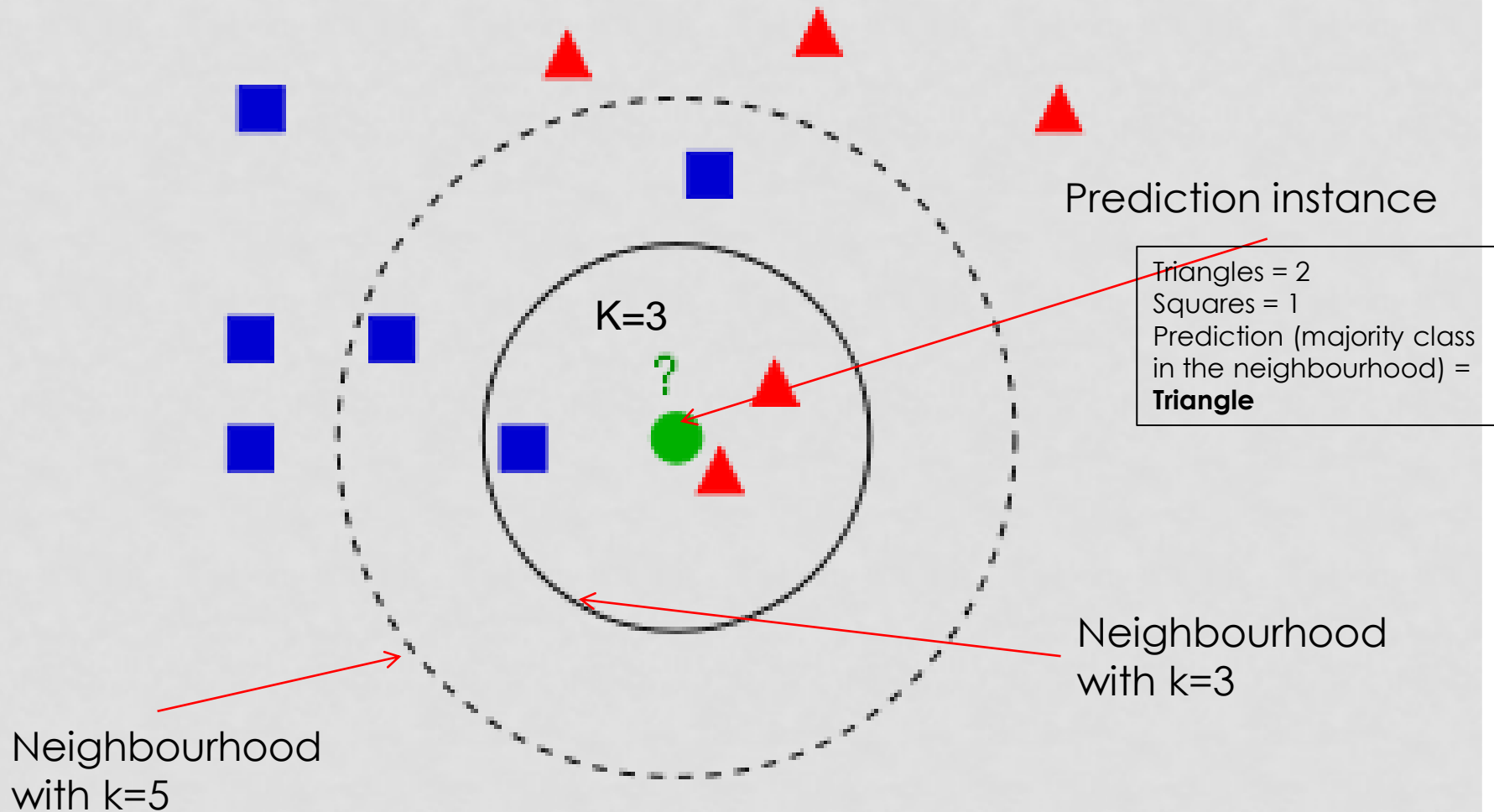
Weight

Prediction  
(inference)

?

# K-NEAREST NEIGHBORS (KNN)

$K > 2 \Rightarrow$  classify new instances as the majority class of the  $k$ -nearest neighbours

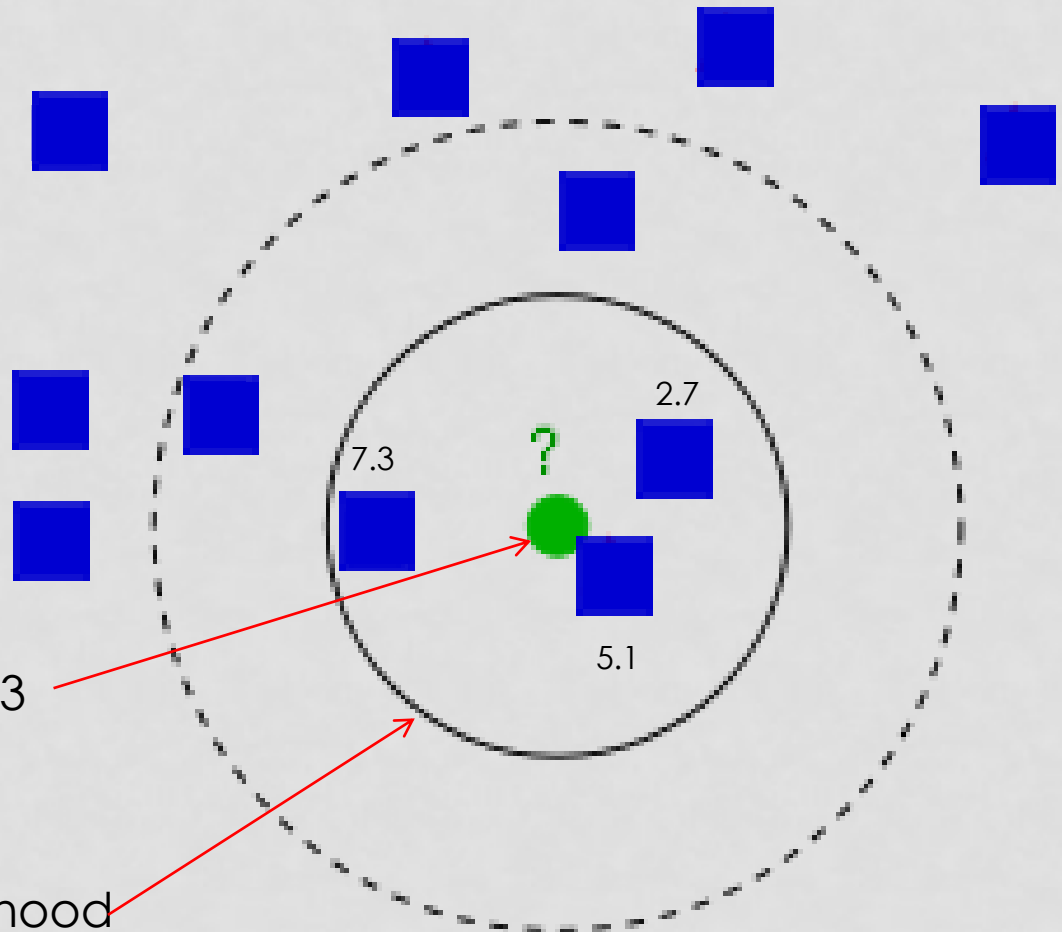


# KNN FOR REGRESSION

It can be easily extended for **regression** by computing the average of the k-nearest neighbours

$$\text{Prediction} = (7.3 + 2.7 + 5.1) / 3$$

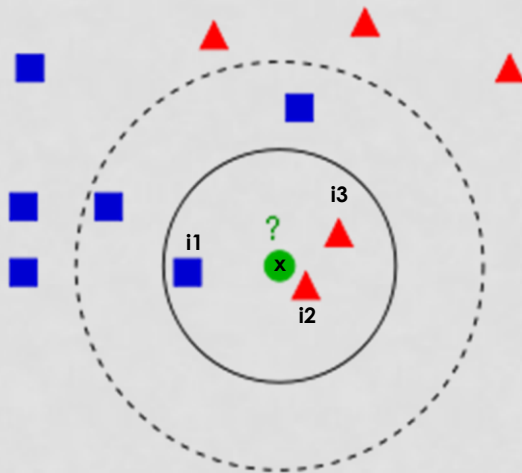
Neighbourhood  
with k=3



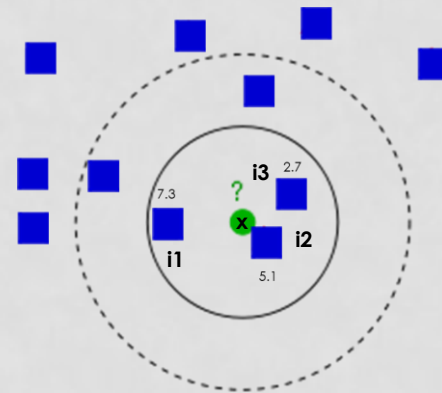


# INVERSE EUCLIDEAN DISTANCE WEIGHTING

- In some cases, it may be better to weight the neighbors so that nearer neighbors contribute more to the classification or regression
- The inverse of the distance is typically used
- "uniform" (euclidean distance) vs. "distance" (inverse euclidean distance)



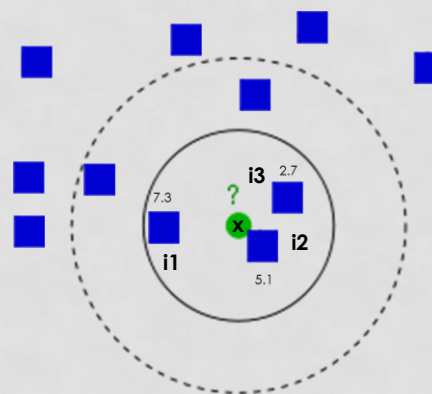
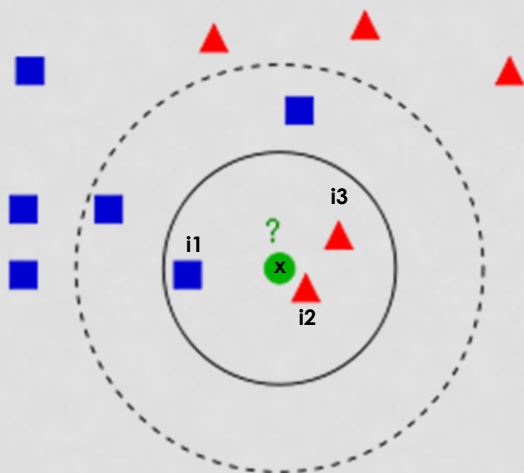
- Blue square:  $\frac{1}{d_1}$
- Red triangle:  $\frac{1}{d_2} + \frac{1}{d_3}$



$$\frac{\frac{7.3}{d_1} + \frac{5.1}{d_2} + \frac{2.7}{d_3}}{\frac{1}{d_1} + \frac{1}{d_2} + \frac{1}{d_3}}$$

# INVERSE EUCLIDEAN DISTANCE WEIGHTING

- In some cases, it may be better to weight the neighbors so that nearer neighbors contribute more to the classification or regression
- The inverse of the distance is typically used



**weights** : {'uniform', 'distance'}, callable or None, default='uniform'

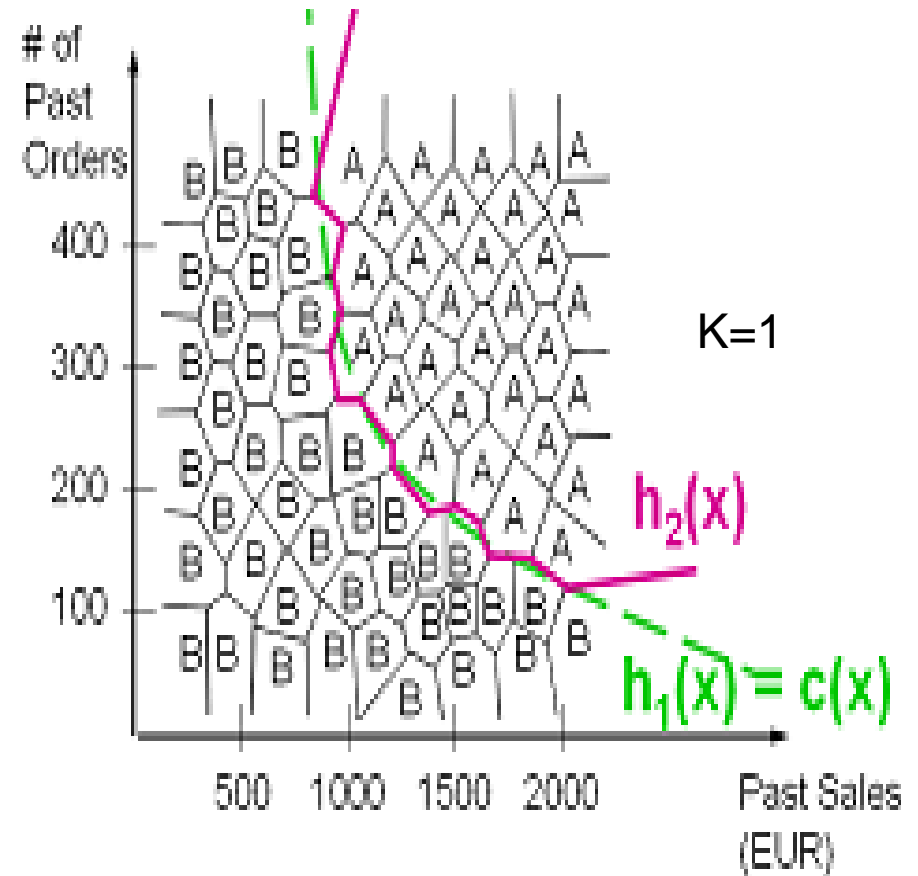
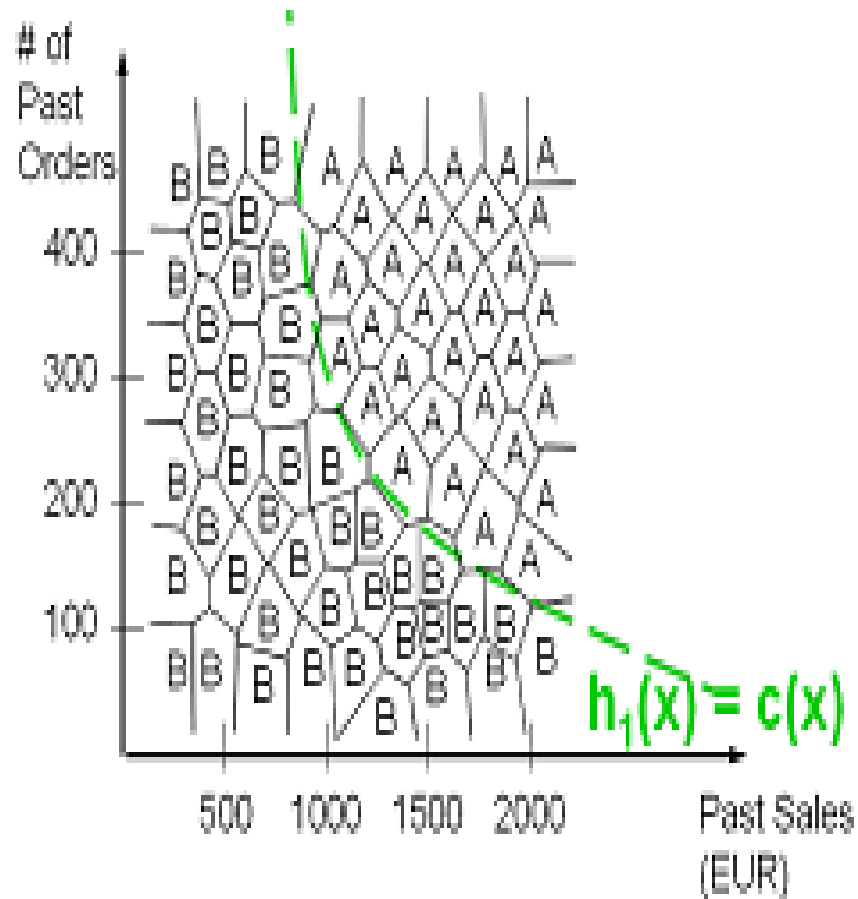
In scikit-learn

Weight function used in prediction. Possible values:

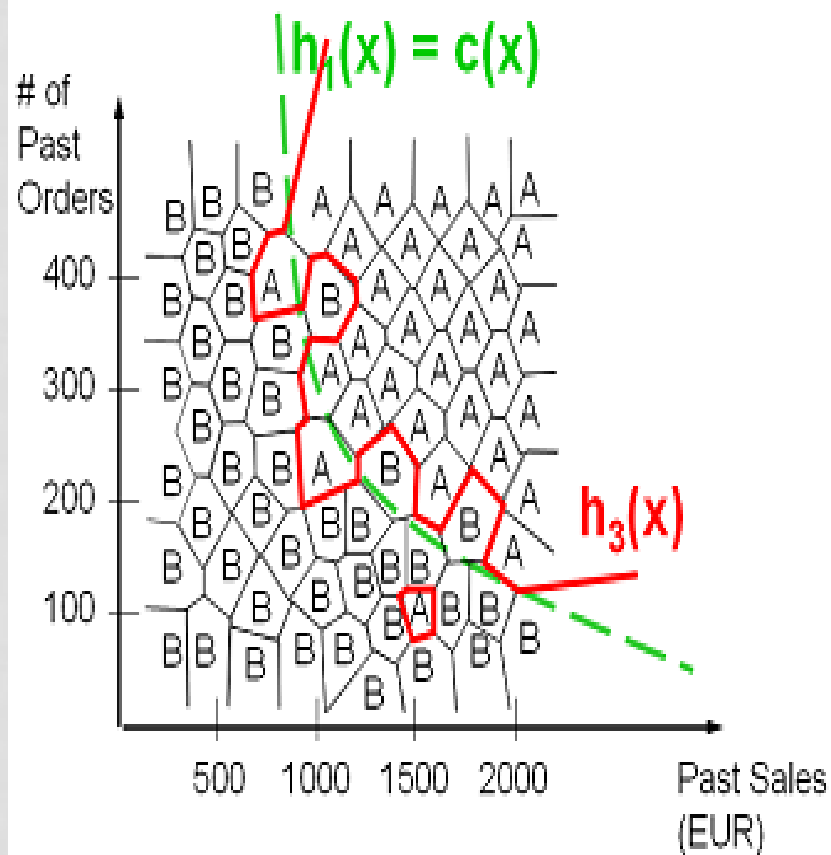
- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

# WHY USE $K > 1$ ?

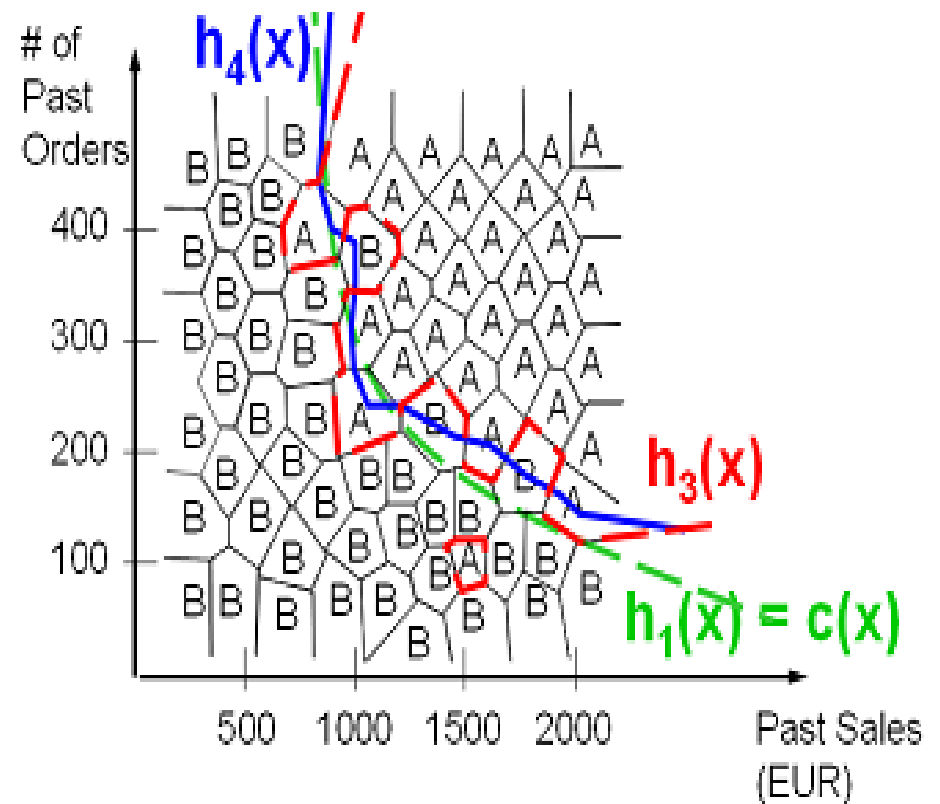
VORONOI TESSELLATION in Instance space



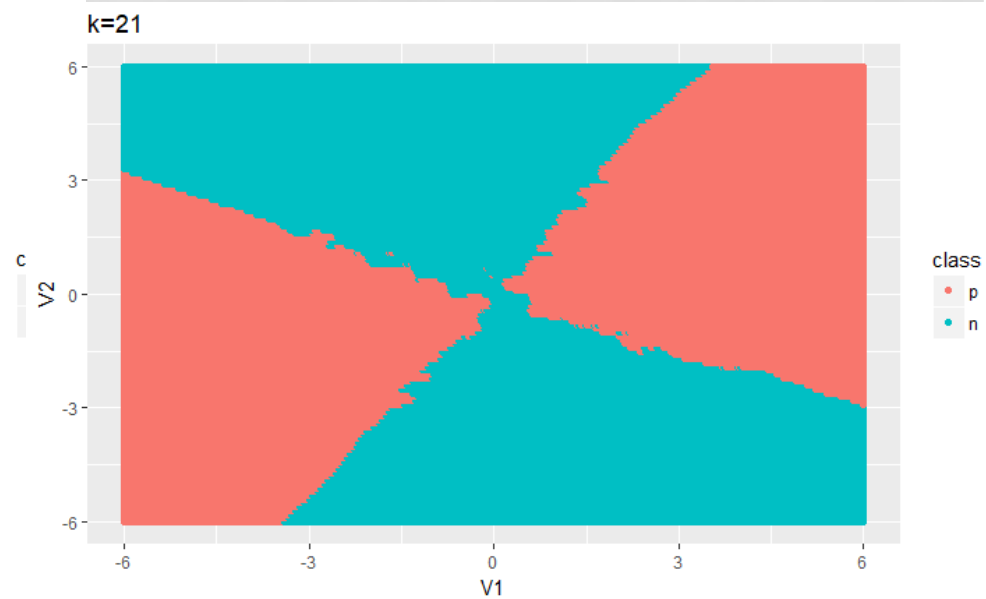
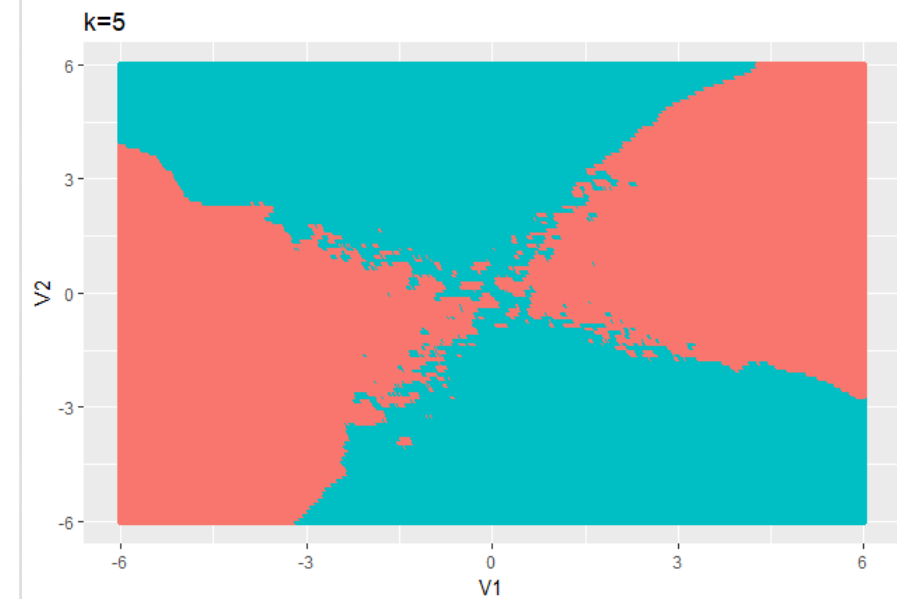
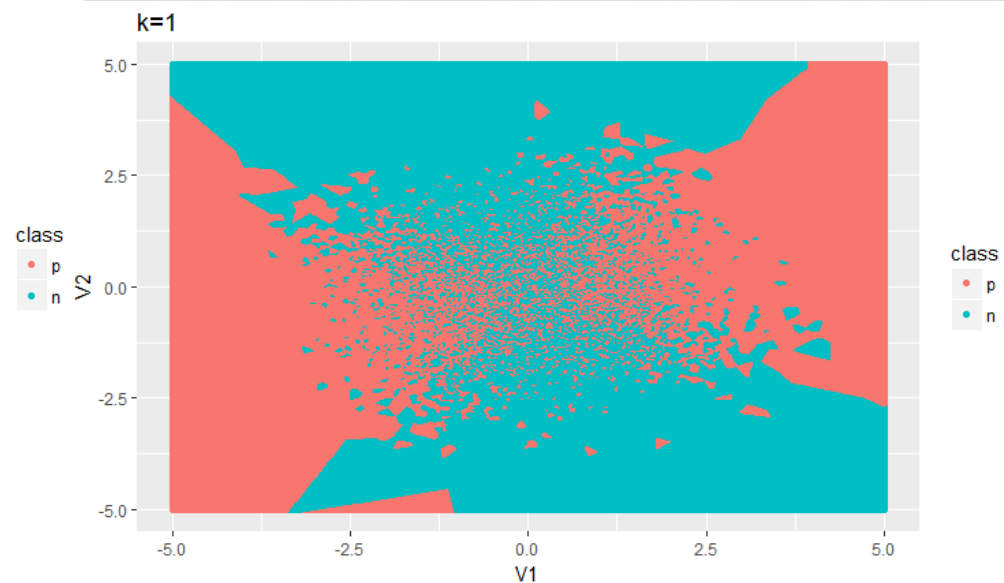
# WHY USE $K>1$ ?



(a) 1-NN on noisy data



(b) 3-NN and noisy data



# WHY USE $K > 1$ ?

- With  $k=1$ , noisy instances (i.e. class overlap) have a large influence
- With  $k > 1$ , more neighbors are considered and noisy instances have less influence (it is like averaging)
- But if  $k$  is very large, locality is lost
  - What is KNN if  $k$ =number of instances?
- If the number of classes is two, use odd  $k$  in order to avoid draws
- $K$  is the main **hyper-parameter** of KNN and has to be tuned properly. Hyperparameter tuning is part of the ML workflow, and we will talk about it in future lectures

# SUMMARY OF KNN

- KNN classifies test instances as the majority class in the neighbourhood of the training set
- KNN is a lazy ML algorithm
  - During training, no model is constructed, but all training instances are stored (model = training instances)
- It is based on the idea that the best model of data is the data itself
- It can be easily extended for **regression** by computing the average of the k-nearest neighbours

# ISSUES WITH THE USE OF THE EUCLIDEAN DISTANCE IN KNN

- instance #i:  $\mathbf{x}_i = (\text{weight}_i, \text{height}_i) = (w_i, h_i)$
- instance #k:  $\mathbf{x}_k = (\text{weight}_k, \text{height}_k) = (w_k, h_k)$
- For numerical attributes, the Euclidean distance is typically used:
  - 2D:  $d(\mathbf{x}_i, \mathbf{x}_k)^2 = (w_i - w_k)^2 + (h_i - h_k)^2$ 
    - $d(\mathbf{x}_i, \mathbf{x}_k) = \text{sqrt}[(w_i - w_k)^2 + (h_i - h_k)^2]$
  - dD:  $d(\mathbf{x}_i, \mathbf{x}_k)^2 = (x_{i1} - x_{k1})^2 + (x_{i2} - x_{k2})^2 + \dots + (x_{id} - x_{kd})^2$
- Consequences/issues of using the Euclidean distance in KNN:
  1. What to do with categorical variables?
  2. Different features may have different value ranges
  3. Sensitivity to irrelevant features
  4. What if the Euclidean distance is not the most appropriate?



# ISSUE #1. WHAT TO DO WITH CATEGORICAL VARIABLES

- instance #i:  $\mathbf{x}_i = (\text{weight}_i, \text{height}_i) = (w_i, h_i)$
- instance #k:  $\mathbf{x}_k = (\text{weight}_k, \text{height}_k) = (w_k, h_k)$
- For numerical attributes, the Euclidean distance is typically used:
  - $d(\mathbf{x}_i, \mathbf{x}_k)^2 = (x_{i1} - x_{k1})^2 + (x_{i2} - x_{k2})^2 + \dots + (x_{id} - x_{kd})^2$
- For categorical attributes: Hamming distance:
  - If attribute e is nominal (categorical), instead of  $(x_{ie} - x_{ke})^2$ , the following is used:  $\delta(x_{ie}, x_{ke})$ : 0 if  $x_{ie} = x_{ke}$  and 1 otherwise
    - e.g.  $\delta(\text{"high"}, \text{"high"}) = 0$ ;  $\delta(\text{"high"}, \text{"low"}) = 1$
- or transform the categorical attribute to one-hot encoding)

# ISSUES WITH THE USE OF THE EUCLIDEAN DISTANCE IN KNN

- instance #i:  $\mathbf{x}_i = (\text{weight}_i, \text{height}_i) = (w_i, h_i)$
- instance #k:  $\mathbf{x}_k = (\text{weight}_k, \text{height}_k) = (w_k, h_k)$ 
  - $d(\mathbf{x}_i, \mathbf{x}_k)^2 = (w_i - w_k)^2 + (h_i - h_k)^2$
- Consequences of using the Euclidean distance in KNN:
  1. What to do with categorical variables?
  2. **Different features may have different value ranges**
  3. Sensitivity to irrelevant features
  4. What if the Euclidean distance is not the most appropriate?

# ISSUE #2. SCALING (NORMALIZATION)

- It is important to scale (normalize) attributes, because ranges can be different (e.g. human body temperature ranges from 35° to 45° celsius, body weight ranges from 0 to 100kg, body height ranges from 0 to 2m, age ranges from 0 to 100 years, etc.)
- Otherwise, attributes with a large range have more weight on the distance
- Scaling attribute  $x_1$ :
  - MinMax (to 0-1 range):  $x'_{1j} = \frac{x_{1j} - \min(x_1)}{\max(x_1) - \min(x_1)}$
  - Standardization:  $x'_{1j} = \frac{x_{1j} - \bar{x}_1}{\sigma_1}$
  - Robust scaler:  $x'_{1j} = \frac{x_{1j} - \text{median}(x_1)}{\text{IQR}(x_1)}$ 
    - Scale features using statistics that are robust to outliers
    - The IQR (interquartile range) is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile)
    - The number of instances between the 1st quartile and the 3rd quartile is 50%

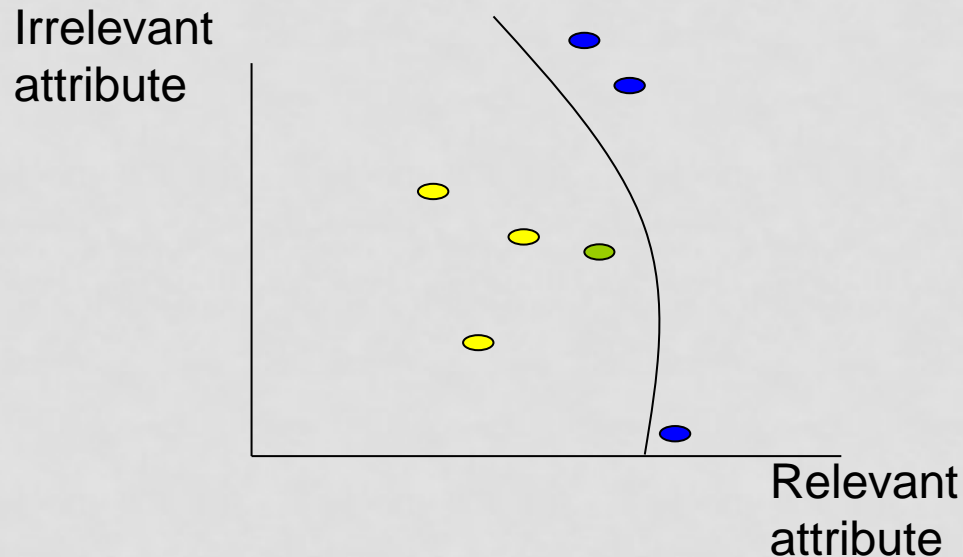
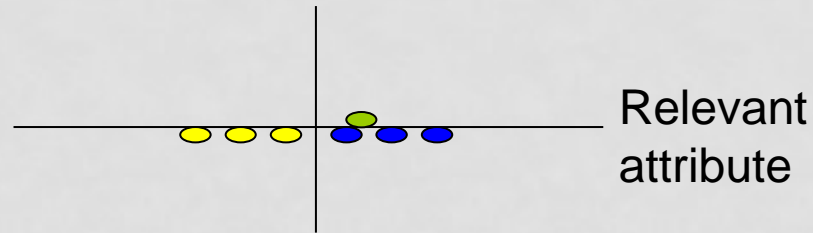
Sky	Temperature	Humidity	Wind	Tennis
Sunny	85	85	No	No
Sunny	80	90	Yes	No
Overcast	83	86	No	Yes
Rainy	70	96	No	No
Rainy	68	80	No	Yes
Overcast	64	65	Yes	Yes
Sunny	72	95	No	No
Sunny	69	70	No	Yes
Rainy	75	80	No	Yes
Sunny	75	70	Yes	Yes
Overcast	72	90	Yes	Yes
Overcast	81	75	No	Yes
Rainy	71	91	Yes	No

$\min(T)$     $\min(H)$   
 $\max(T)$     $\max(H)$

# ISSUES WITH THE USE OF THE EUCLIDEAN DISTANCE IN KNN

- $d(\mathbf{x}_i, \mathbf{x}_k)^2 = (x_{i1} - x_{k1})^2 + (x_{i2} - x_{k2})^2 + \dots + (x_{id} - x_{kd})^2$
- Consequences of using the Euclidean distance in KNN:
  1. What to do with categorical variables?
  2. Different features may have different value ranges
  3. **Sensitivity to irrelevant features**
  4. What if the Euclidean distance is not the most appropriate?

# ISSUE #3. SENSITIVITY TO IRRELEVANT ATTRIBUTES

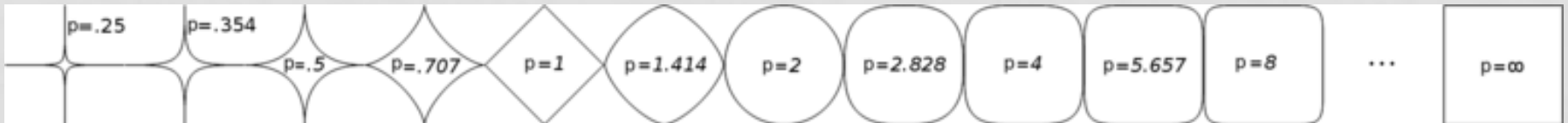


# ISSUES WITH THE USE OF THE EUCLIDEAN DISTANCE IN KNN

- instance #i:  $\mathbf{x}_i = (\text{weight}_i, \text{height}_i) = (w_i, h_i)$
- instance #k:  $\mathbf{x}_k = (\text{weight}_k, \text{height}_k) = (w_k, h_k)$ 
  - $d(\mathbf{x}_i, \mathbf{x}_k)^2 = (x_{i1} - x_{k1})^2 + (x_{i2} - x_{k2})^2 + \dots + (x_{id} - x_{kd})^2$
- Consequences of using the Euclidean distance in KNN:
  1. What to do with categorical variables?
  2. Different features may have different value ranges
  3. Sensitivity to irrelevant features
  4. **What if the Euclidean distance is not the most appropriate for a particular problem?**

# OTHER DISTANCES: MINKOWSKY

$$\left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}.$$

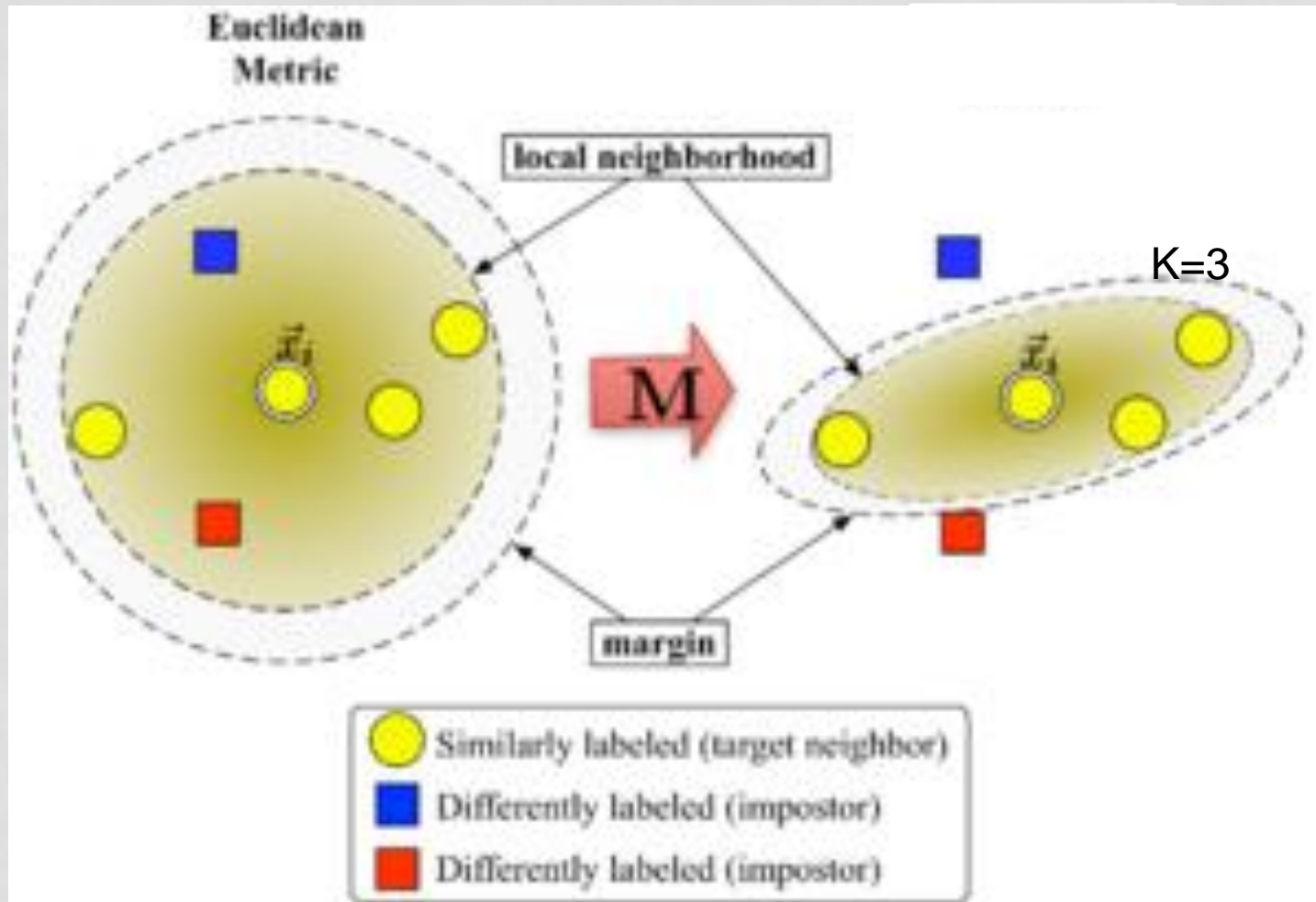


$p$  is another hyper-parameter in scikit-learn

**$p$  : int, default=2**

Power parameter for the Minkowski metric. When  $p = 1$ , this is equivalent to using `manhattan_distance` ( $l_1$ ), and `euclidean_distance` ( $l_2$ ) for  $p = 2$ . For arbitrary  $p$ , `minkowski_distance` ( $l_p$ ) is used.

# OTHER DISTANCES: METRIC LEARNING

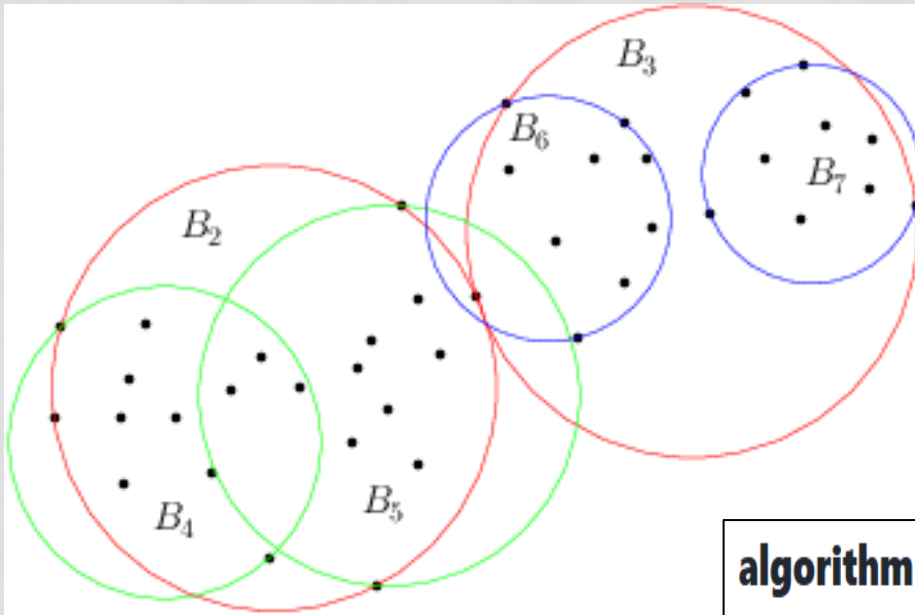




# LIMITATIONS OF KNN

- Slow (when making predictions): all distances to each training instance must be computed. Solution: ball-trees
- Large storage requirements (all training data must be stored). Possible solution: pre-processing with "condensation"
- Very sensitive to noise. Solutions:
  - Large K's
  - Pre-processing with Wilson editing
- Very sensitive to irrelevant attributes and the curse of dimensionality. Solutions: pre-processing with feature selection / feature extraction
- Depends on the metric (euclidean by default): Solution:
  - Minkowsky Distance
  - Metric learning

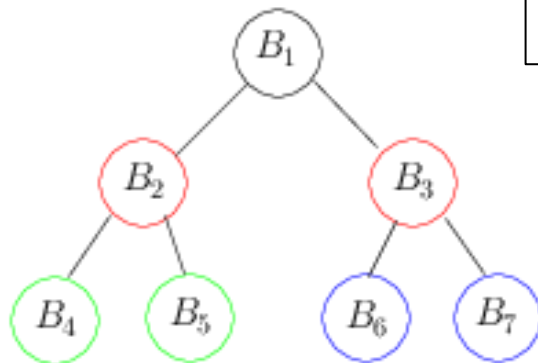
# BALL-TREES



In scikit-learn

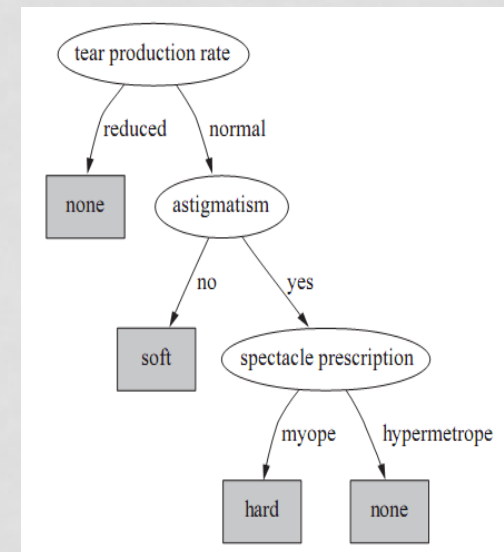
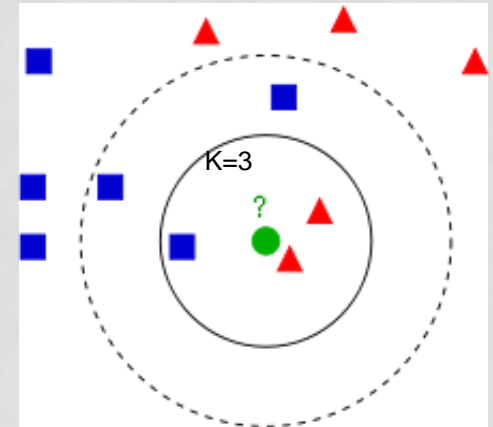
**algorithm** : `{'auto', 'ball_tree', 'kd_tree', 'brute'}`, `default='auto'`

Algorithm used to compute the nearest neighbors:

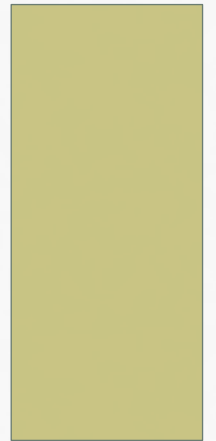


# SYLLABUS

1. Introduction to Machine Learning: tasks, algorithms & models
2. Basic methods for training classification and regression models:
  - K Nearest Neighbours (KNN)
  - **Classification / regression trees & rules**
3. Methodology
4. Methods for preprocessing
5. Advanced training methods based on ensembles of models
6. Large Scale Machine Learning. Big Data
7. Advanced topics
8. Software tools



MODELS:  
TREES (AND RULES) FOR  
CLASSIFICATION AND REGRESSION



# MODELS: DECISION TREES

Attributes, features,  
Input variables,  
Independent variables

Label, class, output variable,  
dependent variable

Future data

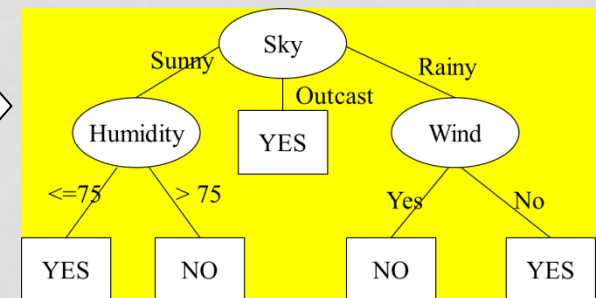
Sky	Temperature	Humidity	Wind	Tennis
Sunny	85	85	No	No
Sunny	80	90	Yes	No
Over cast	83	86	No	Yes
Rainy	70	96	No	Yes
Rainy	68	80	No	Yes
Over cast	64	65	Yes	Yes
Sunny	72	95	No	No
Sunny	69	70	No	Yes
Rainy	75	80	No	Yes
Sunny	75	70	Yes	Yes
Over cast	72	90	Yes	Yes
Over cast	81	75	No	Yes
Rainy	71	91	Yes	No

Training Data

Sky	Temperature	Humidity	Wind	Tennis
Sunny	60	65	No	?????

ML  
Algorithm

DECISION TREE



Model (Classifier)

Class = Yes

Prediction

Instances, examples

# MODELS (CLASSIFICATION): RULES

Attributes, features,  
Input variables,  
Independent variables

Label, class, output variable,  
dependent variable

Future data

Sky	Temperature	Humidity	Wind	Tennis
Sunn	85	85	No	No
Sunn	80	90	Yes	No
Over cast	83	86	No	Yes
Rainy	70	96	No	Yes
Rainy	68	80	No	Yes
Over cast	64	65	Yes	Yes
Sunn	72	95	No	No
Sunn	69	70	No	Yes
Rainy	75	80	No	Yes
Sunn	75	70	Yes	Yes
Over cast	72	90	Yes	Yes
Over cast	81	75	No	Yes
Rainy	71	91	Yes	No

Training Data

Sky	Temperature	Humidity	Wind	Tennis
Sunny	60	65	No	?????

ML  
Algorithm

RULES

**IF Sky = Sunny AND  
Humidity  $\leq$  75  
THEN Tennis = Yes ...**

Model (Classifier)

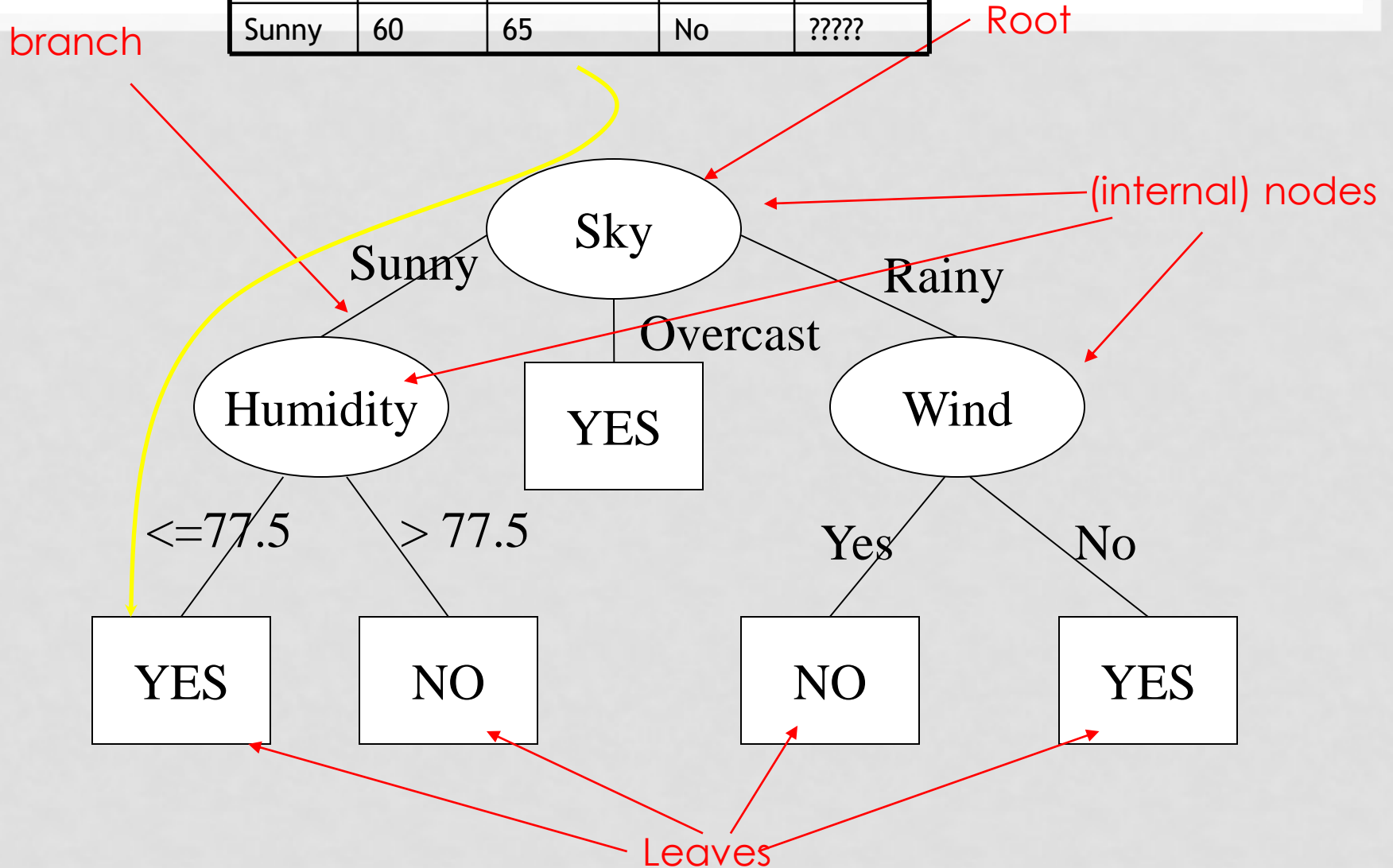
Class = Yes

Prediction

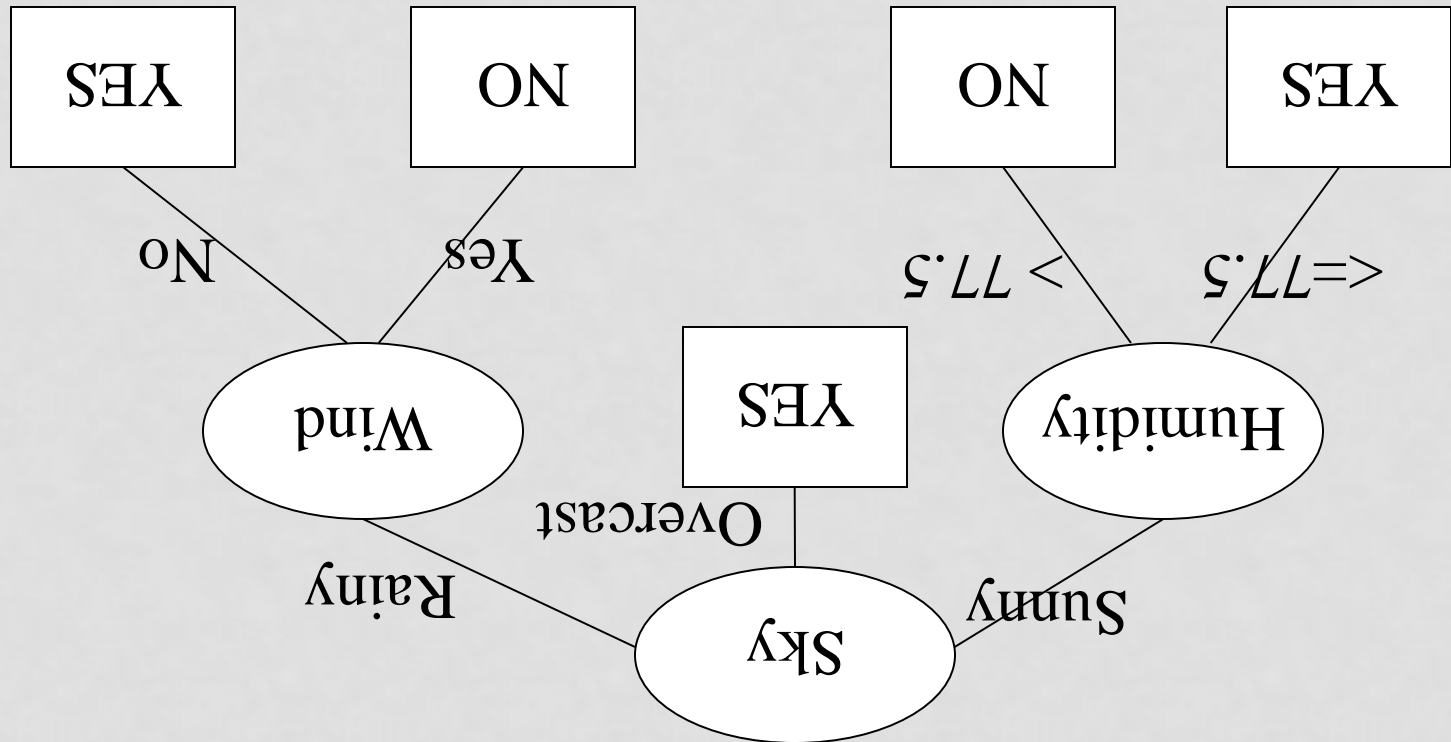
Instances, examples

# Decision trees

Sky	Temperature	Humidity	Wind	Tennis
Sunny	60	65	No	?????



# Decision trees

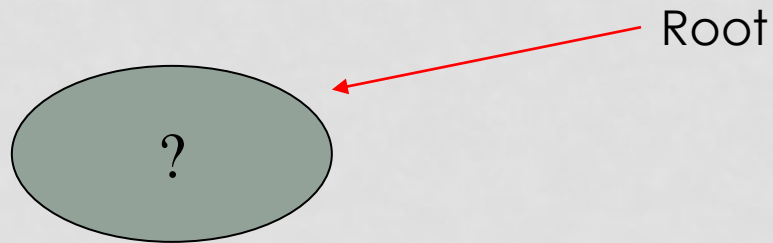




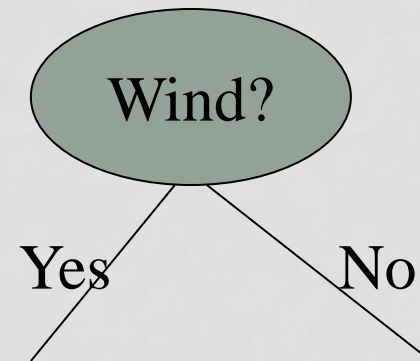
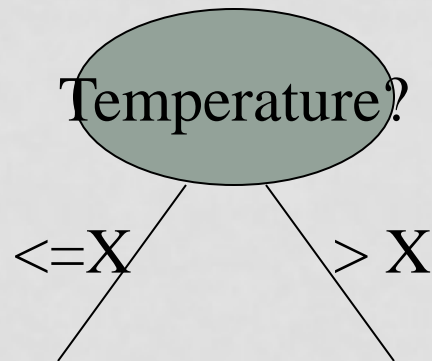
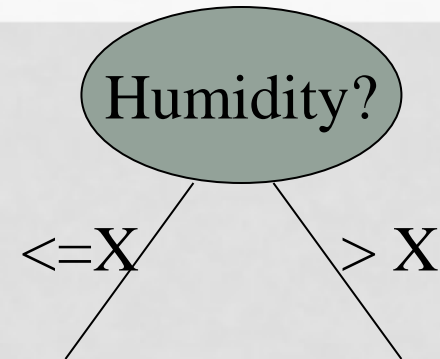
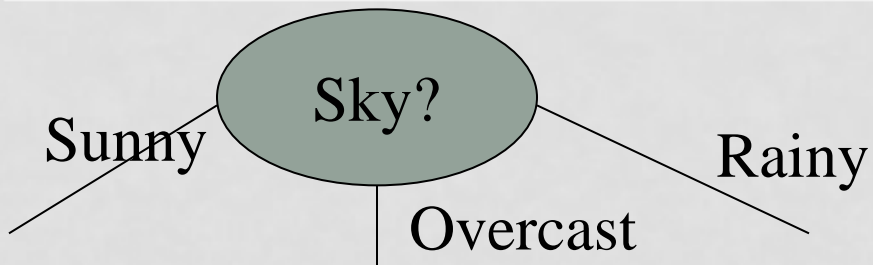
# DECISION TREE ALGORITHM

- There are many algorithms for training trees: ID3, C4.5 (J48), C5.0, CART, ...
- Here, the basic ideas for all of them will be explained:
  - Trees are constructed iteratively (iterative improvement), starting from the empty tree
  - The main step in the algorithm (repeated many times) is selecting the best feature for each tree node, using some criterion (such as entropy minimization).
  - Growing the tree stops when it is not possible to continue splitting the data or hyper-parameters force to stop the growth process.

THE CONSTRUCTION OF THE TREE STARTS  
WITH AN EMPTY TREE, AT THE ROOT



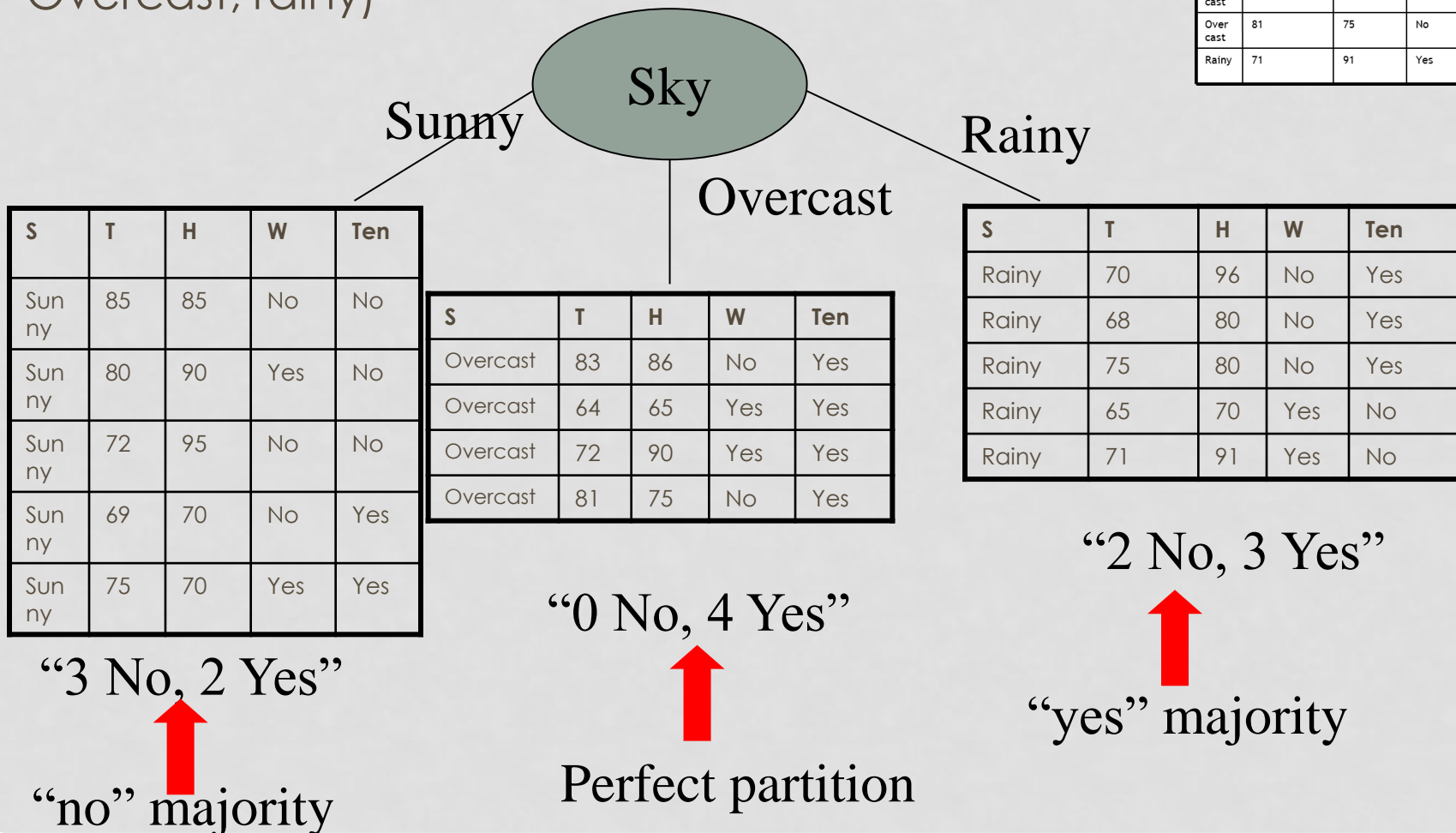
WHAT IS THE BEST ATTRIBUTE TO PUT IN THE  
ROOT OF THE TREE?



# Let's try with attribute SKY

Sky generates as many partitions as values (3: sunny, Overcast, rainy)

Sky	Temperature	Humidity	Wind	Tennis
Sunn	85	85	No	No
Sunn	80	90	Yes	No
Overcast	83	86	No	Yes
Rainy	70	96	No	Yes
Rainy	68	80	No	Yes
Overcast	64	65	Yes	Yes
Sunn	72	95	No	No
Sunn	69	70	No	Yes
Rainy	75	80	No	Yes
Sunn	75	70	Yes	Yes
Overcast	72	90	Yes	Yes
Overcast	81	75	No	Yes
Rainy	71	91	Yes	No



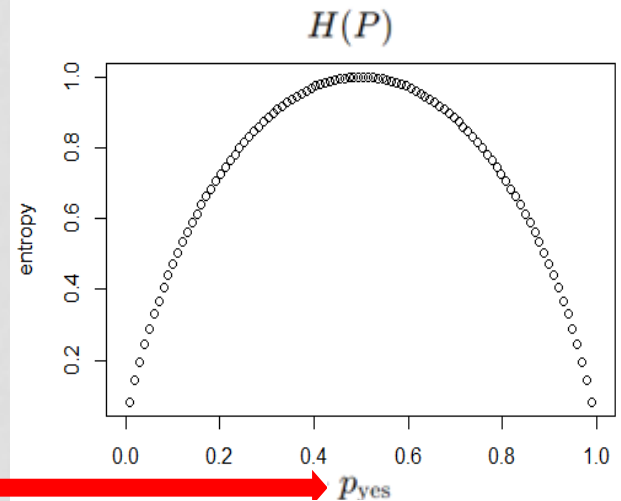
# How do we know if SKY is a good attribute?

- Perfect partition (homogeneous):
  - 0% No, 100% Yes
  - 100% No, 0% Yes
- Worst partition: 50% No, 50% Yes (completely heterogeneous)
- Entropy measures partition quality (the larger, the worse)

$$H(P) = - \sum_{C_i} p_{C_i} \log_2(p_{C_i})$$

$$H(P) = -(p_{\text{yes}} \log_2(p_{\text{yes}}) + p_{\text{no}} \log_2(p_{\text{no}}))$$

$$p_{\text{no}} = 1 - p_{\text{yes}}$$

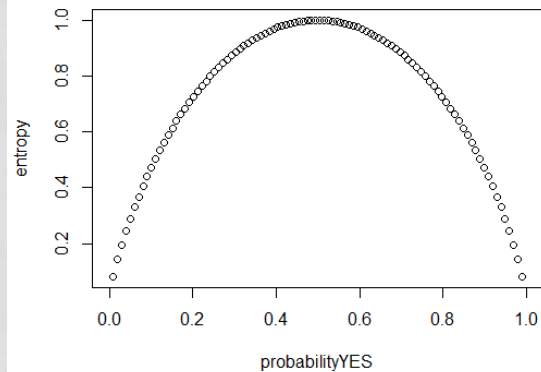


Proportion  
of Yes

# OTHER WAYS TO MEASURE PARTITION QUALITY

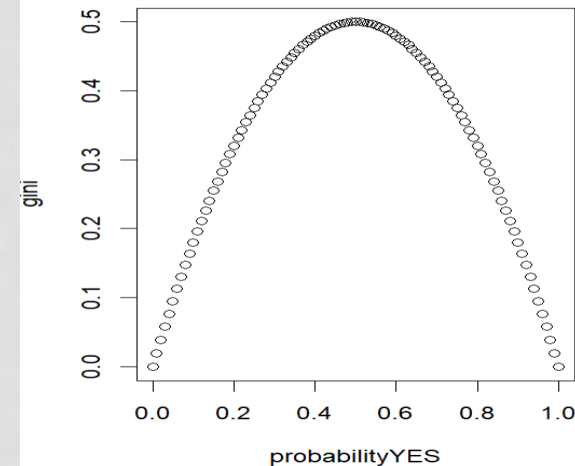
Entropy

$$H(P) = - \sum_{C_i} p_{C_i} \log_2(p_{C_i})$$

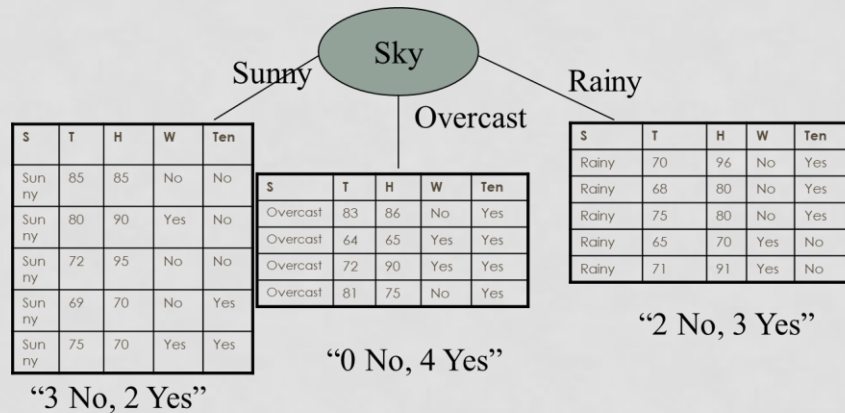


Gini

$$\text{Gini}(P) = \sum_{C_i} p_{C_i} (1 - p_{C_i})$$



# Average entropy for Sky



Entropy for the three partitions of Sky:

1. "3 No, 2 Yes":  $H(P_{\text{Sunny}}) = -((3/5) \cdot \log_2(3/5) + (2/5) \cdot \log_2(2/5)) = 0.97$
2. "0 No, 4 Yes":  $H(P_{\text{Overcast}}) = -((0/4) \cdot \log_2(0/4) + 1 \cdot \log_2(1)) = 0$
3. "2 No, 3 Yes":  $H(P_{\text{Rain}}) = -((2/5) \cdot \log_2(2/5) + (3/5) \cdot \log_2(3/5)) = 0.97$

(Weighted) Average Sky entropy:

- $H_{\text{Sky}} = (5/14) \cdot 0.97 + (4/14) \cdot 0 + (5/14) \cdot 0.97 = \mathbf{0.69}$
- Note: there are 14 instances in the data set

# ENTROPY AND INFORMATION GAIN

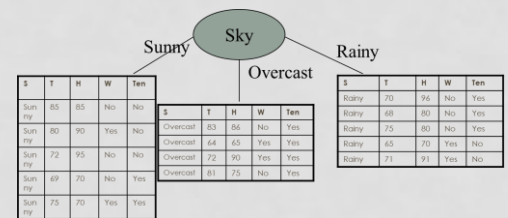
- Sometimes, instead of entropy ( $H$ ) (to minimize), information gain ( $IG$ ) is used (to be maximized)

- $IG$  is the difference between the entropy of the original partition  $H(P)$  and the weighted average entropy after using the attribute ( $H_{P_{Sky}}$ ).

- Maximizing  $IG$  is equivalent to minimizing entropy.

$P$

Cielo	Temp	Hum	Viento	Tenis
sol	85	85	no	no
sol	80	90	sl	no
nubes	83	86	no	sl
lluvia	70	96	no	sl
lluvia	68	80	no	sl
lluvia	65	70	sl	no
nubes	64	65	sl	sl
sol	72	95	no	no
sol	69	70	no	sl
lluvia	75	80	no	no
sol	75	70	sl	sl
nubes	72	90	sl	sl
nubes	81	75	no	sl
lluvia	71	91	sl	no



$P_1$

$P_2$

$P_3$

$$IG_{Sky} = H(P) - H_{Sky}(P_1, P_2, P_3)$$



# ENTROPY AND INFORMATION GAIN

- Choosing the attribute with highest IG is equivalent to choosing the attribute with smallest entropy, because  $H(P)$  is the same for all attributes.

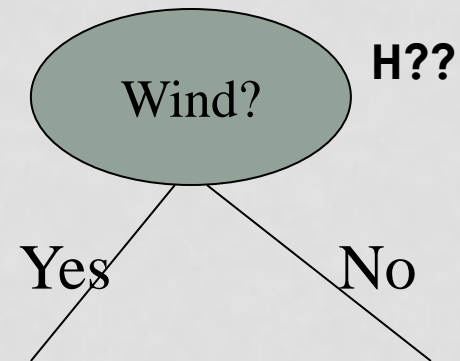
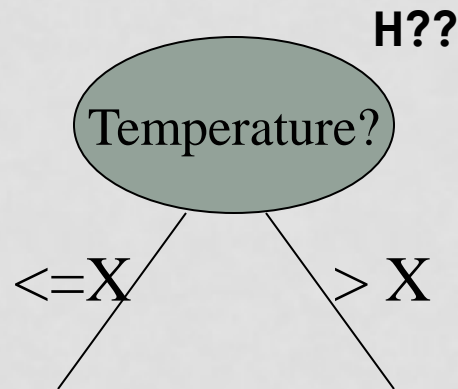
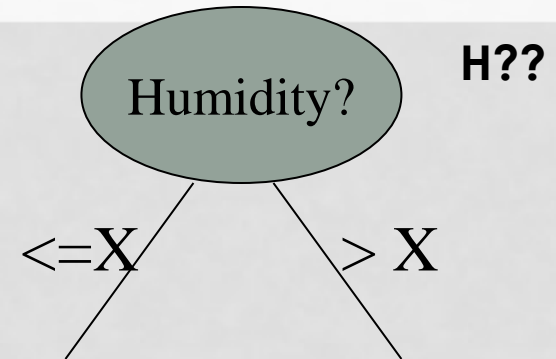
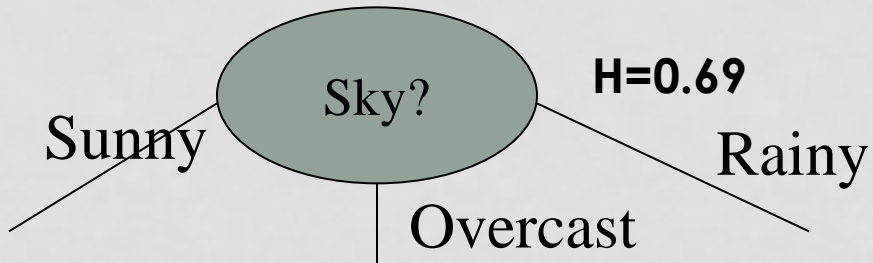
$$IG_{\text{Sky}} = H(P) - H_{\text{Sky}}(P_1, P_2, P_3)$$

$$IG_{\text{Humidity}} = H(P) - H_{\text{Humidity}}(P_1, P_2)$$

$$IG_{\text{Temperature}} = H(P) - H_{\text{Temperature}}(P_1, P_2)$$

$$IG_{\text{Wind}} = H(P) - H_{\text{Wind}}(P_1, P_2)$$

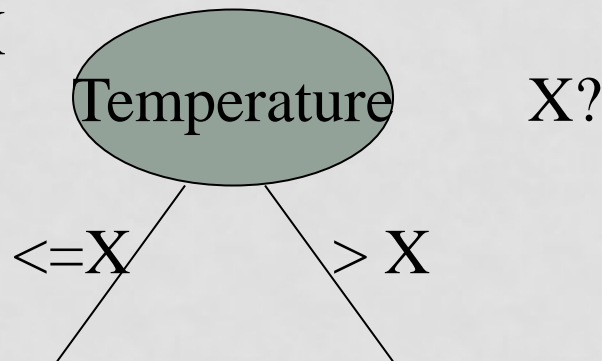
# WHAT IS THE BEST ATTRIBUTE TO PUT IN THE ROOT OF THE TREE?



# WHAT TO DO WITH CONTINUOUS ATTRIBUTES?

A binary (two-values) attribute is created by computing a threshold  $X$

Note: only some thresholds are shown. The best one is  $X=84$  with average entropy = 0.83



Sky	Temperature	Humidity	Windy	Play
sunny	85	85	FALSE	no
sunny	80	90	TRUE	no
overcast	83	86	FALSE	yes
rainy	70	96	FALSE	yes
rainy	68	80	FALSE	yes
rainy	65	70	TRUE	no
overcast	64	65	TRUE	yes
sunny	72	95	FALSE	no
sunny	69	70	FALSE	yes
rainy	75	80	FALSE	yes
sunny	75	70	TRUE	yes
overcast	72	90	TRUE	yes
overcast	81	75	FALSE	yes
rainy	71	91	TRUE	no

64 – Yes, 65-No, 68 – Yes, 69 – Yes, 70 – Yes, 71 – No, 72 – YesNo, 75 – YesYes, 80 – No, 81 – Yes, 83 – Yes, 85 - No

$H = 0.83$

4 No, 9 Yes

$X=84$

1 No, 0 Yes

64 – Yes, 65-No, 68 – Yes, 69 – Yes, 70 – Yes, 71 – No, 72 – YesNo, 75 – YesYes, 80 – No, 81 – Yes, 83 – Yes, 85 - No

$H = 0.93$

2 No, 4 Yes

$X=71.5$

3 No, 5 Yes

64 – Yes, 65-No, 68 – Yes, 69 – Yes, 70 – Yes, 71 – No, 72 – YesNo, 75 – YesYes, 80 – No, 81 – Yes, 83 – Yes, 85 - No

$H = 0.89$

1 No, 4 Yes

$X=70.5$

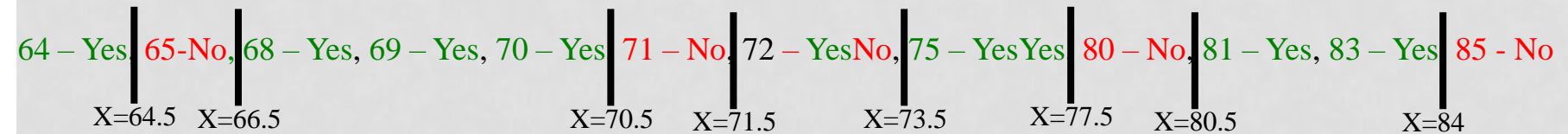
4 No, 5 Yes

?

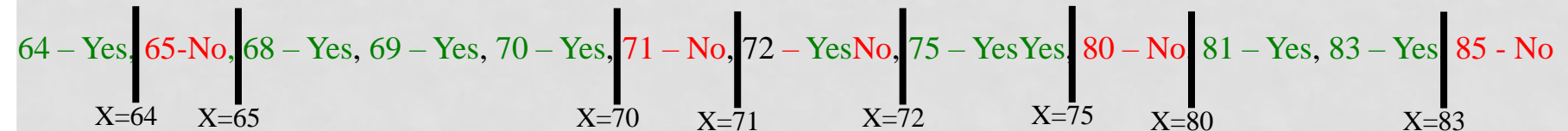
- Should **all** possible thresholds be actually tried? Or can we make the algorithm more efficient by ignoring some possible thresholds?

# ALTERNATIVE POSSIBLE THRESHOLDS

Possible thresholds are transitions from Yes to No, or from No to Yes:

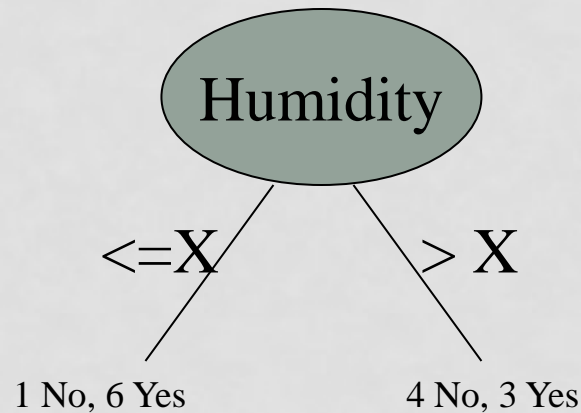


- The actual threshold may depend on the algorithm implementation. Some implementations use the average:  $E_j: 64.5 = (64+65)/2$ .
- Other implementations use the maximum of the left partition. In that case, the possible thresholds would have been:



- Notice that entropy computed with the training data is the same in both cases, because in the two cases data is partitioned in the same way.

# HUMIDITY



65-Yes, 70-No, 75-Yes, 80-Yes, 85-No, 86-Yes, 90-No, 91-No, 95-No, 96-Yes

1 No, 6 Yes      X=82.5      4 No, 3 Yes

$H = 0.79$

Note: there are other alternatives for the threshold, but this is the best one (minimum entropy)

# WHAT IS THE BEST NODE TO PUT IN THE ROOT OF THE TREE?

$H = 0.69$

Sunny

Rainy

Overcast

3 No, 2 Yes

0 No, 4 Yes

2 No, 3 Yes

$H = 0.83$

Temperatura

$\leq 84$

$> 84$

4 No, 9 Yes

1 No, 0 Yes

$H = 0.79$

Humidity

$\leq 82.5$

$> 82.5$

1 No, 6 Yes

4 No, 3 Yes

$H = 0.89$

Wind

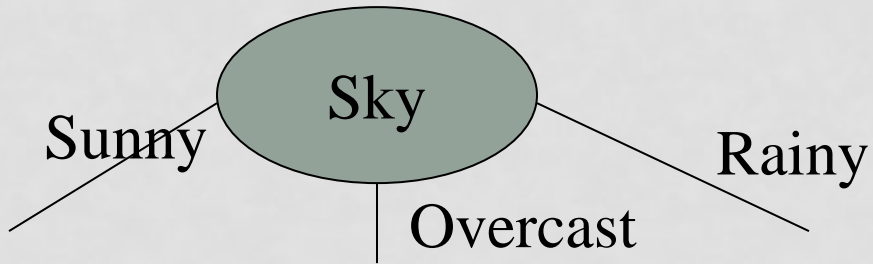
Yes

No

3 No, 3 Yes

2 No, 6 Yes

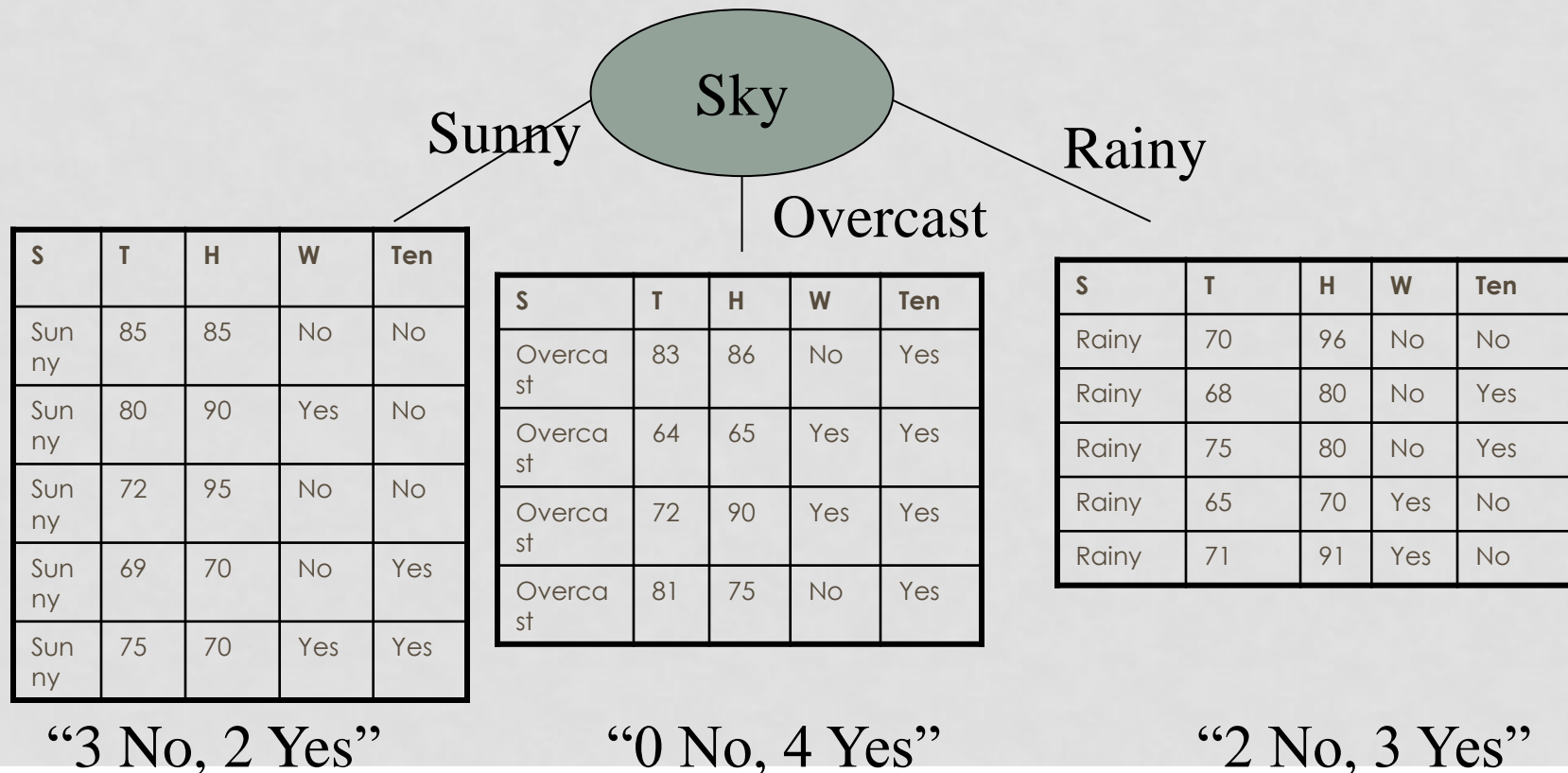
WHAT IS THE BEST NODE TO PUT IN  
THE ROOT OF THE TREE?





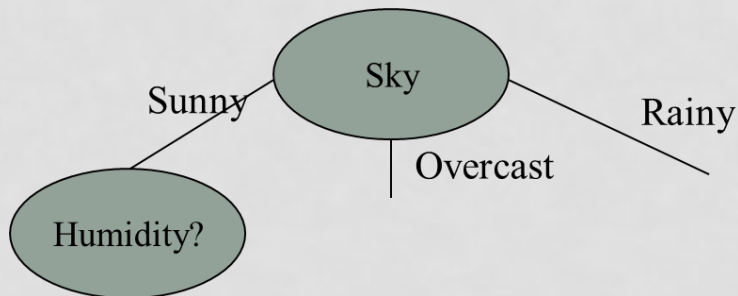
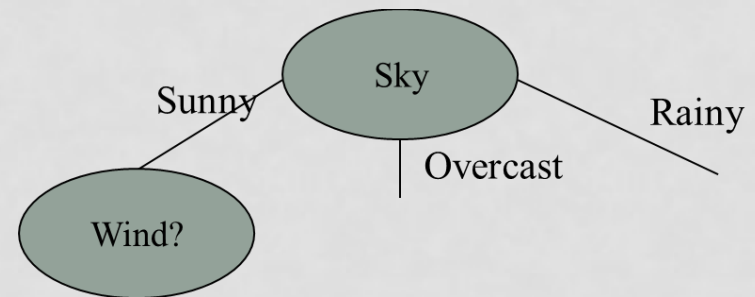
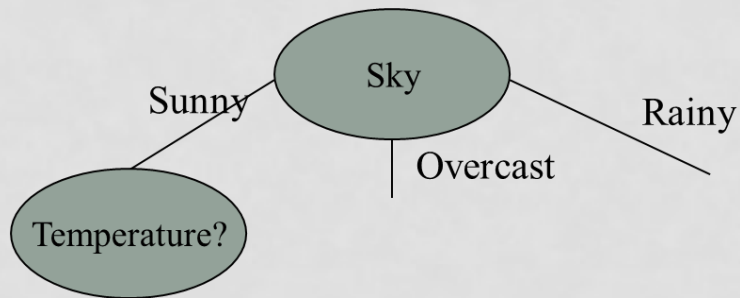
# RECURSIVE TREE GROWTH

Now that the attribute for the root node has been determined, the process continues recursively. Now, the algorithm has to construct three new subtrees.



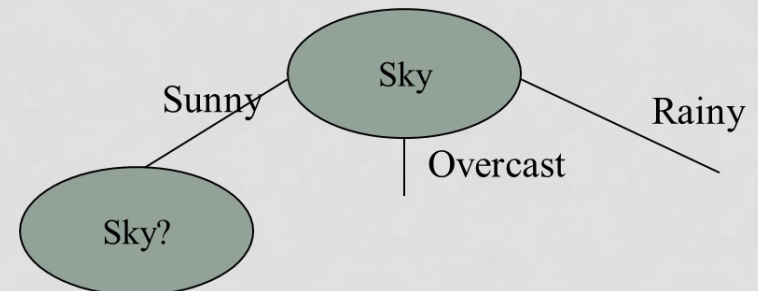
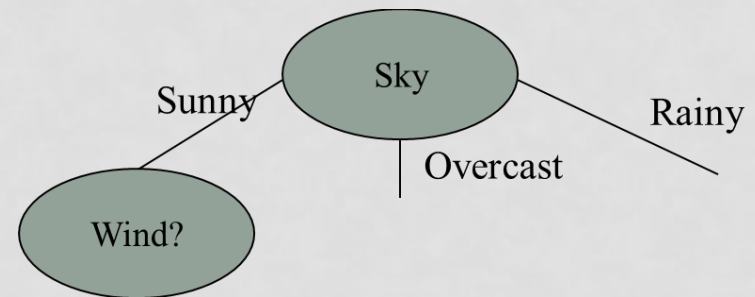
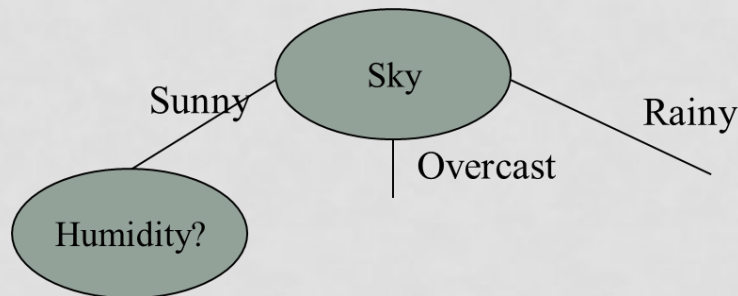
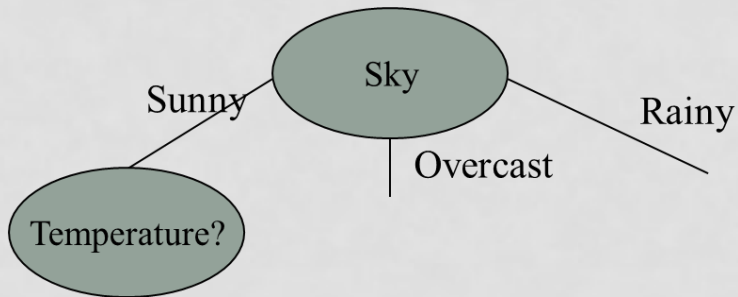
# RECURSIVE TREE GROWTH

Now that the attribute for the root node has been determined, the process continues recursively. Now, the algorithm has to construct three new subtrees.



# RECURSIVE TREE GROWTH

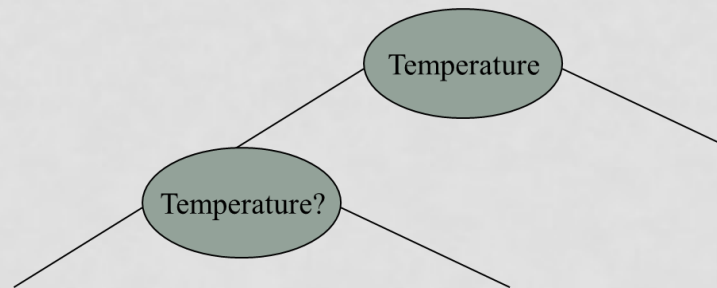
Now that the attribute for the root node has been determined, the process continues recursively. Now, the algorithm has to construct three new subtrees.



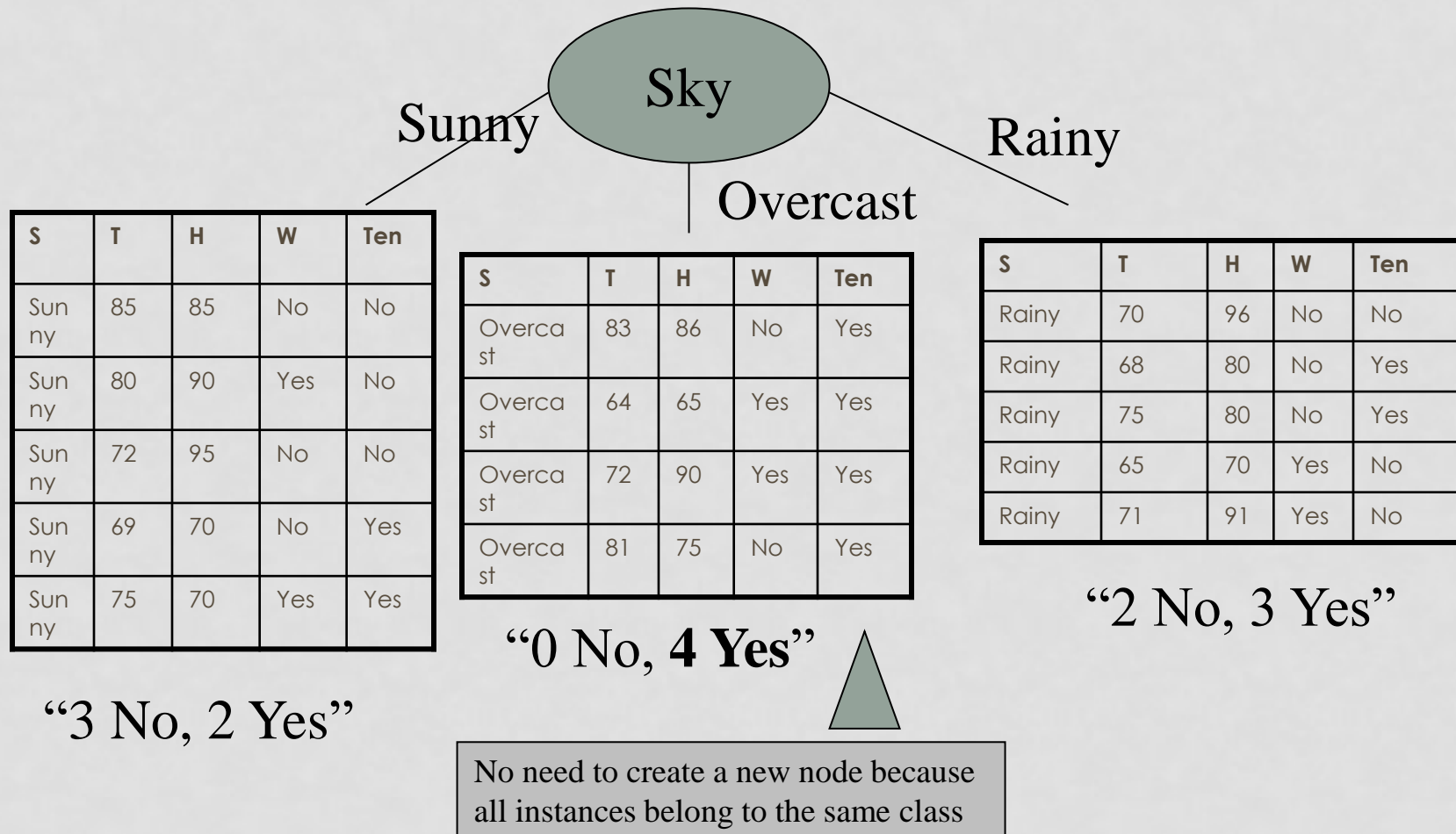
BTW, would this make sense?

?

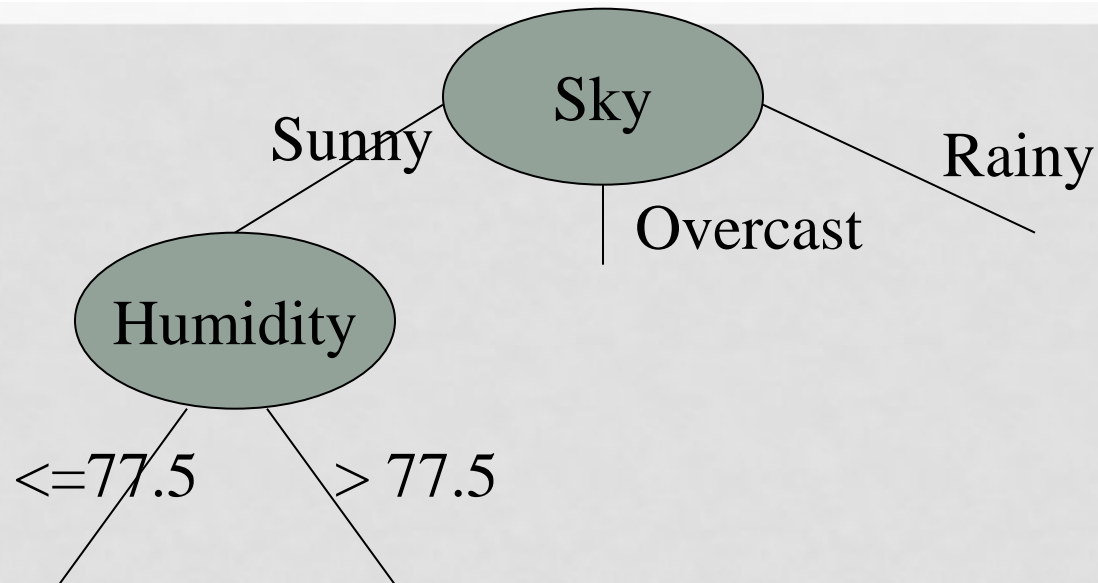
- If Temperature had been selected for the root, would it make sense to try (again) attribute Temperature?



# WHEN TO STOP SPLITTING PARTITIONS?



# WHY TO STOP TREE GROWTH?



T	H	V	Ten
69	70	No	Yes
75	70	Yes	Yes

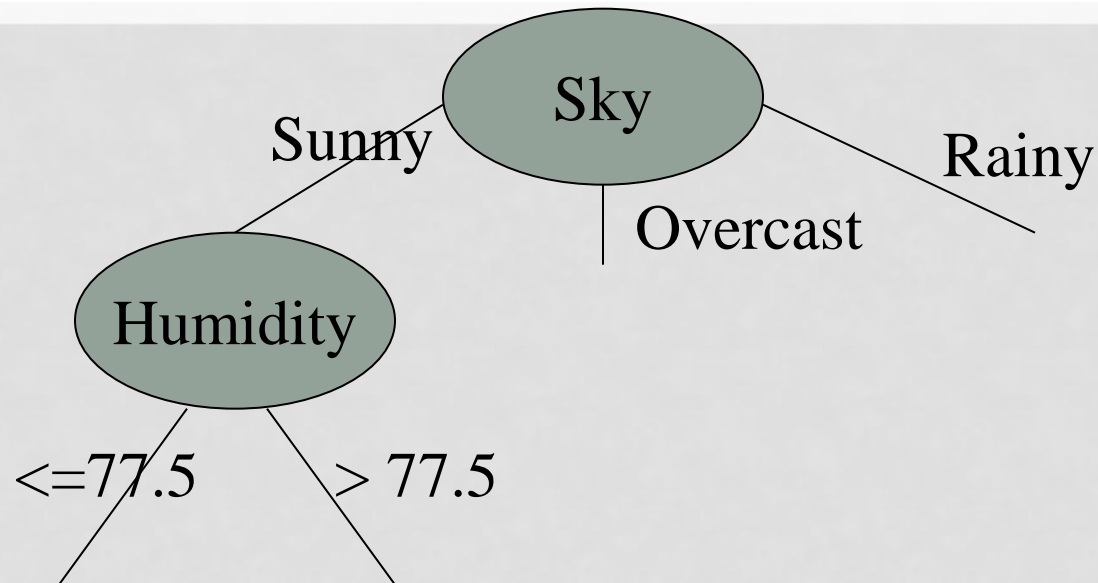
**“2 Yes, 0 No”**

T	H	V	Ten
85	85	No	No
80	90	Yes	No
72	95	No	No

**“3 No, 0 Yes”**

No need to create a new node because  
all instances belong to the same class

# WHY TO STOP TREE GROWTH?



T	H	V	Ten
69	70	No	Yes
75	70	Yes	Yes

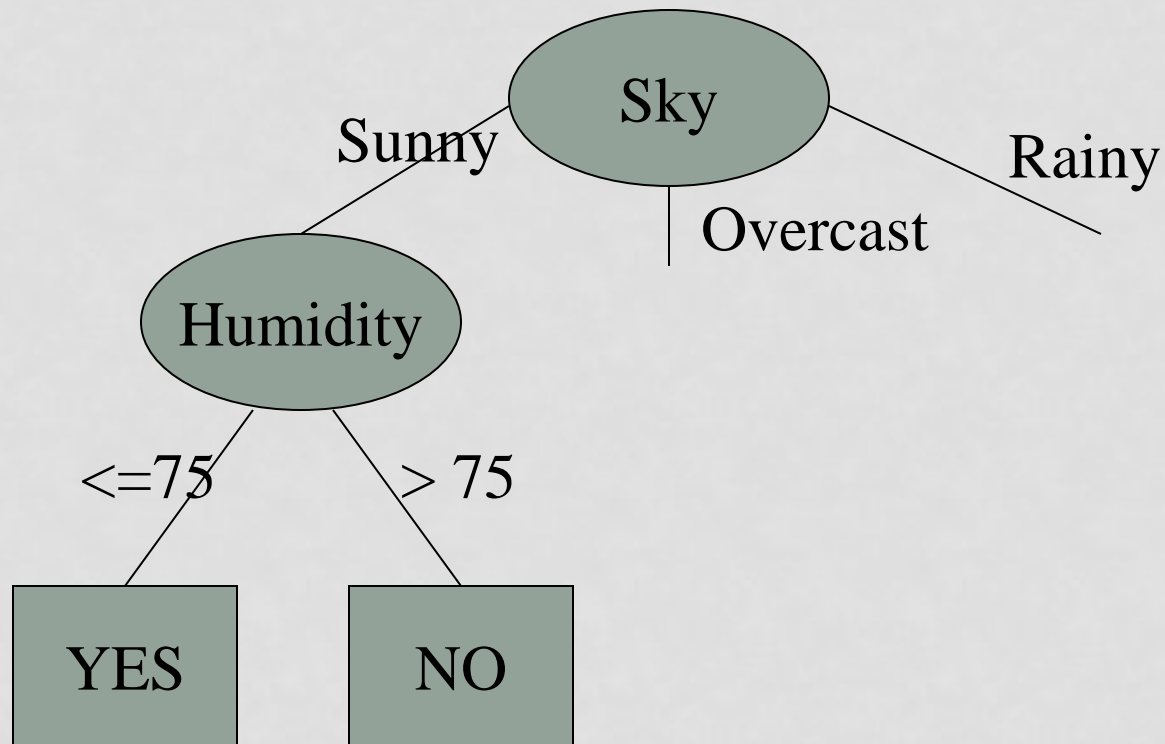
**“2 Yes, 0 No”**

T	H	V	Ten
85	85	No	No
80	90	Yes	No
72	95	No	No

**“3 No, 0 Yes”**

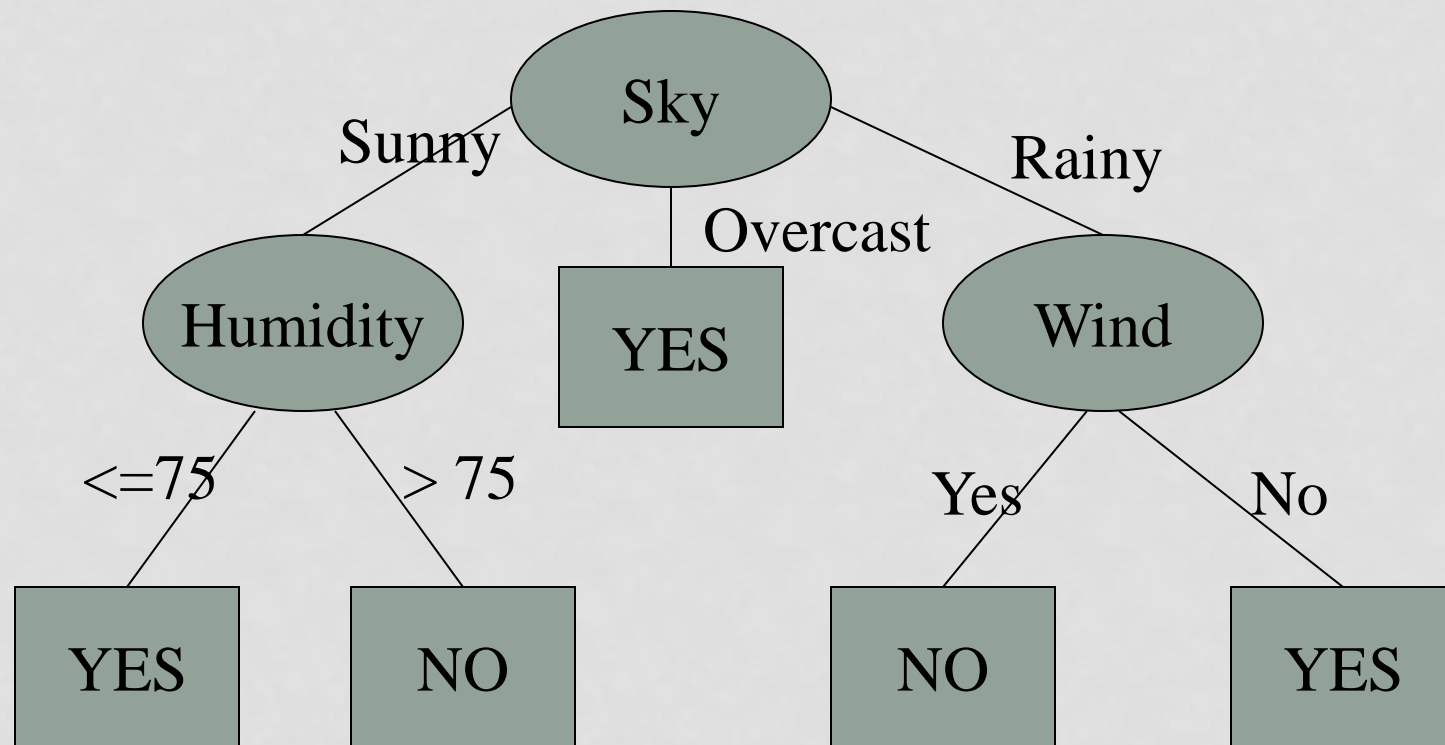
No need to create a new node because  
all instances belong to the same class

# RECURSIVE TREE GROWTH



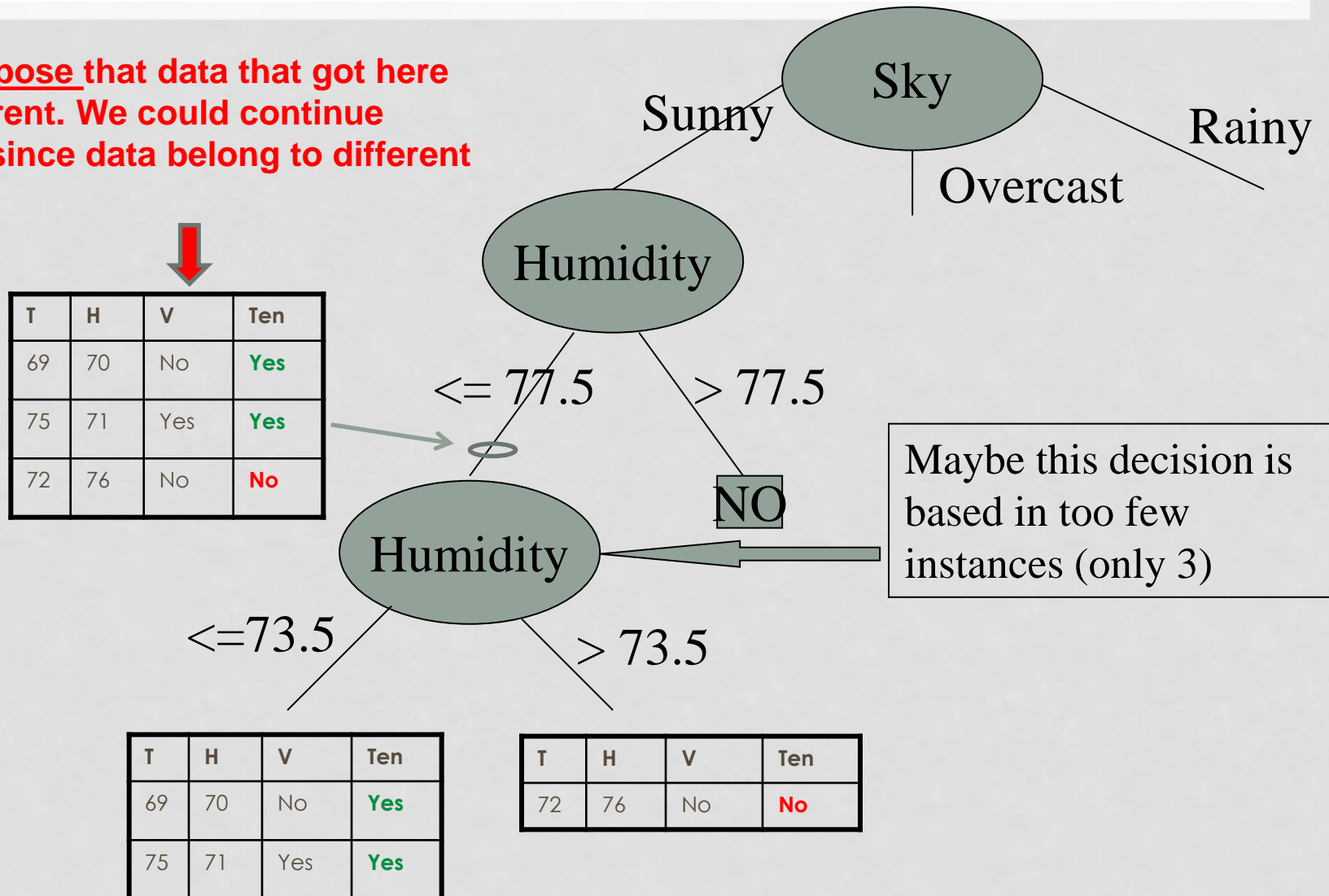


# RECURSIVE TREE GROWTH

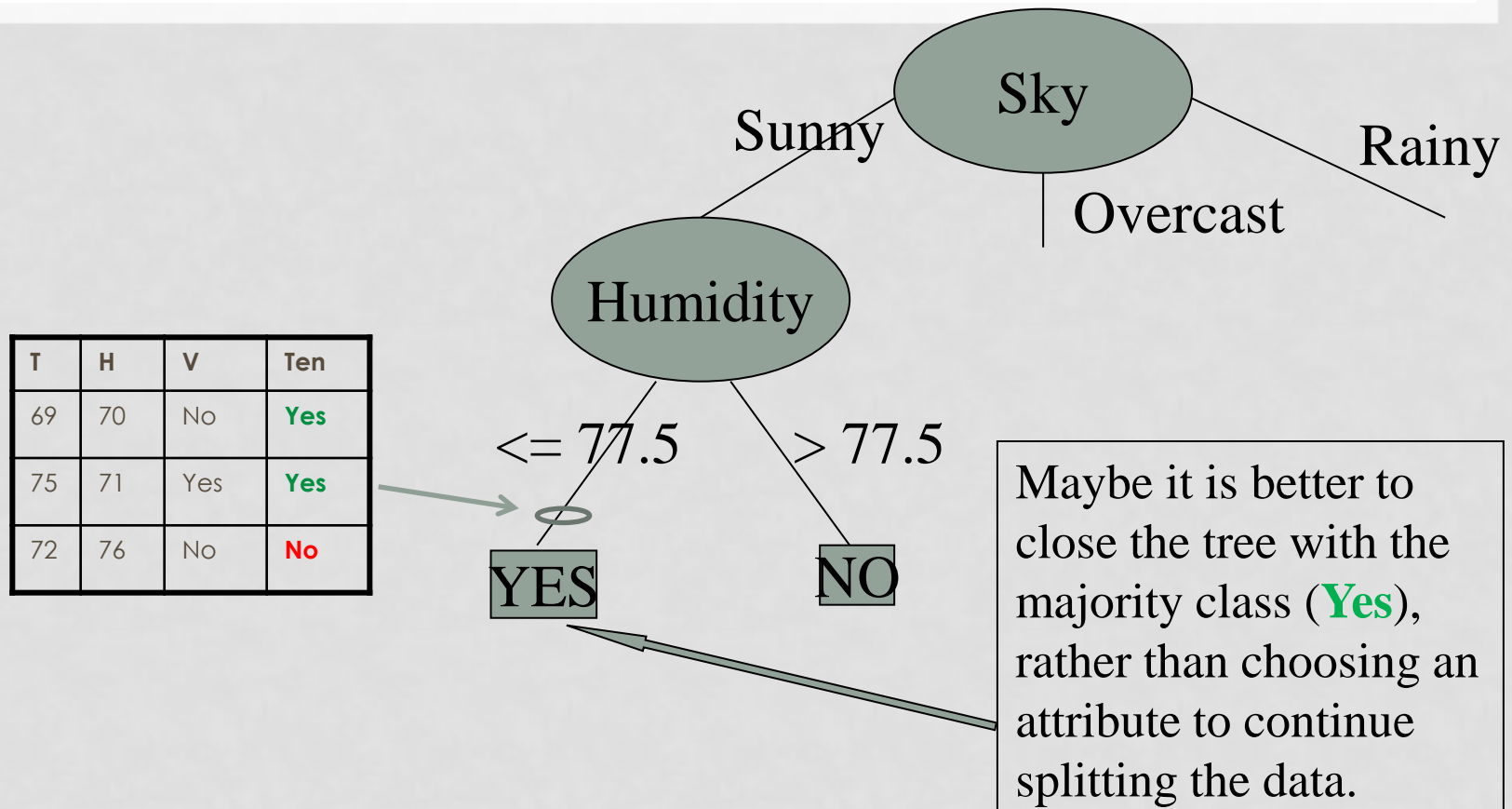


# WHY TO STOP TREE GROWTH?

**Let's suppose that data that got here was different. We could continue splitting since data belong to different classes.**



# WHY TO STOP TREE GROWTH?



- The algorithm may use a statistical criterion in order to determine whether it is worth to continue tree growth, whether the sample is too small, etc.
- But stopping tree growth is usually controlled with the **min-samples** hyperparameter (or **max-depth**).

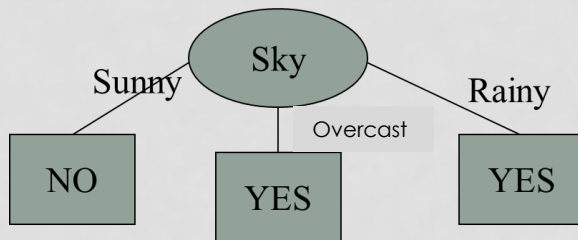
# MAIN HYPER-PARAMETERS OF DECISION TREES

- **max\_depth**: maximum depth of the tree
- **min\_samples\_split**: minimum number of instances in order to continue subdividing the tree

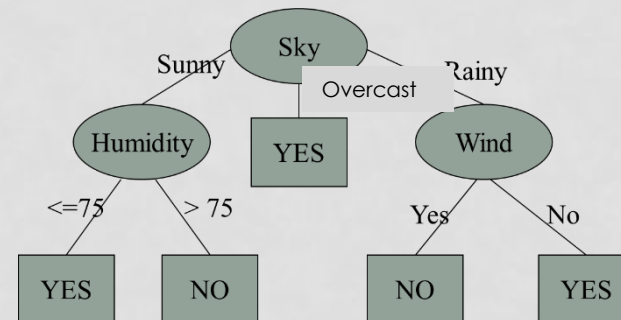
# MAIN HYPER-PARAMETERS OF DECISION TREES

- **max\_depth**: maximum depth of the tree

max\_depth = 1



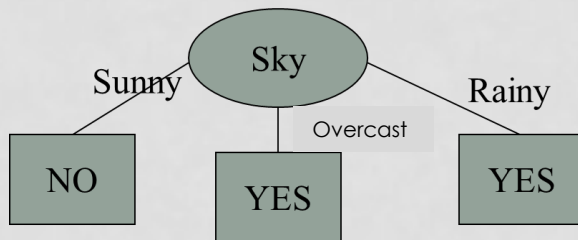
max\_depth = 2



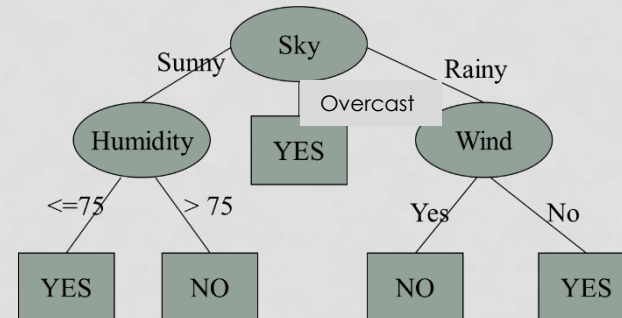
# MAIN HYPER-PARAMETERS OF DECISION TREES

- **min\_samples\_split**: minimum number of instances in order to continue subdividing the tree

min\_samples\_split = 6



min\_samples\_split = 5



```
graph TD; Sky([Sky]) -- Sunny --> T1[Table 1]; Sky -- Overcast --> T2[Table 2]; Sky -- Rainy --> T3[Table 3];
```

Three data tables corresponding to the branches of the decision tree for min\_samples\_split = 5:

S	T	H	W	Ten	
Sun	ny	85	85	No	No
Sun	ny	80	90	Yes	No
Sun	ny	72	95	No	No
Sun	ny	69	70	No	Yes
Sun	ny	75	70	Yes	Yes

S	T	H	W	Ten
Overcast	83	86	No	Yes
Overcast	64	65	Yes	Yes
Overcast	72	90	Yes	Yes
Overcast	81	75	No	Yes

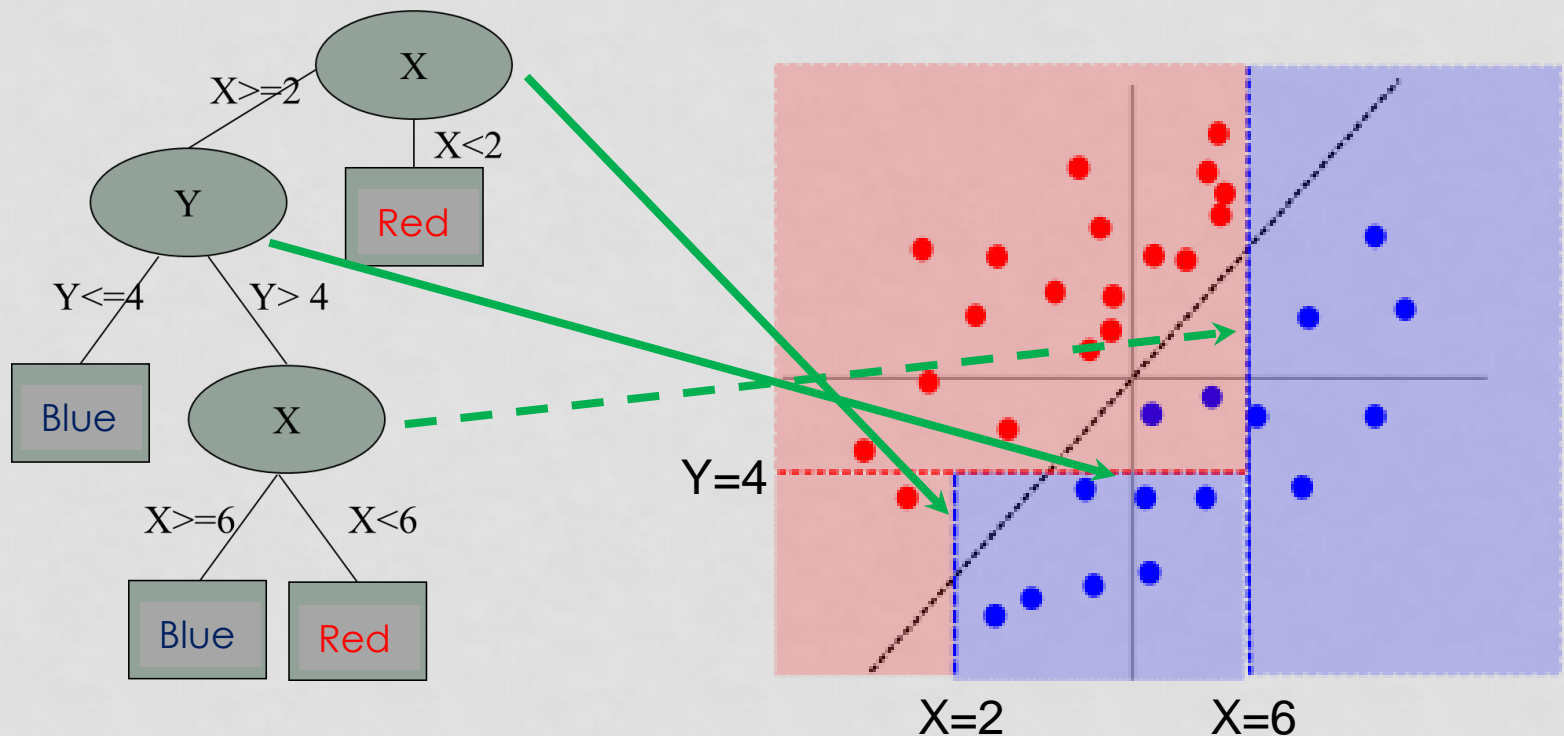
S	T	H	W	Ten
Rainy	70	96	No	No
Rainy	68	80	No	Yes
Rainy	75	80	No	Yes
Rainy	65	70	Yes	No
Rainy	71	91	Yes	No

# MAIN HYPER-PARAMETERS OF DECISION TREES

- **max\_depth**: maximum depth of the tree
- **min\_samples\_split**: minimum number of instances in order to continue subdividing the tree
- Both hyper-parameters (and hyper-parameters in general) control the complexity of the model (in feature-space, the complexity of the boundary)

# TREE BOUNDARIES IN FEATURE SPACE

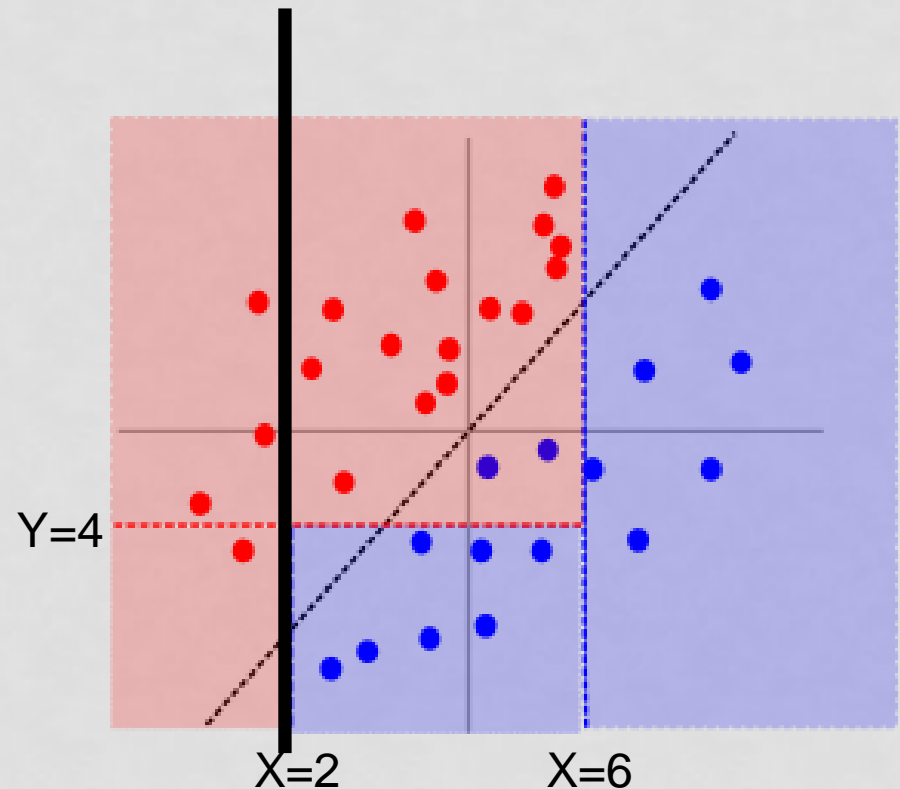
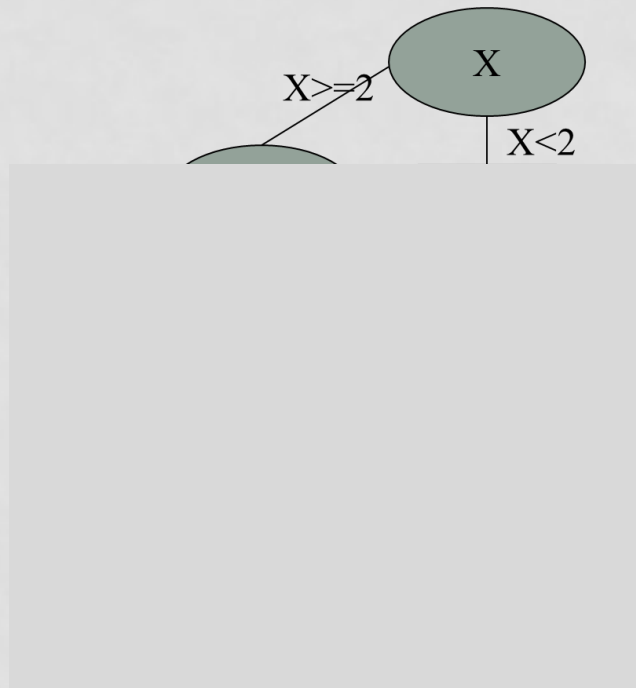
- Decision trees are non-linear. Boundary is made of piece-wise linear segments parallel to axis.
- Therefore, not very good for oblique boundaries (there are “oblique trees”, but not widely used)





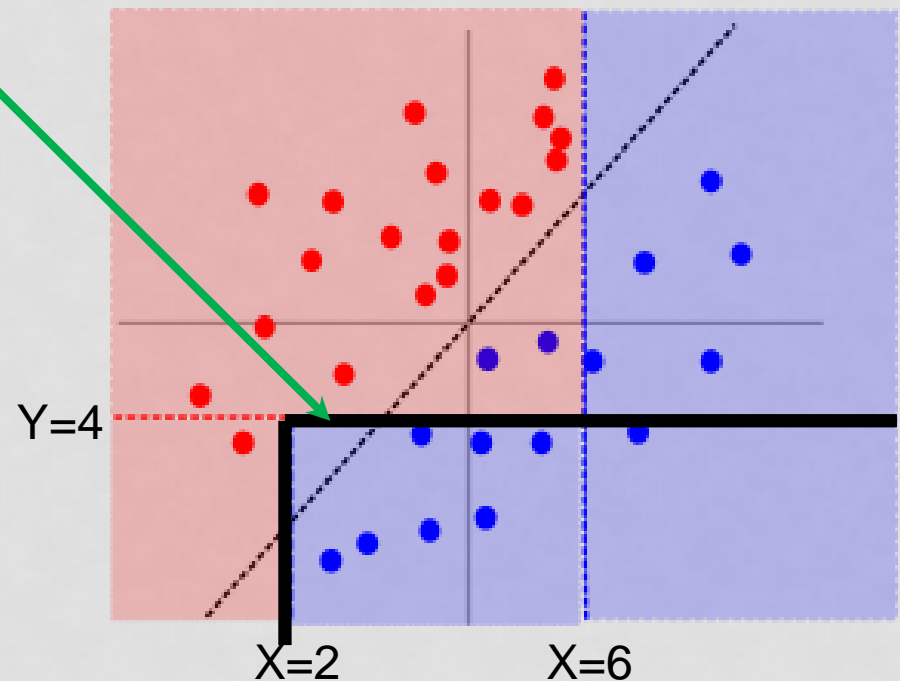
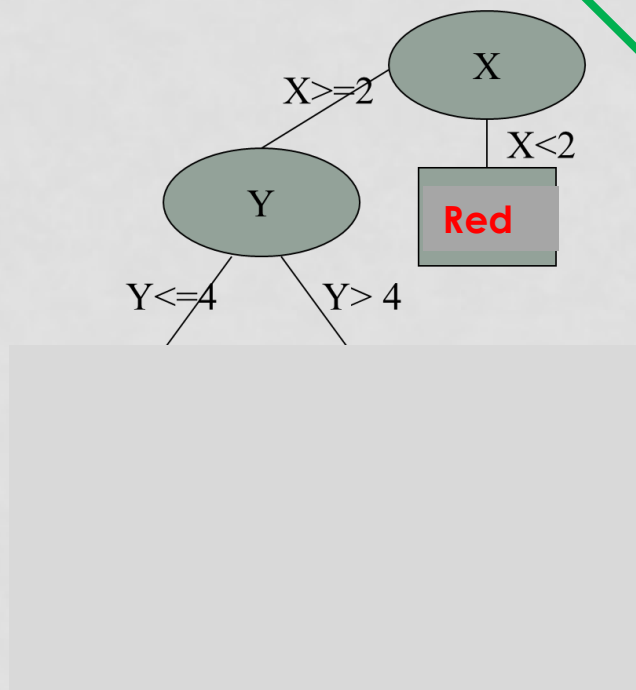
# MAX-DEPTH HYPER-PARAMETER FOR DECISION TREES

- With  $\text{max\_depth} = 1$ , boundary is a line.



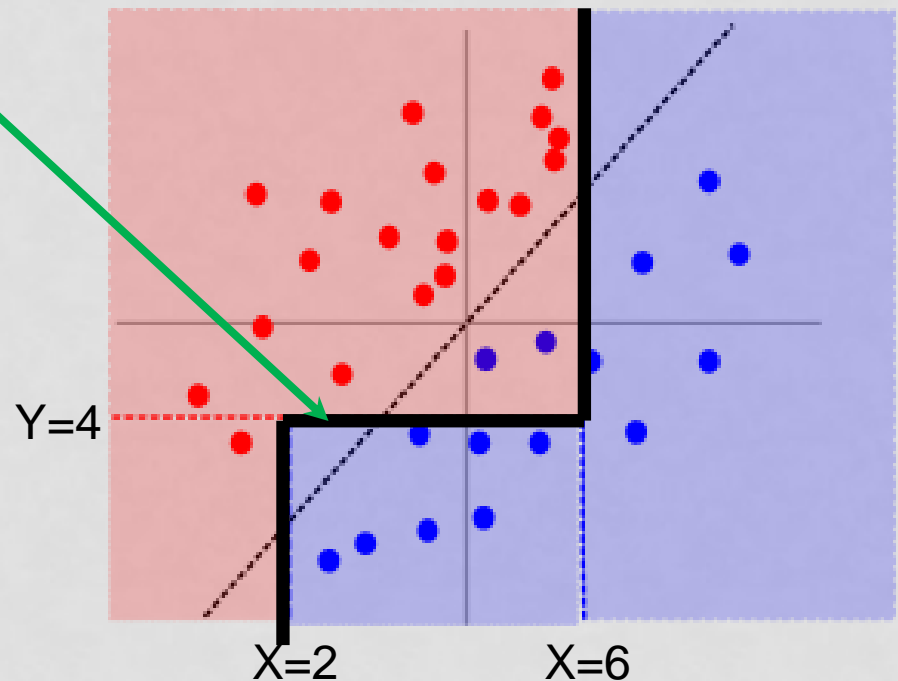
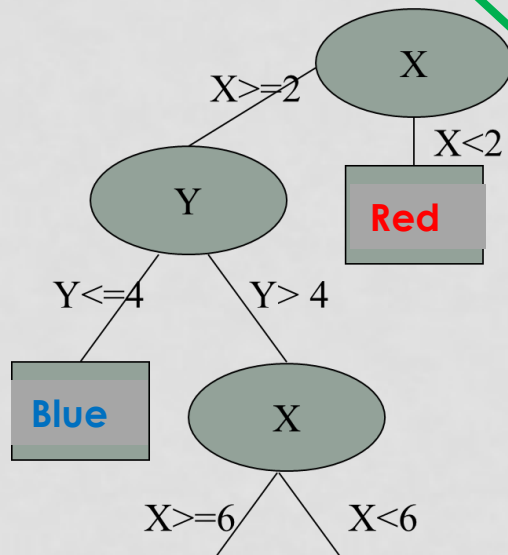
# MAX-DEPTH HYPER-PARAMETER FOR DECISION TREES

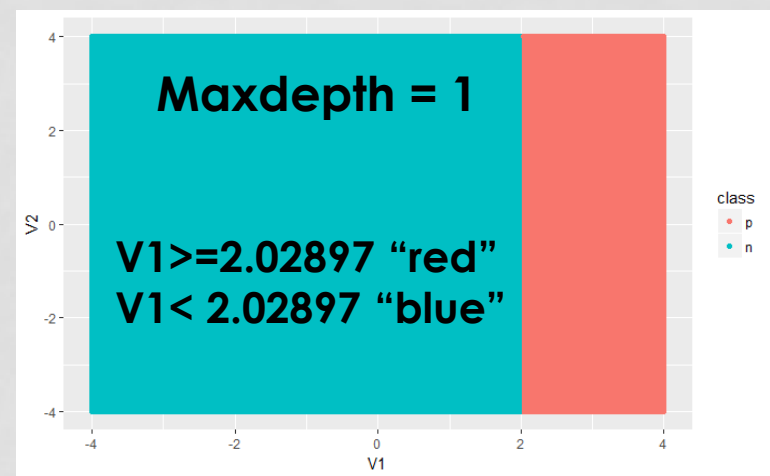
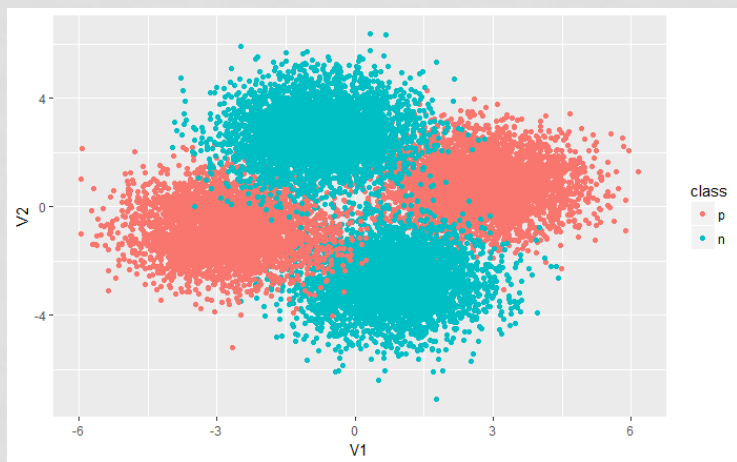
- With `max_depth = 2`, boundary is non-linear



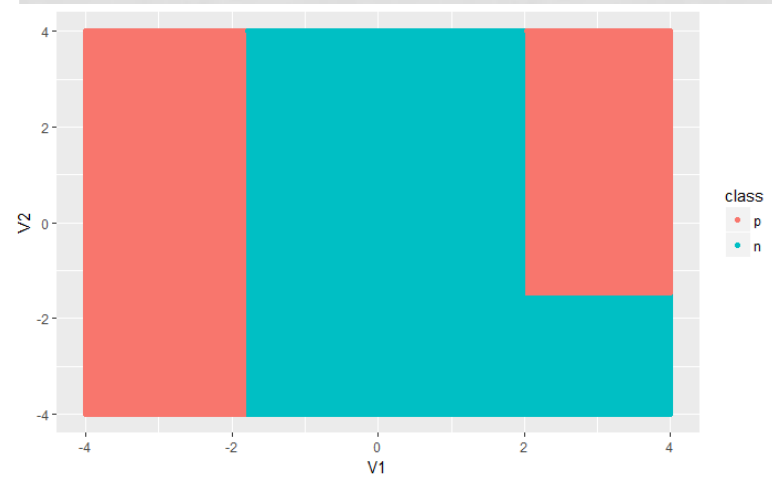
# MAX-DEPTH HYPER-PARAMETER FOR DECISION TREES

- With  $\text{max\_depth} = 3$ , boundary is non-linear and more complex than with  $\text{max\_depth} = 2$
- And so on



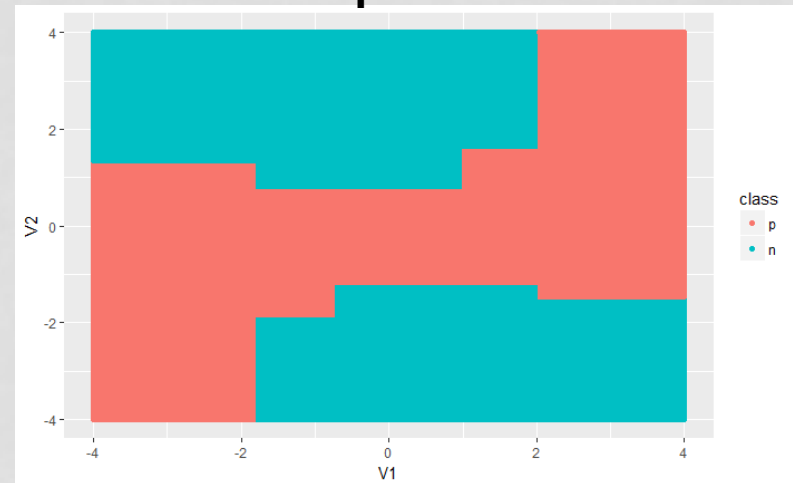


**Maxdepth = 2**



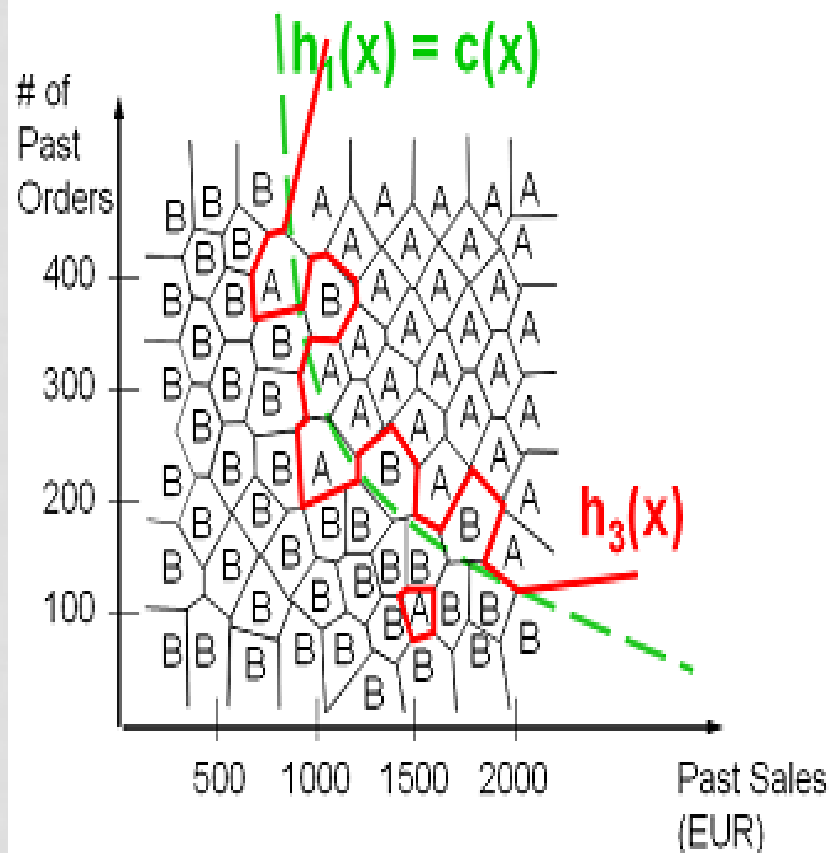
$V1 \geq 2.02897$  "red"  
 $V2 \geq -1.474671$  "red"  
 $V2 < -1.474671$  "blue"  
 $V1 < -1.797563$  "blue"  
 6)  $V1 < -1.797563$  "red"  
 7)  $V1 \geq -1.797563$  "blue"

**Maxdepth = 8**

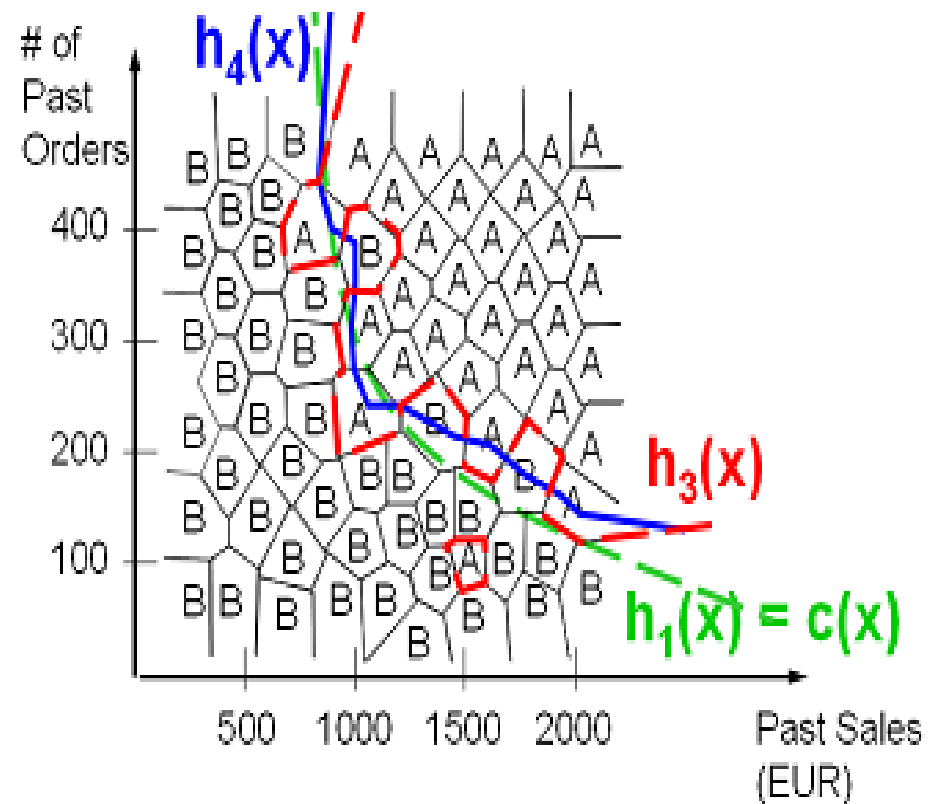


Usually, hyper-parameters control the **complexity** of the model

# FOR KNN, LARGE K = SIMPLER MODEL



(a) 1-NN on noisy data



(b) 3-NN and noisy data

# Rules (created from the decision tree)

Obtain one rule from each path from the root to the leaves

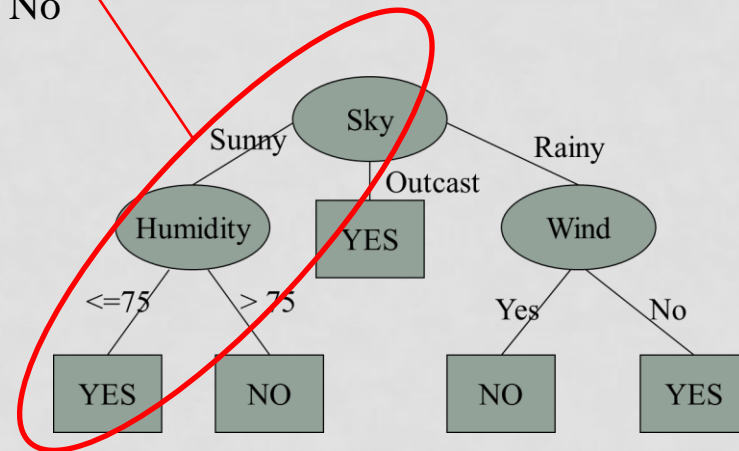
**IF Sky = Sunny AND Humidity  $\leq 75$  THEN Tennis = Yes**

**ELSE IF Sky = Sunny AND Humidity  $> 75$  THEN Tennis = No**

**ELSE IF Sky = Overcast THEN Tennis = Yes**

**ELSE IF Sky = Rainy AND Wind = Yes THEN Tennis = Yes**

**ELSE Tennis = No**

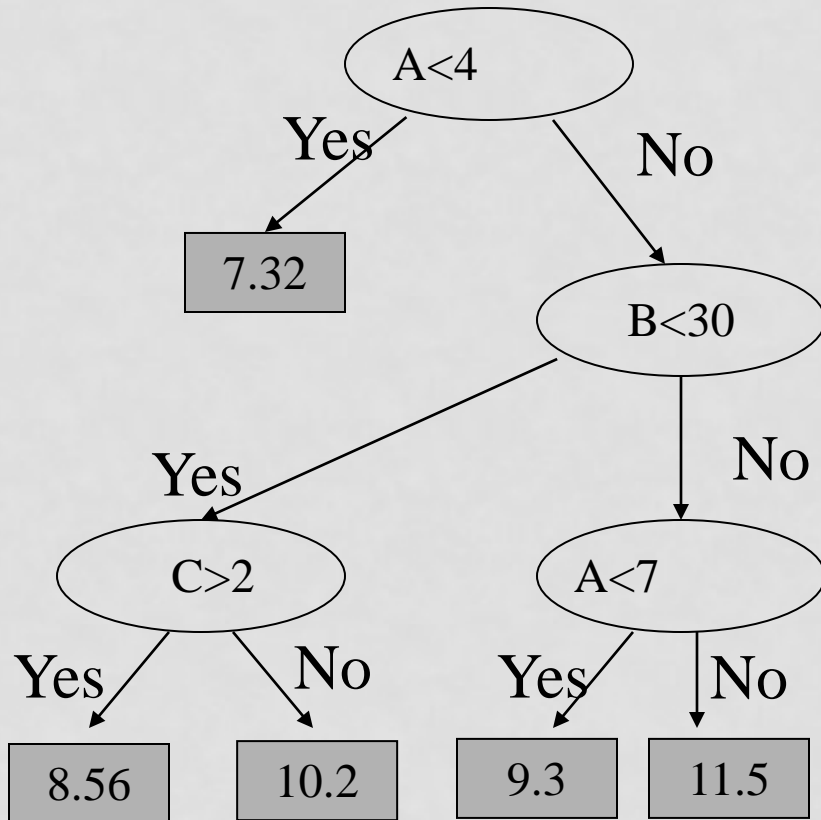


But there are algorithms that build rules directly from data

# TREES FOR REGRESSION

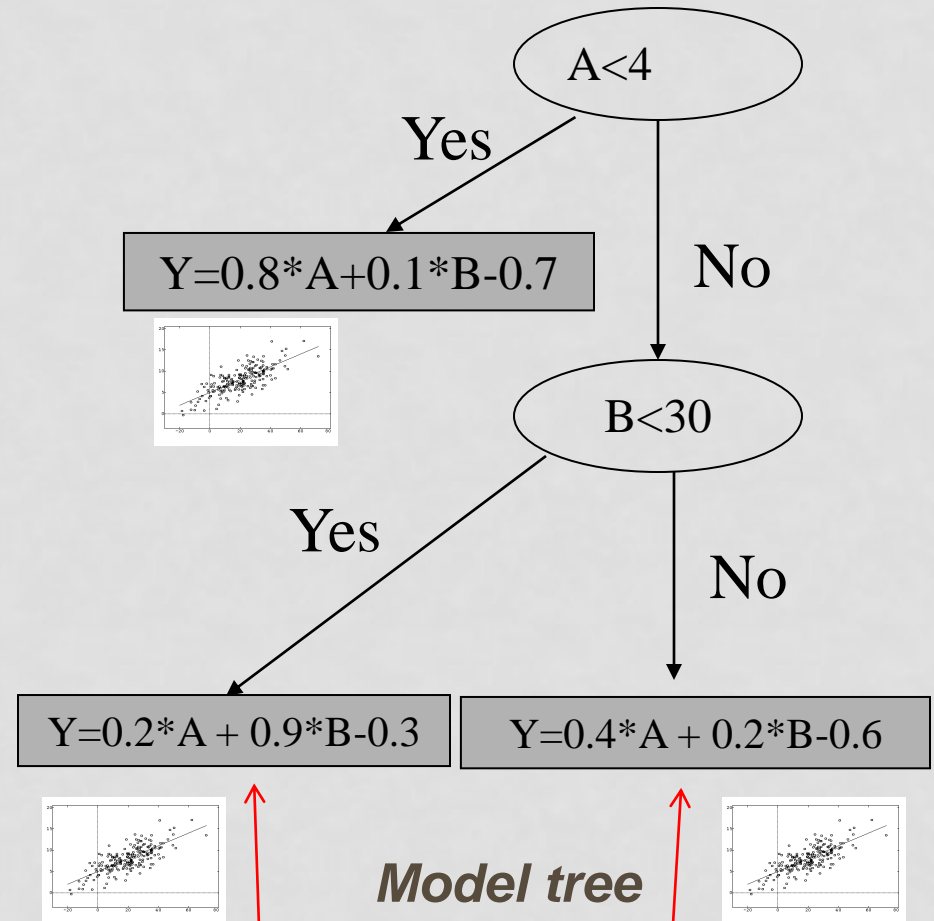
- What to do if the “class” (response variable) is continuous (rather than categorical)?
  - Answer: variance reduction (instead of entropy reduction)
- Two types:
  - Model trees
  - Regression trees

# TREES FOR REGRESSION: TWO TYPES



**Regression tree**

Constants



**Model tree**

Linear models in the leaves



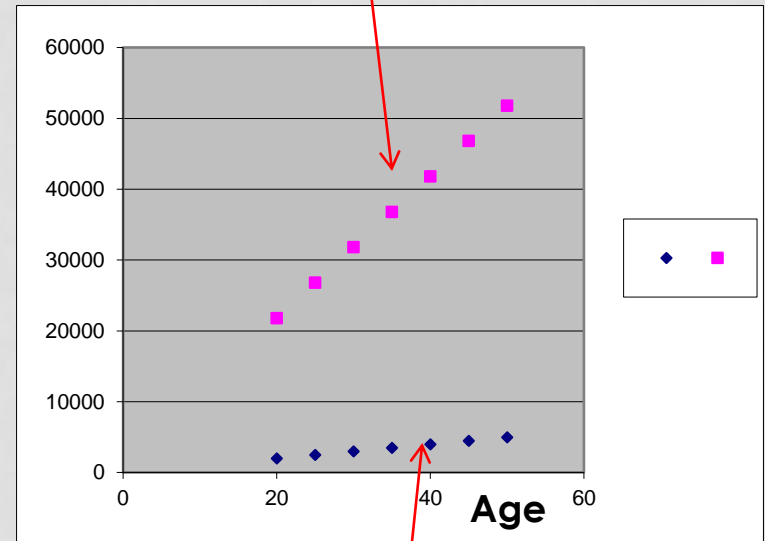
# EXAMPLE

Training data

Data Miner?	Age	Salary
Yes	20	2000
Yes	25	2500
Yes	30	3000
Yes	35	3500
Yes	40	4000
Yes	45	4500
Yes	50	5000
No	20	2000
No	25	2050
No	30	2100
No	35	2150
No	40	2200
No	45	2250
No	50	2300

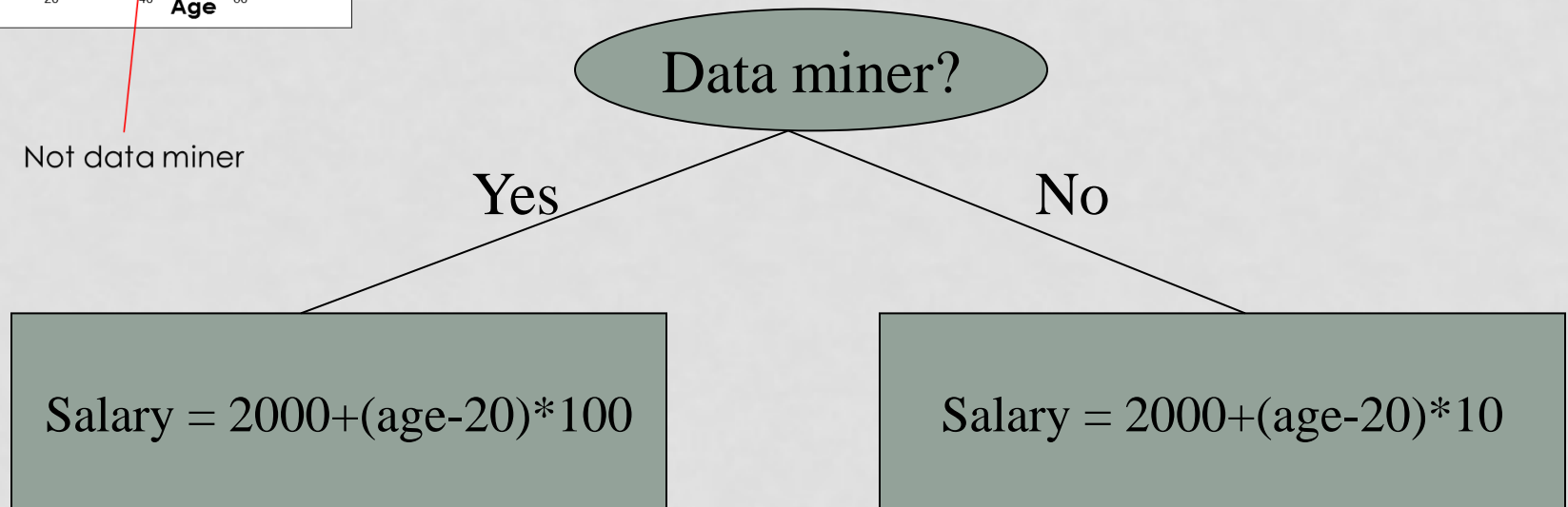
Data miner

Salary

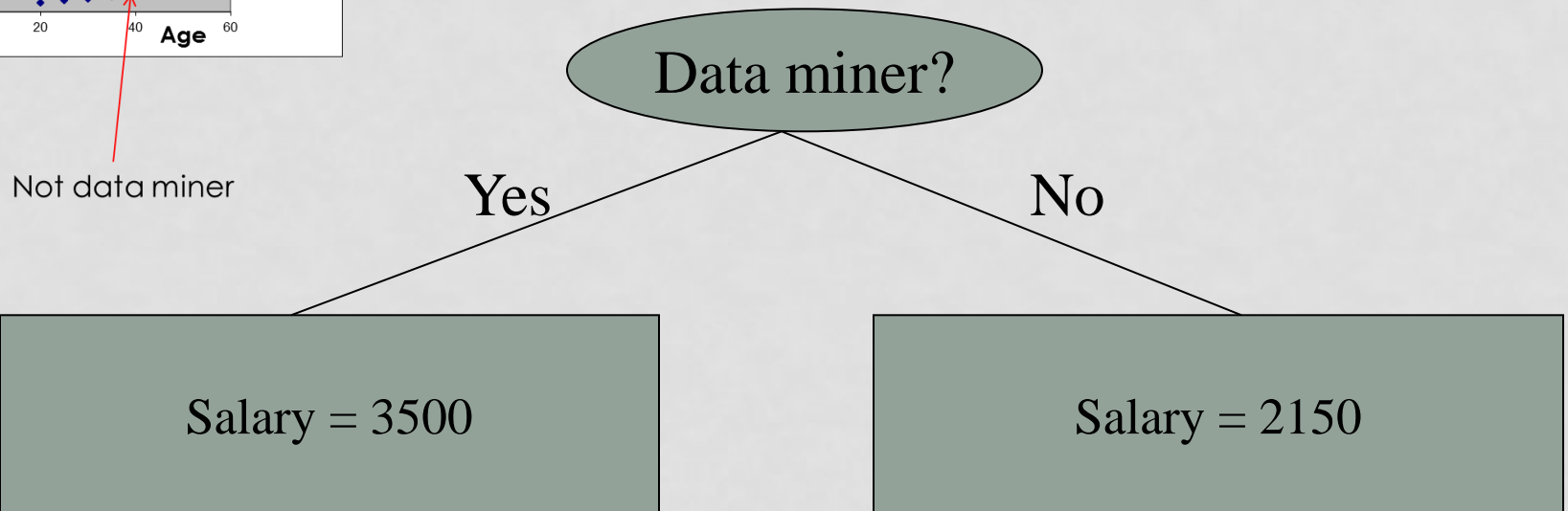


Not data miner

# MODEL TREES. EXAMPLE



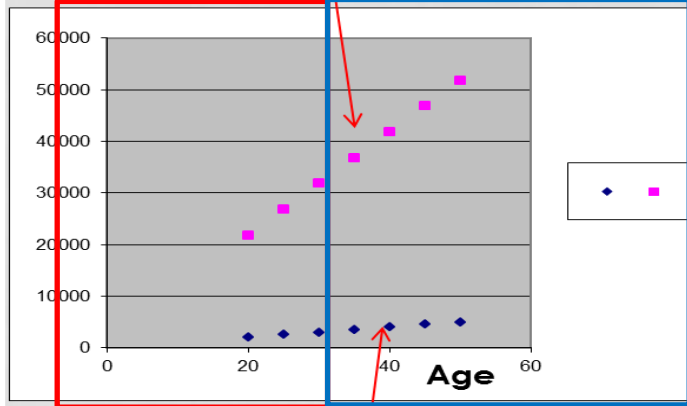
# REGRESSION TREES. EXAMPLE



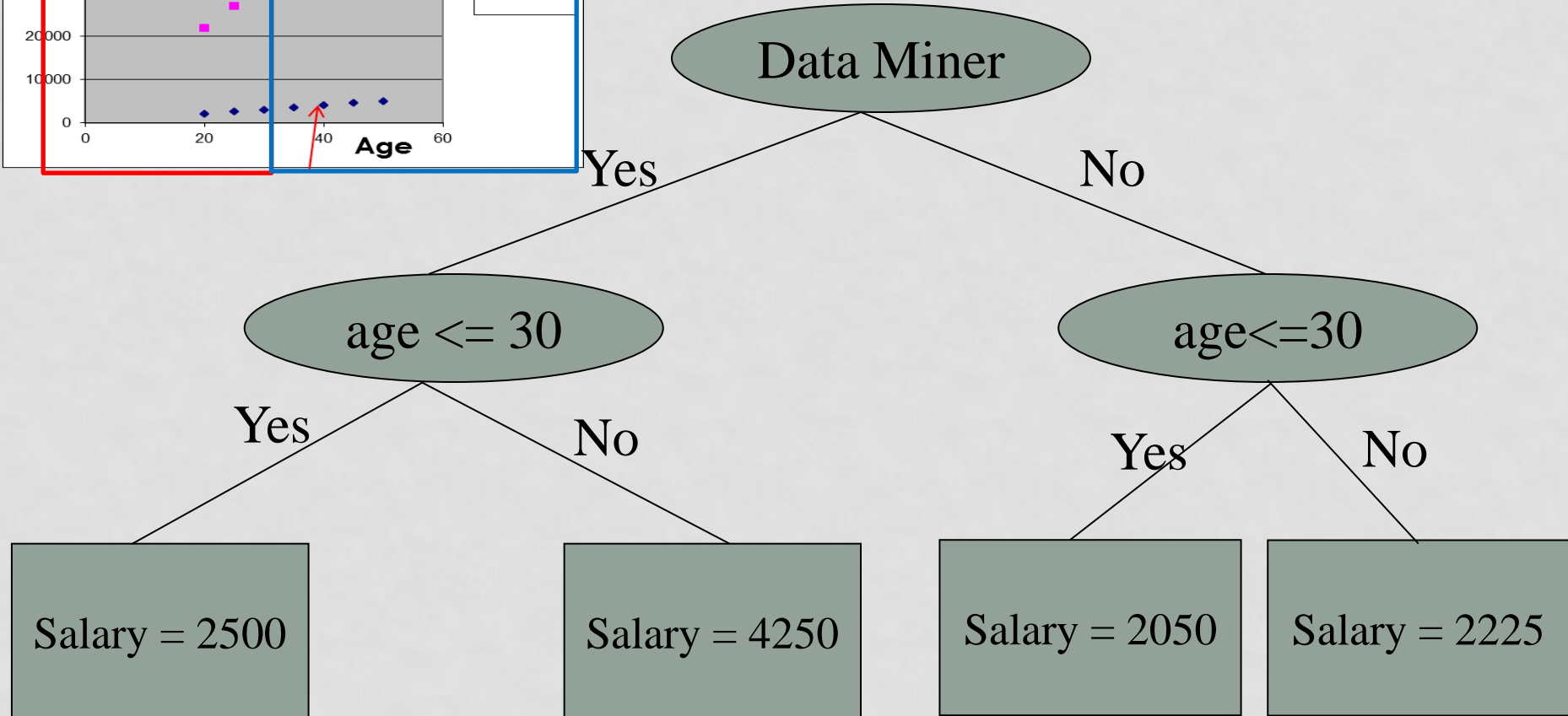
In the leaves, we can see the average salary for data miners (3500 euros) and the average salary for non-data miners (2150 euros)

# REGRESSION TREES. EXAMPLE

Salary



A larger depth and using age allows the regression tree to approximate the problema, piece-wise.



# TREES FOR REGRESSION

- Regression and model trees are built similarly, except that in the leaves:
  - For regression trees, the average output value is computed
  - For model trees, a linear model is constructed (M5 (Quinlan, 93))
- Trees for regression are built similarly than trees for classification (decision trees), except that **standard deviation / variance** is reduced (instead of entropy)
- The tree is built recursively until a stopping condition is reached (max\_depth, minsplit, ...)

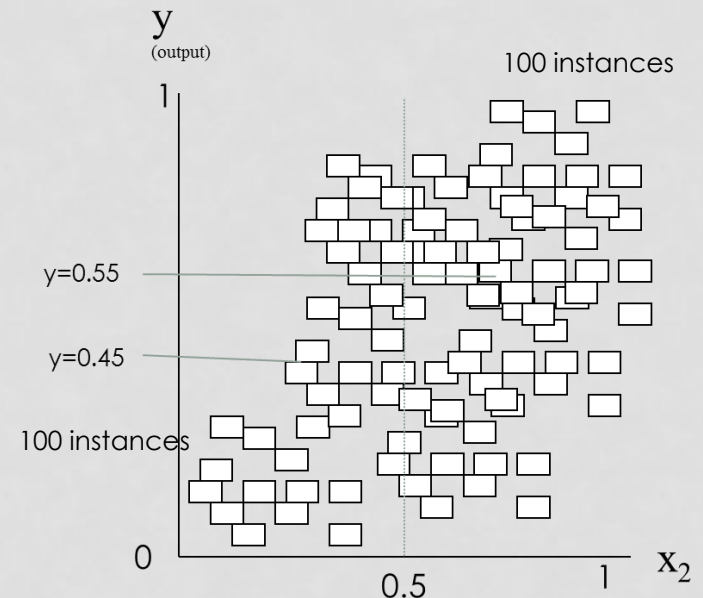
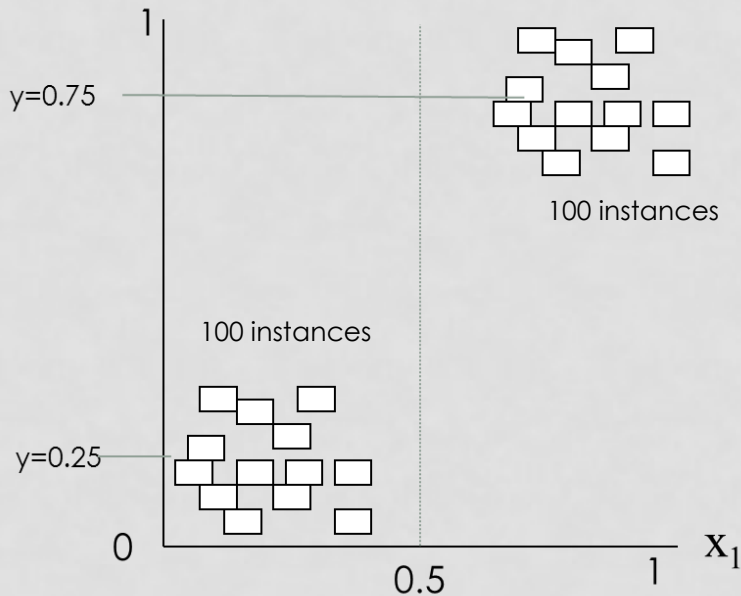
# WHAT IS THE BEST NODE TO PUT IN THE ROOT OF THE TREE?

Which attribute is better?

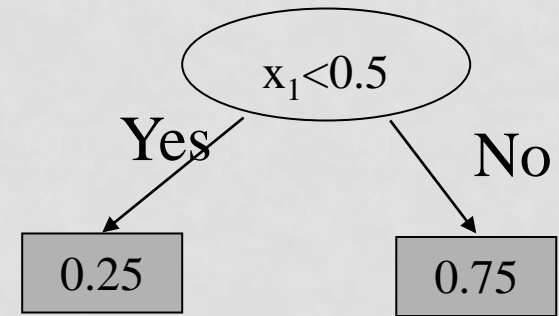
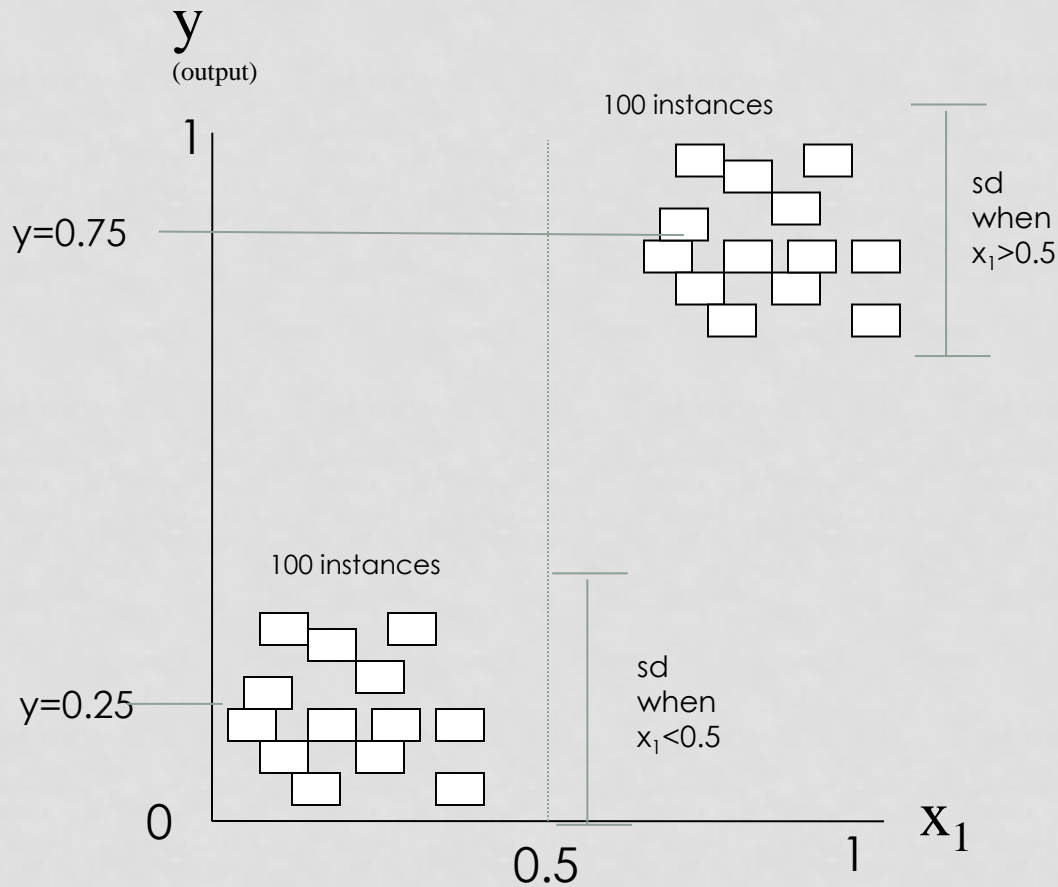
$x_1$  or  $x_2$ ?

We will choose the attribute for which the average standard deviation (sd) **after the partition** is small:

$$100/200 * \text{sd}(x_i < 0.5) + 100/200 * \text{sd}(x_i > 0.5)$$



# WHAT IS THE BEST NODE TO PUT IN THE ROOT OF THE TREE?

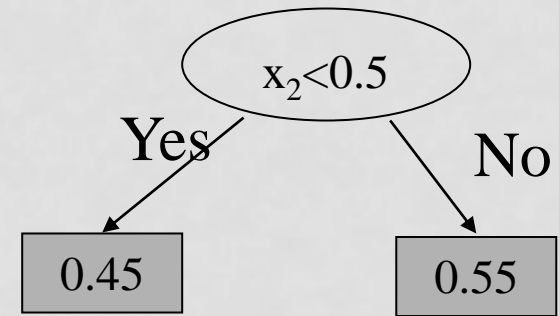
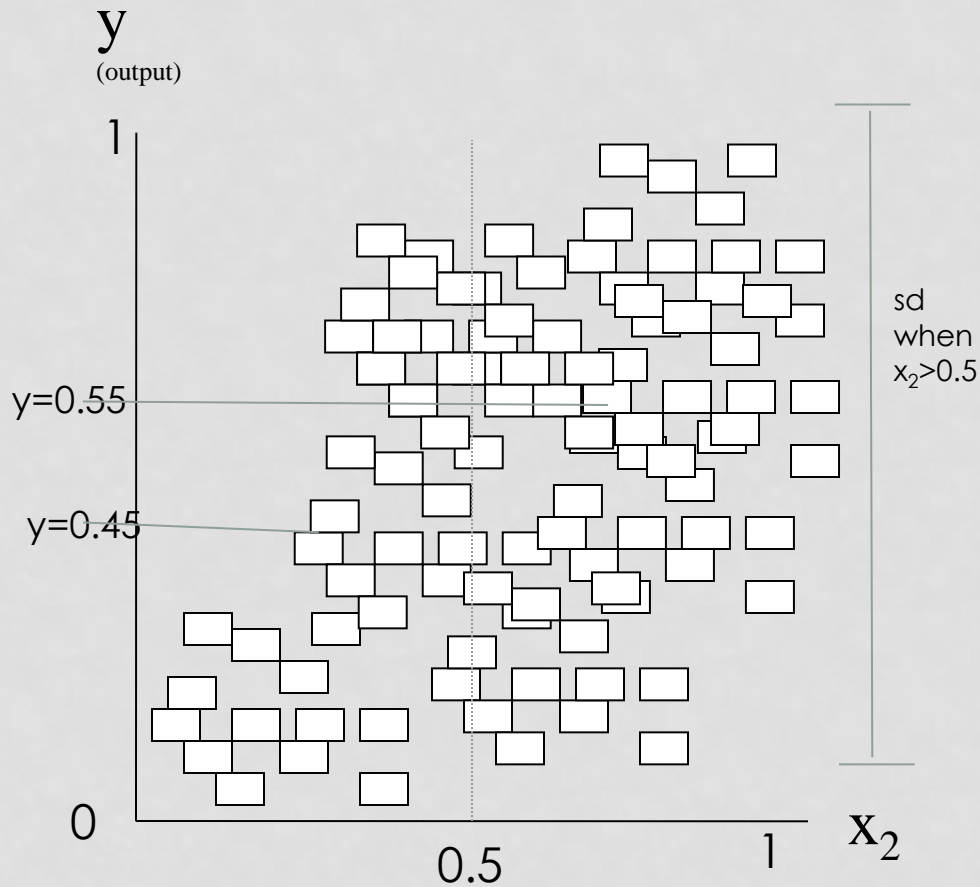


It can be noticed that  $x_1$  is quite predictive

Instances after doing the partitions are not spread

$\frac{1}{2} * sd(x_1 < 0.5) + \frac{1}{2} * sd(x_1 > 0.5)$  is small

# WHAT IS THE BEST NODE TO PUT IN THE ROOT OF THE TREE?



It can be noticed that  $x_2$  is not predictive

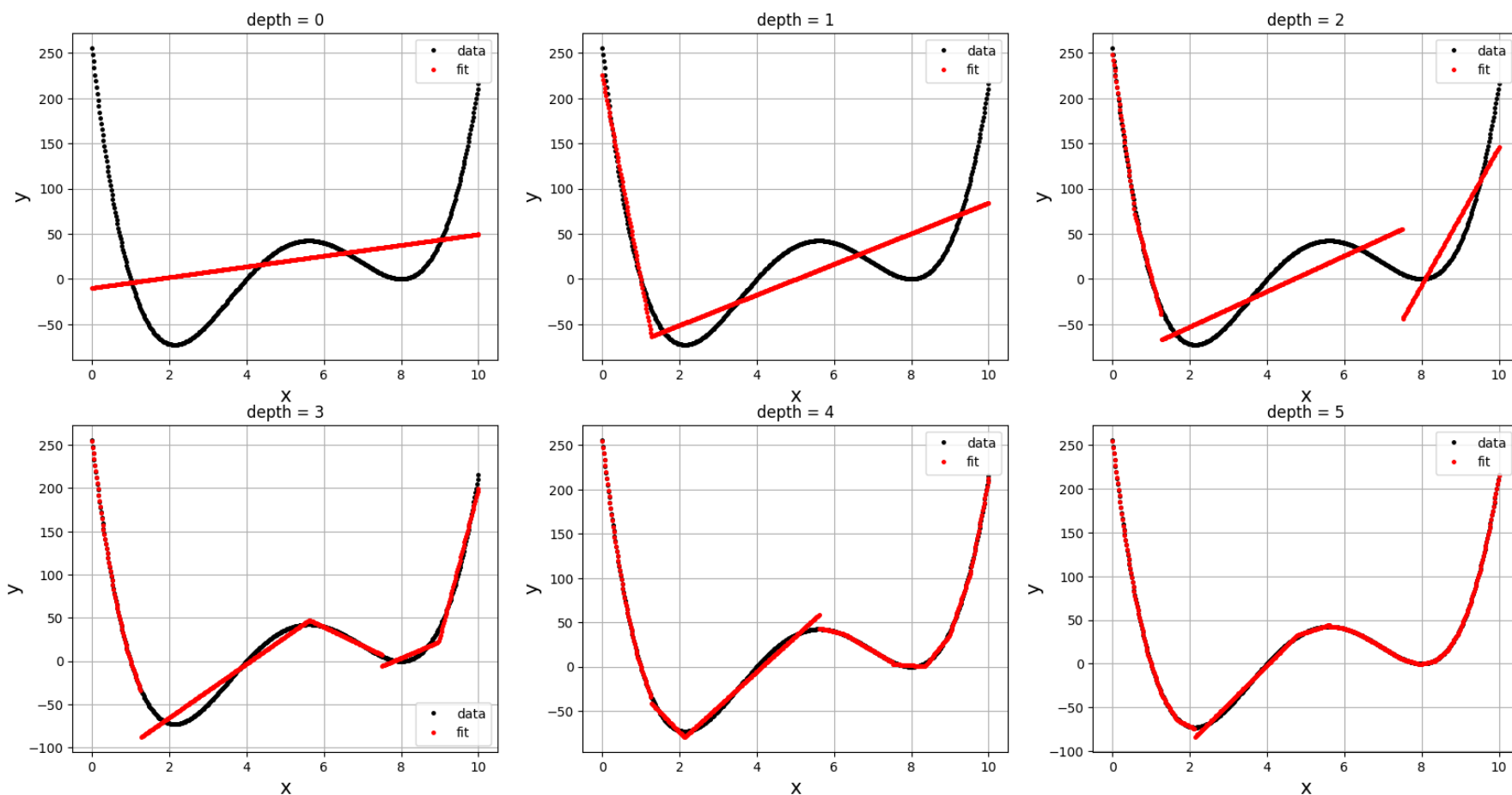
Instances after the partition are very spread

$\frac{1}{2} * \text{sd}(x_2 < 0.5) + \frac{1}{2} * \text{sd}(x_2 > 0.5)$  is very large



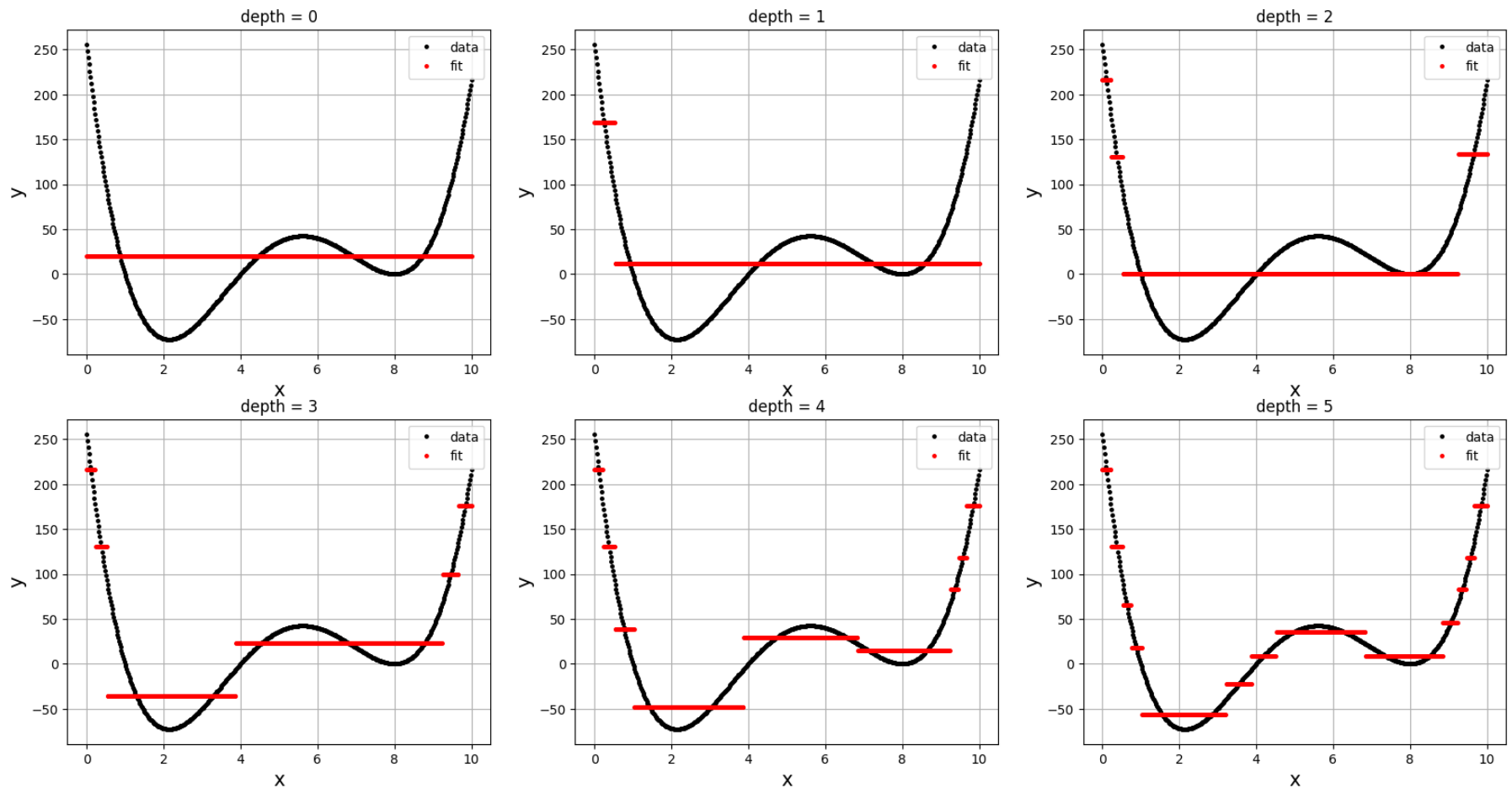
# MODEL TREE

Model tree (model = linear\_regr) fits for different depths



# REGRESSION TREE

Model tree (model = mean\_regr) fits for different depths

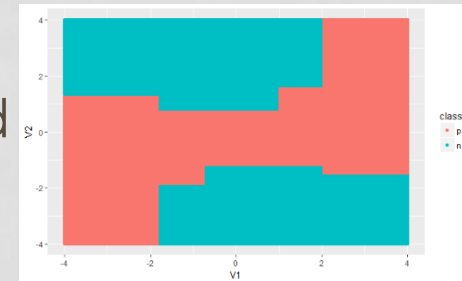


# ACTUAL METHODS

- Classification and regression trees: C4.5, C5.0, CART
- Model trees: M5P, Cubist

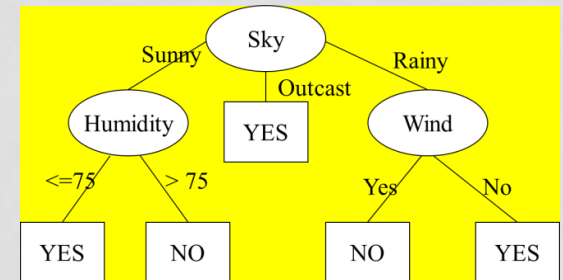
# DISADVANTAGES OF TREES

- Boundaries too simple for some problems (parallel to axes)
  - However, trees are the elements of advanced methods, such as Random Forests and Gradient Tree Boosting (ensembles).
- Hyper-parameters must be tuned carefully (maxdepth, ..), otherwise (deep) trees are prone to overfitting (bad generalization).

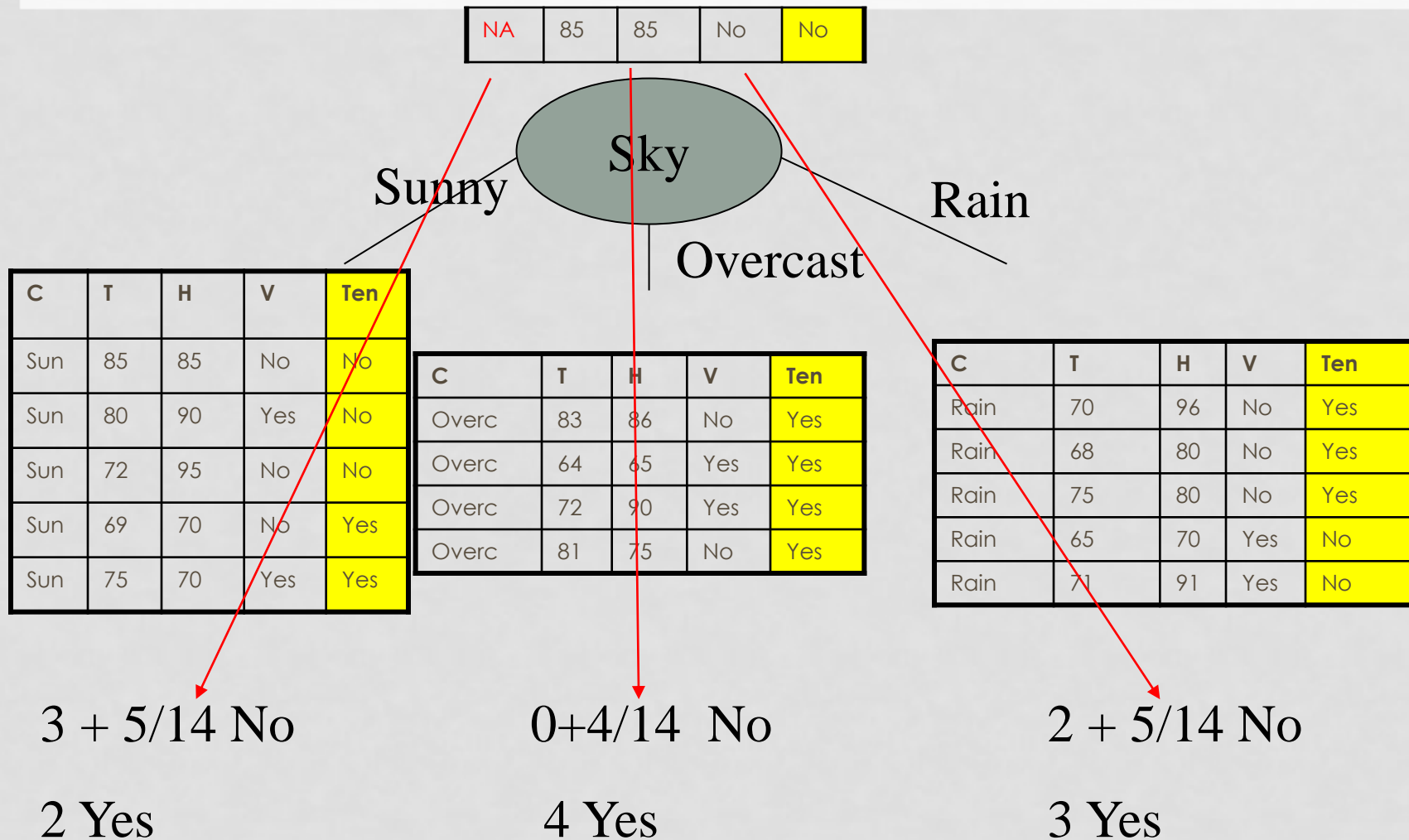


# ADVANTAGES OF TREES

- Interpretable.
  - Attributes closer to the root are the most relevant.
- They can handle naturally categorical attributes (no need for one-hot-encoding variables)
  - But not in scikit-learn 😞
- Training is fast (compared to other methods)
- They can handle multiple classes naturally.
- They can handle missing values



# HANDLING "MISSING VALUES"



# HANDLING "MISSING VALUES" IN SCIKIT-LEARN TREES

- Unfortunately, the previous approach is not available in scikit-learn.
- Before version 1.3, the way of dealing with missing values (np.nan) in scikit-learn was **imputation** : replacing the missing value by some other value (mean, median, mode). This can still be used, and belongs to "preprocessing".
- Since version 1.3, sklearn decision trees are able to deal with missing values in this manner:
  - For each potential threshold on the non-missing data, the splitter will evaluate the split with all the missing values going to the left node or the right node. That is, the algorithm determines what is the best branch to send instances with a missing value for that particular feature.
  - When making predictions for new instances, the tree will remember which way the missing values should go.
- Yet another way is **to consider the missing value just like another value**: if feature A has possible values 'red', 'green', 'blue', but some values are missing, then the actual set of values is: 'red', 'green', 'blue', NA