

# Lecture: Machine Learning for Data Science

Winter semester 2021/22

Lectures 3 & 4: Classification (Basics & Decision  
Trees & KNNs)

Prof. Dr. Eirini Ntoutsi

# Recap

- Data (instances and features)
- Different feature types
- Proximity

# Outline

- Supervised learning
  - Classification basics
  - Decision trees
  - Lazy vs Eager learners
  - KNNs
- Things you should know from this lecture & reading material

# Supervised learning vs unsupervised learning

## ■ Supervised learning tasks

- Supervision: The training data are accompanied by labels indicating the class of the observations (**labeled dataset**)
  - Labels as direct feedback by an expert
- Classification: The goal is to learn a model from the training data which can be used to classify new instances

ID	Age	Car type	Risk
1	23	Familie	high
2	17	Sport	high
3	43	Sport	high
4	68	Familie	low
5	32	LKW	low

*Class attribute: risk={high, low}*

## ■ Unsupervised learning tasks

- The class labels of training data is unknown (**unlabeled dataset**)
- Clustering: The goal is to group the data into groups of similar instances (the so-called clusters)

ID	Age	Car type	Risk
1	23	Familie	high
2	17	Sport	high
3	43	Sport	high
4	68	Familie	low
5	32	LKW	low

# Supervised learning tasks: Classification vs Regression

- Both supervised learning tasks
  - In **classification**, the class attribute is discrete.
  - In **regression**, the class attribute is continuous.

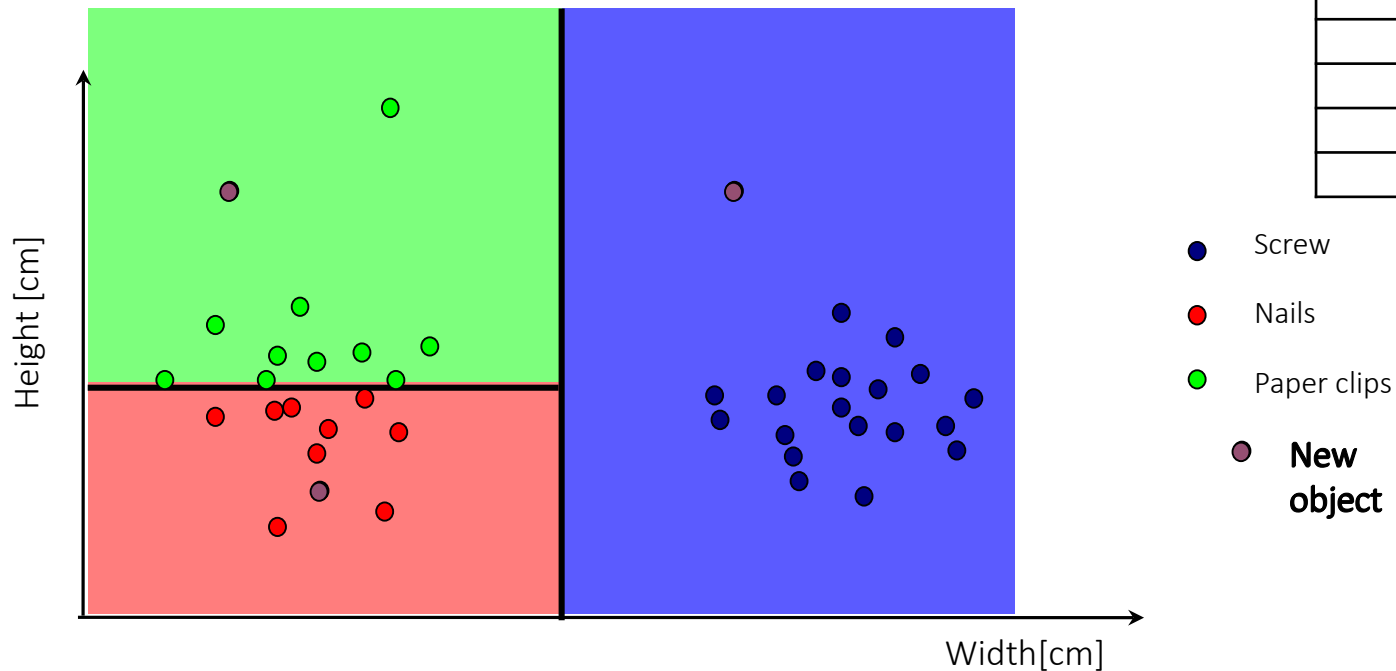
ID	Age	Car type	Risk
1	23	Familie	high
2	17	Sport	high
3	43	Sport	high
4	68	Familie	low
5	32	LKW	low

What is the predicted risk of a person with a certain Age and Car type?

House ID	Size (feet)	Old	Price
1	500	10	100K
2	1000	20	500K
3	2000	50	300K
4	300	15	200K

What is the predicted price for a house of a certain size and age?

# Classification: an example (from 1st lecture)



- We want to learn a model to classify new instances as screw, nails or paper clips
- The model provides a mapping/function from the “height, width space” to the (“nails, screw, paper clips”) space

# Applications of classification

- Churn prediction
  - Is the customer going to leave the company? e.g., in telecommunication companies
- Target marketing
  - Is the customer a potential buyer for a new computer?
- Medical diagnosis
  - Detecting lung cancer or strokes based on computerized tomography (CT) scans
- Character recognition
- Credit approval
  - Classify bank loan applications as e.g. safe or risky.
- Fraud detection
  - e.g., in credit cards: is a transaction fraud or not?
- ...

# Outline

- Supervised learning
- Classification basics
- Decision trees
- Lazy vs Eager learners
- KNNs
- Things you should know from this lecture & reading material



# Problem formulation

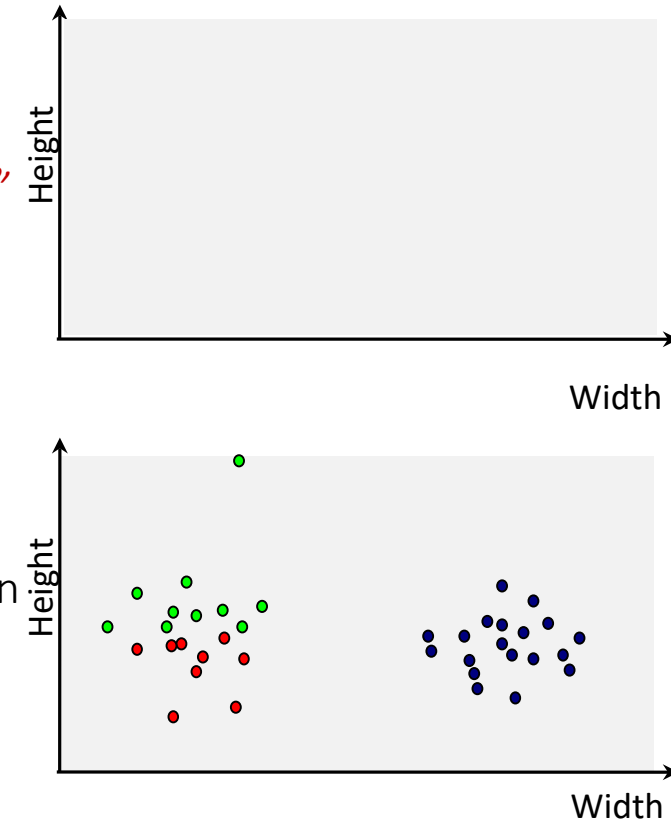
- “A computer program is said to learn from *experience*  $E$  w.r.t. some class of *tasks*  $T$  and *performance measure*  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .”

[Tom Mitchell](#), Machine Learning 1997.

- What is the task  $T$ ?
- What is the experience  $E$ ?
- What is the performance/evaluation measure  $P$ ?

# Problem formulation

- A classification problem is described through a vector **predictive attributes**  $X = (X_1, X_2, \dots, X_d)$  and the **class attribute**  $Y$ .
  - each predictive attribute  $X_i$  has a value domain,  $d(X_i)$
  - the domain of the class attribute is  $d(Y) = \{c_1, c_2, \dots, c_k\}$ ;  $k$  is the number of classes.
- The **attribute space**  $S_X = d(X_1) \times d(X_2) \dots \times d(X_d)$
- The **attribute-class space**  $S_{(X,Y)} = S_X \times Y$
- The goal of a classifier is to learn a **function/ mapping/hypothesis**  $h: X \rightarrow Y$  which given a new instance  $x$  can predict its class label  $h(x)$ .
- To this end, **training data**  $D$ ,  $|D|=n$  are provided to the classifier
  - Training examples are drawn from the joint distribution  $P(X,Y)$  of the predictive attributes and the class attribute (i.e., from the  $S_{(X,Y)}$  space).
  - Training examples have the form  $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$
- Please note that
  - we don't know the **underlying data distribution**  $P(X)$  that generates the data,
  - neither the **true labeling function**  $f(): X \rightarrow Y$  that produces the labels  $y_i = f(x_i)$  for all  $i$ .
  - The only information provided about the problem provided to the learner is  $D$



Task  $T$

Experience  $E$

# Problem formulation

Performance measure  $P$

- How to evaluate the performance of  $h()$ ?
- By measuring the errors of  $h()$  on instances drawn from the distribution of the instances  $P(X)$ 
  - What is the probability of randomly choosing an example  $x$  for which the prediction  $h(x)$  is different from the true label  $f(x)$

$$L_{P(X),f}(h) = P_{x \sim P(X)}[h(x) \neq f(x)]$$

- Referred to as the **loss of the learner**.
  - This is **true error** of  $h()$
- What is the problem with this definition?
  - We don't have access to the probability distribution  $P(X)$  neither to the true labeling function  $f()$  → so we cannot compute the true error of  $h$
  - The only thing that we have is the training set  $D = \{(\vec{x}, y)\}$
- We can compute the **training error** (also, called **empirical error** or **empirical loss** of  $h$ )

$$L_D(h) = \frac{|\{xi \in D: h(xi) \neq yi\}|}{|D|}$$

- since  $D$  is a sample of our population, we can look for a solution  $h()$  that minimizes  $L_D(h)$  ← **Empirical Risk Minimization (ERM)**

# Problem formulation

- Why finding  $h$  that minimizes the training error is not a good idea?
- A simple example of a classifier  $h()$  that minimizes the training error is one that **memorizes** all training instances:

$$h_D(x_i) = \{y_i, \text{if } x_i \in D; \text{otherwise } 0\}$$

- In this case, the error on the sample is 0:  $L_D(h)=0$
- However, is this a good predictor w.r.t. instances drawn from  $P(X)$ ?
  - No!
- So, a classifier that is excellent in the training data, might have poor performance in the “real world”
  - This is called **overfitting**
- Empirical risk minimization might lead to overfitting
- The goal is to learn a classifier that performs well on the training data but also it is highly likely to perform well over the underlying data distribution (so, avoid overfitting)

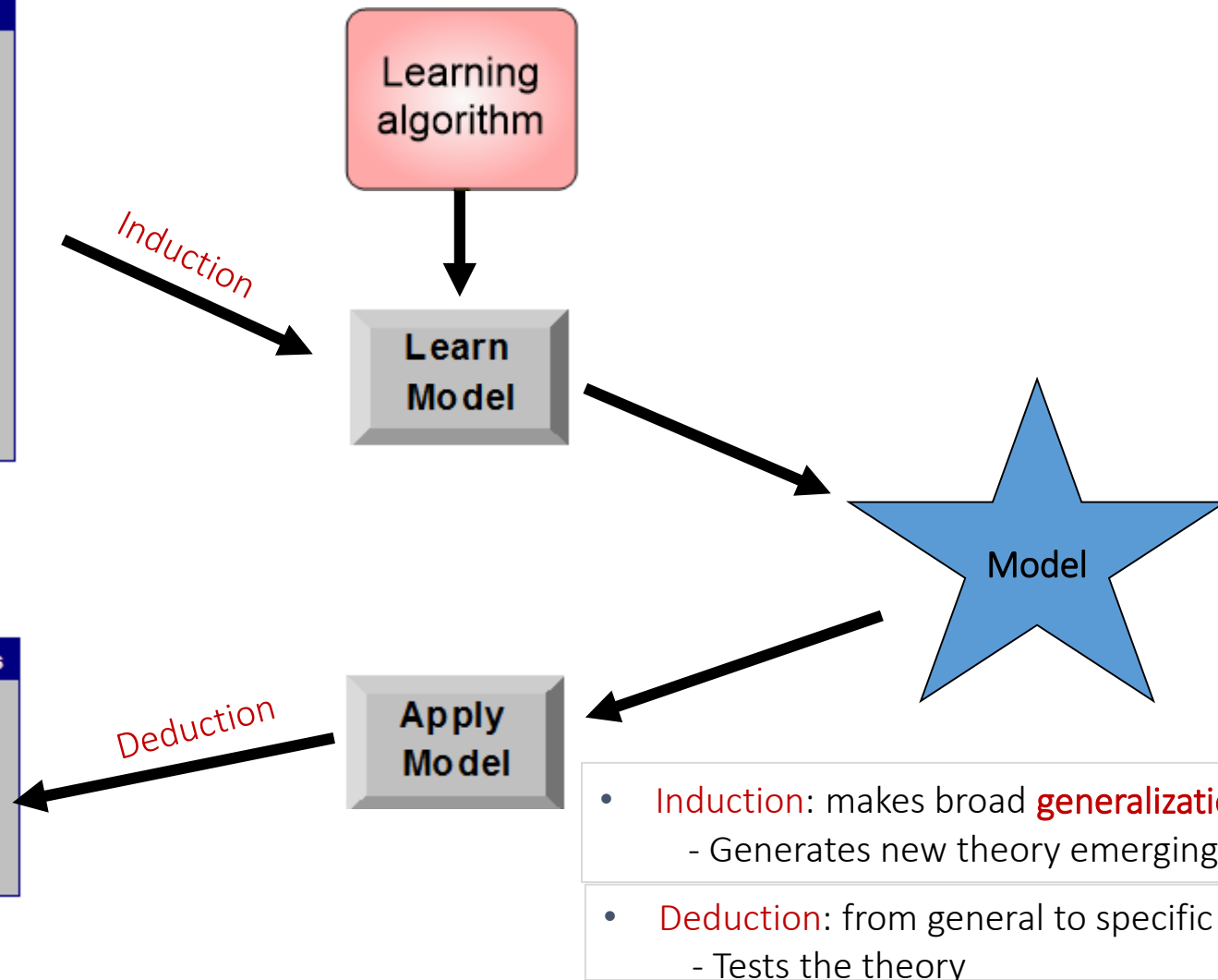
# General approach for building a classifier

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training data

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Future unseen instances



# General approach for building a classifier

- How to avoid overfitting and enable generalization?
  - Restrict/ Make assumptions about the learner  $h()$  → **inductive bias**
  - **Inductive bias** "refers to a set of (explicit or implicit) assumptions made by a learning algorithm in order to perform induction, that is, to generalize a finite set of observation (training data) into a general model of the domain. Without a bias of that kind, induction would not be possible, since the observations can normally be generalized in many ways." (Hüllermeier, Fober & Mernberger, 2013)
  - **Bias-free learning is futile**: A learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.

# Empirical risk minimization (ERM) with inductive bias

- A popular type of inductive bias: Restrict the hypothesis  $h(): X \rightarrow Y$  to a hypothesis class  $H$ 
  - e.g., assume  $h()$  is a decision tree
  - e.g., assume  $h()$  is a linear function
  - ...
- Choose the hypothesis  $h \in H$  with the smallest possible error over the training data  $D$

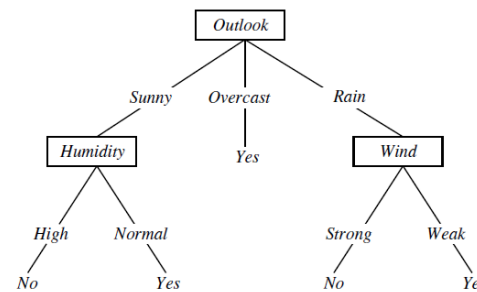
$$ERM_H(D) \in \operatorname{argmin}_{h \in H} L_D(h)$$

- This already helps (under conditions, proof not covered)
- Many more techniques to mitigate overfitting:
  - Regularization
  - Pruning
  - Early stopping
  - ...

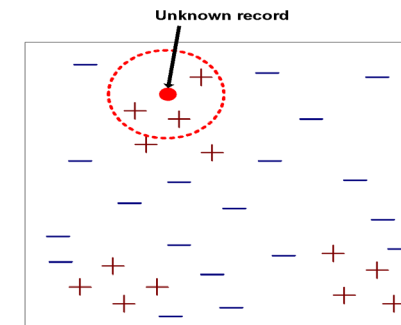
# A large variety of classifiers (in red the ones we will cover)

- There are different types of mappings/functions/hypotheses  $h(): X \rightarrow Y$ .
- We refer to them mainly as **classification models** or **classifiers**, hereafter.

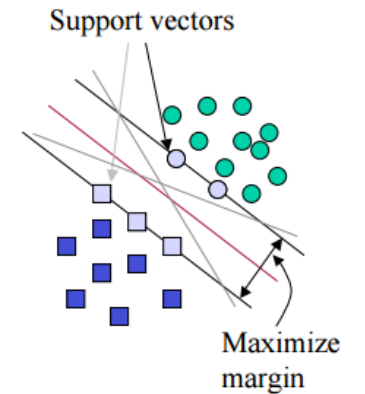
- Decision trees
- Nearest neighbors
- Bayesian classifiers
- Support vector machines
- Neural networks
- Boosting
- Bagging
- Random forests
- ....



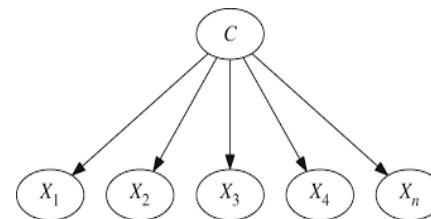
Decision trees



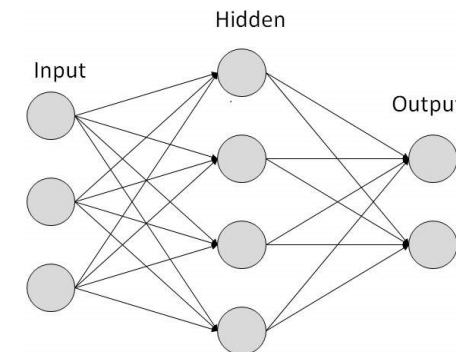
k nearest neighbours



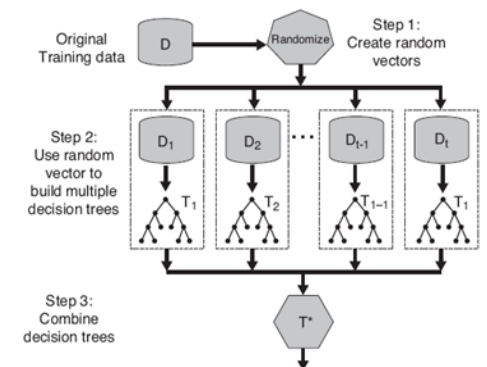
Support vector machines



Bayesian classifiers



Neural networks

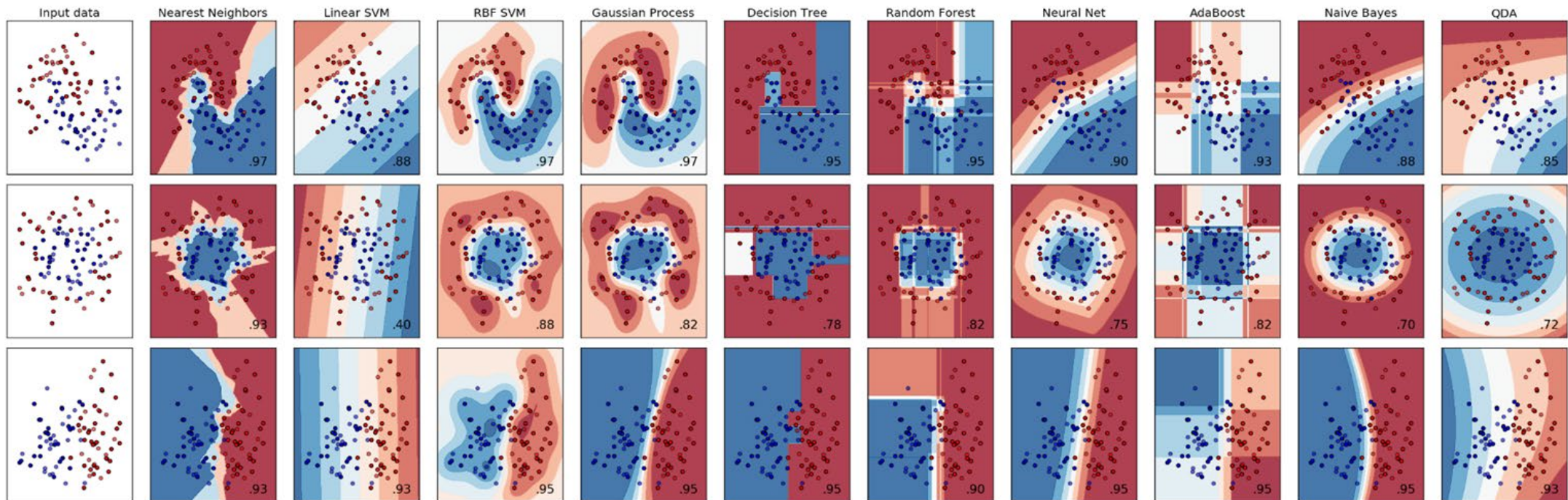


Ensembles



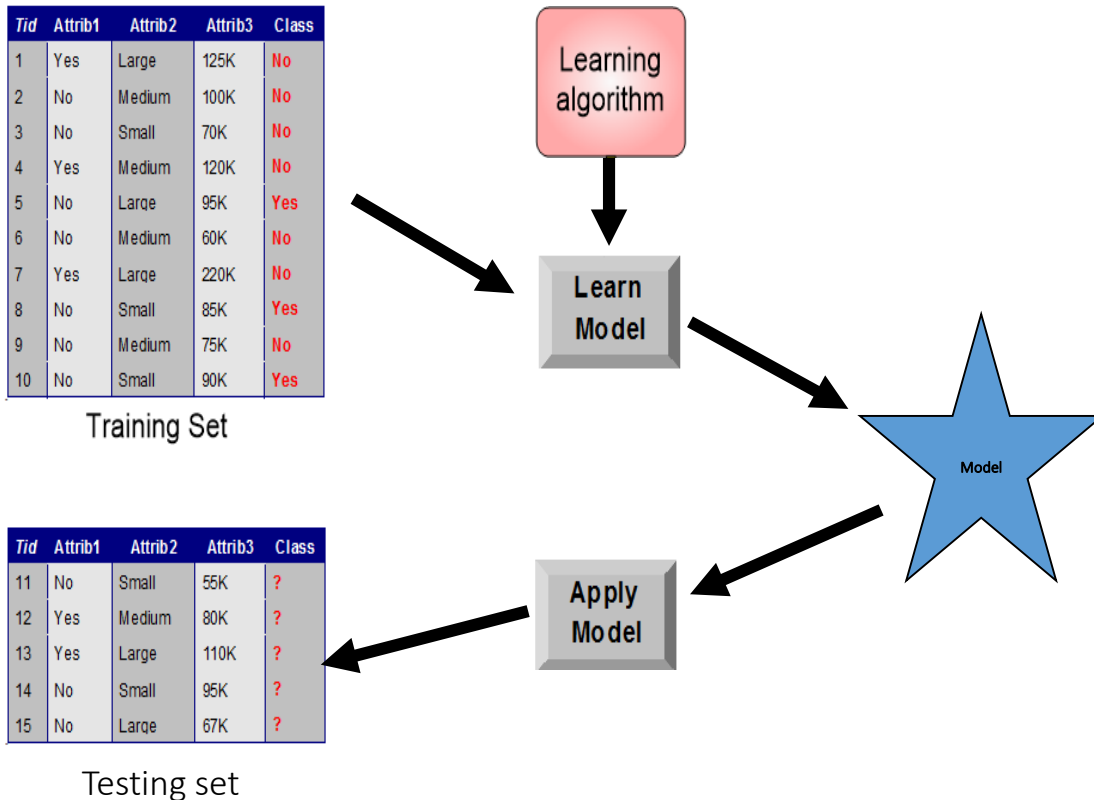
# Classification models

- Each **classifier** comes with its own assumptions (**inductive bias**) and parameters
  - and there are >1 learning algorithms for each classifier



Source: [http://scikit-learn.org/stable/auto\\_examples/classification/plot\\_classifier\\_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py](http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py)

# The role of training/testing sets



- **Training set**: used to learn a model of the population/phenomenon we are studying
    - Requirement: It should be **representative** of the population
    - Samples follow the **i.i.d. assumption** (independent and identically distributed)
  - **Testing set**: “Simulates” future unseen instances of the population
    - It is used for model evaluation only
  - Training and testing sets are assumed to come from the same distribution (**non-stationarity assumption**)
  - Training and testing sets should be disjoint
- Why?
- Why?

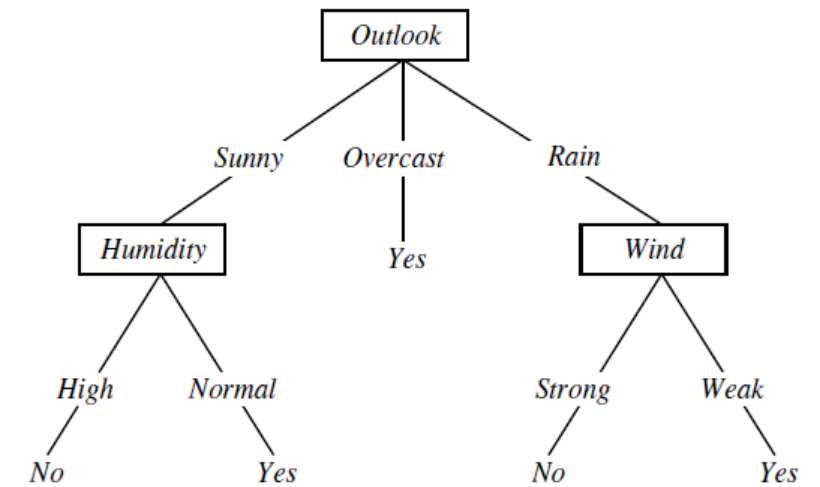
*We will cover classifier evaluation in a next lecture!*

# Outline

- Supervised learning
- Classification basics
- Decision trees
- Lazy vs Eager learners
- KNNs
- Things you should know from this lecture & reading material

# Decision tree (DTs) classifiers

- One of the most popular classification methods
- DTs are included in many commercial systems nowadays
- Easy to interpret, human readable, intuitive
- Simple and fast methods
- Many DT induction algorithms have been proposed
  - ID3 (Quinlan 1986)
  - C4.5 (Quinlan 1993)
  - CART (Breiman et al 1984)
  - ...



# Decision tree classifiers outline

- In this lecture, we will cover
  - Representation
  - How to build a tree/ Splitting attributes
  - Hypothesis space and inductive bias
  - Decision boundary
  - When to consider decision trees
  - Overfitting

# Decision tree classifiers outline

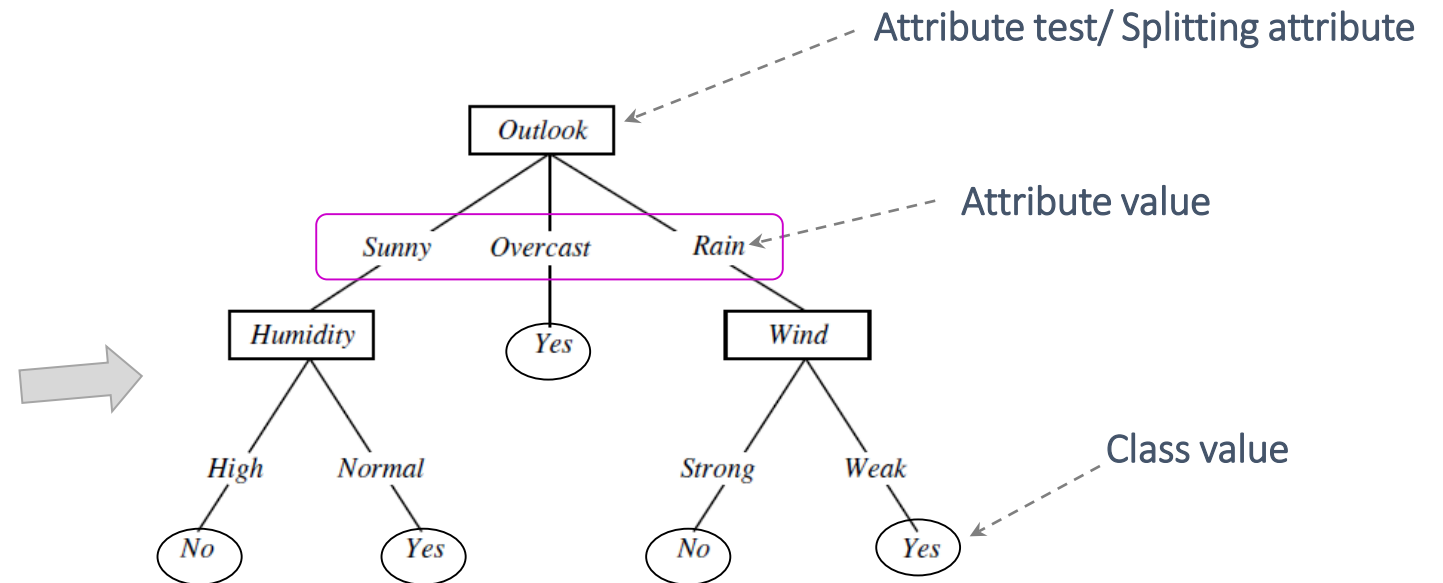
- In this lecture, we will cover
  - Representation
  - How to build a tree/ Splitting attributes
  - Hypothesis space and inductive bias
  - Decision boundary
  - When to consider decision trees
  - Overfitting

# Representation

- The learned function  $f()$  is represented by a decision tree!
- Representation
  - Each **internal node** specifies a test of some predictive attribute
  - Each **branch** descending from a node corresponds to one of the possible values for this attribute
  - Each **leaf node** assigns a class label

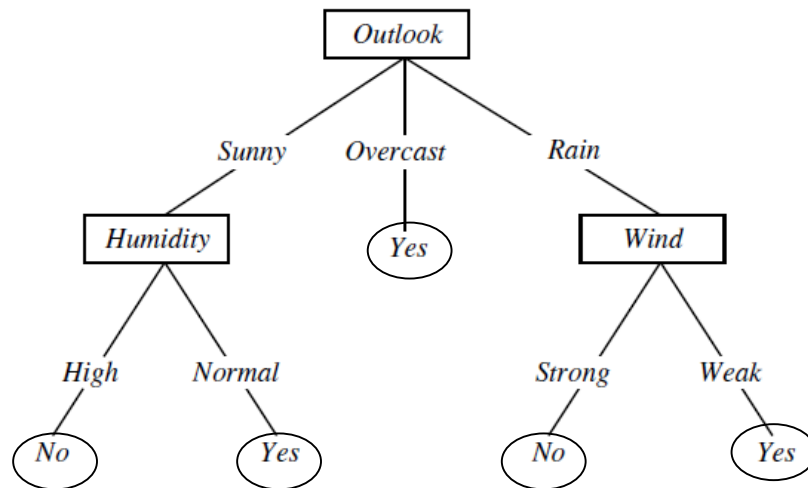
Training set

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
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



# Representation

- Decision trees represent a disjunction of conjunctions of constraints on the attribute values of the instances
  - Each path from the root to a leaf node, corresponds to a conjunction of attribute tests
  - The tree corresponds to a disjunction of these conjunctions, i.e.,  $(... \wedge ... \wedge ...) \vee (... \wedge ... \wedge ...) \vee ...$
- We can “translate” each path into IF-THEN rules (human readable)



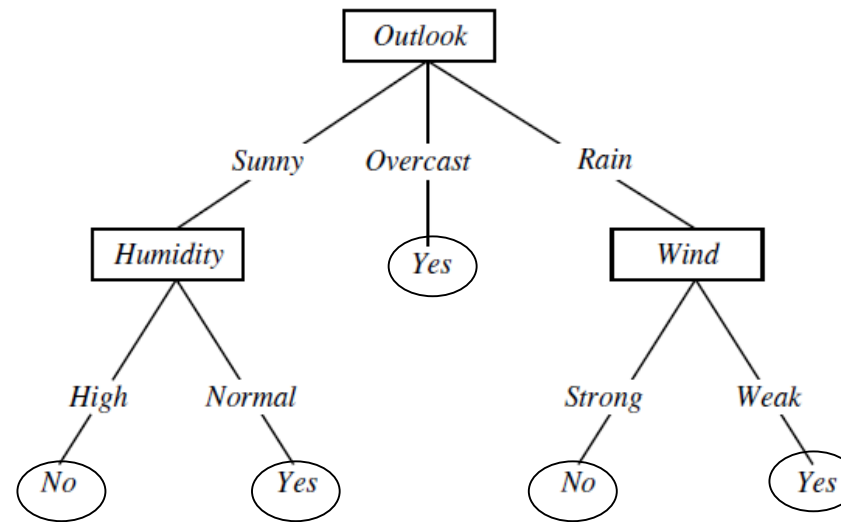
*IF ((Outlook = Sunny) ^ (Humidity = Normal)),  
THEN (Play tennis=Yes)*

*IF ((Outlook = Rain) ^ (Wind = Strong)),  
THEN (Play tennis=No)*



# Representation

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.



Should we play tennis?

$X_1 = ((\text{Outlook}=\text{sunny}) (\text{Temperature}=\text{hot})(\text{Humidity}=\text{high})(\text{Wind}=\text{Weak}))$



$X_2 = ((\text{Outlook}=\text{overcast}) (\text{Temperature}=\text{hot})(\text{Humidity}=\text{high})(\text{Wind}=\text{Weak}))$

# Short break (5')

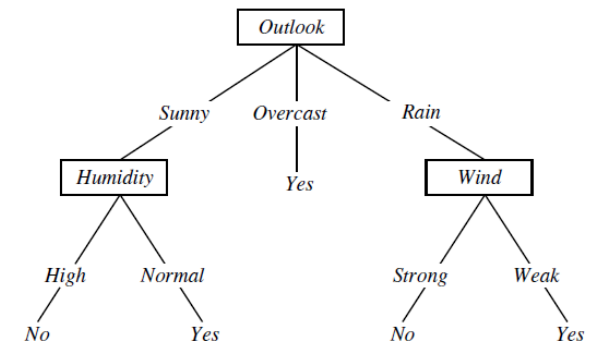
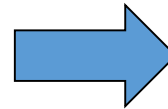


## How to build a decision tree?

- ❑ Think for 1'
- ❑ Discuss with your neighbours
- ❑ Discuss in the class

*Training set*

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
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



# Decision tree classifiers outline

- In this lecture, we will cover
  - Representation
  - How to build a tree/ Splitting attributes
  - Hypothesis space and inductive bias
  - Decision boundary
  - When to consider decision trees
  - Overfitting

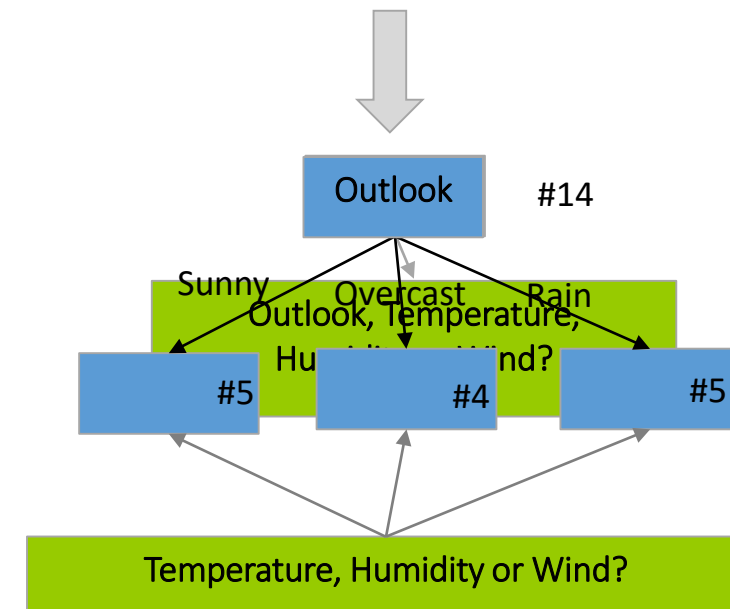
# The basic decision tree learning algorithm

## Basic algorithm (ID3, Quinlan 1986)

- The tree is constructed in a top-down recursive divide-and-conquer manner
- At start, all the training examples are at the **root node**
- The question is “*Which attribute should be tested/ selected for split?*”
  - Attributes are evaluated using some statistical measure, which determines how well each attribute alone classifies the training examples.
  - The **best attribute** is selected and used as the **splitting attribute** at the root.
- For each possible value of the splitting attribute, a descendant of the root node is created and the instances are mapped to the appropriate descendant node.
- The procedure is repeated for each descendant node, so instances are partitioned **recursively**.

Training set

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
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



# The basic decision tree learning algorithm

## ■ Pseudocode

Main loop:

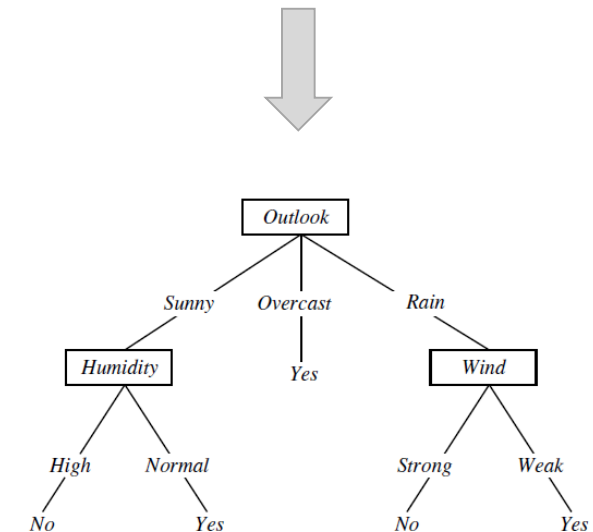
1.  $A \leftarrow$  the “best” decision attribute for next *node*
2. Assign  $A$  as decision attribute for *node*
3. For each value of  $A$ , create new descendant of *node*
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then **STOP**, Else iterate over new leaf nodes

## ■ “When do we stop partitioning?”

- All samples for a given node belong to the same class
  - This is the class of the leaf
- There are no remaining attributes for further partitioning
  - The majority class is the class of the leaf (*majority voting*)

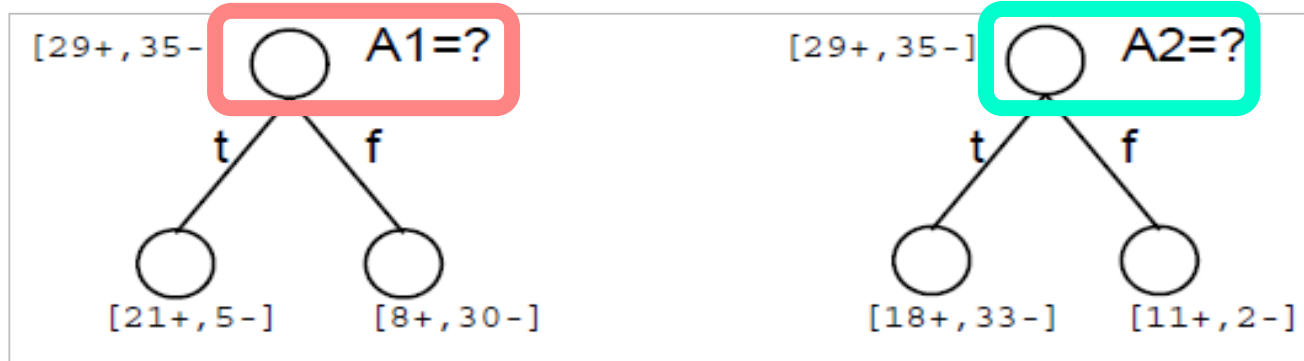
Training set

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
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

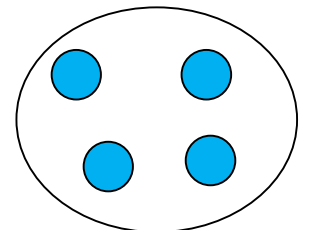
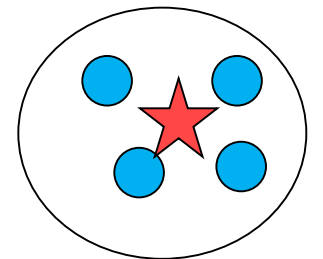
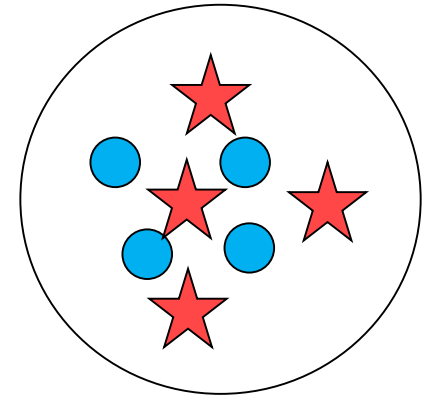


# Splitting attributes: Which attribute is the best?

- Which attribute to choose for splitting:  $A_1$  or  $A_2$ ?

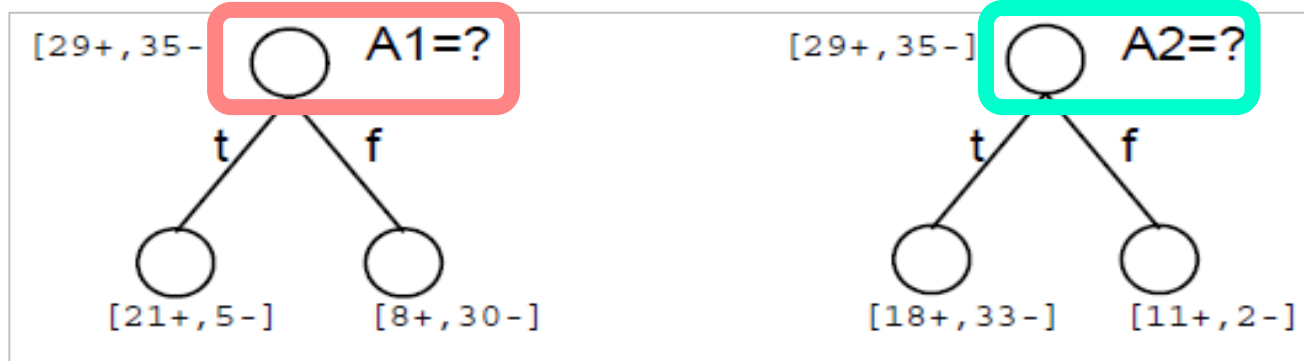


- The goal is to select the attribute that is most useful for classifying examples.
  - helps us be more certain about the class after the split
- We would like the resulting partitionings to be as pure as possible
  - A partition is **pure** if all its instances belong to the same class.



# Splitting attributes: Which attribute is the best?

- Which attribute to choose for splitting:  $A_1$  or  $A_2$ ?



- Different **split attribute selection measures**
  - ❑ Information gain
  - ❑ Gain ratio
  - ❑ Gini index
  - ❑ ...
  - ❑ all based on the degree of impurity of the parent node (before splitting) vs the children nodes (after splitting)

# Entropy for measuring impurity of a set of instances

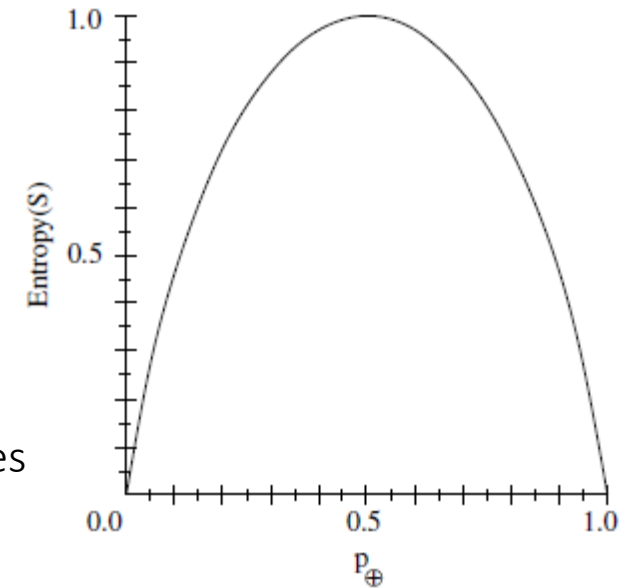
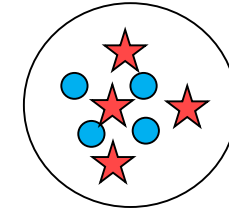
- Let  $S$  be a collection of positive and negative examples for a binary classification problem,  $C=\{+, -\}$ .

- $p_+$ : the percentage of positive examples in  $S$
- $p_-$ : the percentage of negative examples in  $S$

- Entropy measures the impurity of  $S$ :

$$Entropy(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$$

- $Entropy = 0$ , when all members belong to the same class
- $Entropy = 1$ , when there is an equal number of positive and negative examples



- Entropy comes from information theory.
  - It represents the average amount of information needed to identify the class label of an instance in  $S$
  - The higher the entropy the more the information content

in the general case  
( $k$ -classification problem)  
$$Entropy(S) = \sum_{i=1}^k -p_i \log_2(p_i)$$



# Entropy example

② What is the entropy in the following cases?

□ S: [9+,5-]

□ S: [7+,7-]

□ S: [14+,0-]

$$Entropy(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$$

$$Entropy(S) = -\frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) = 0.940$$

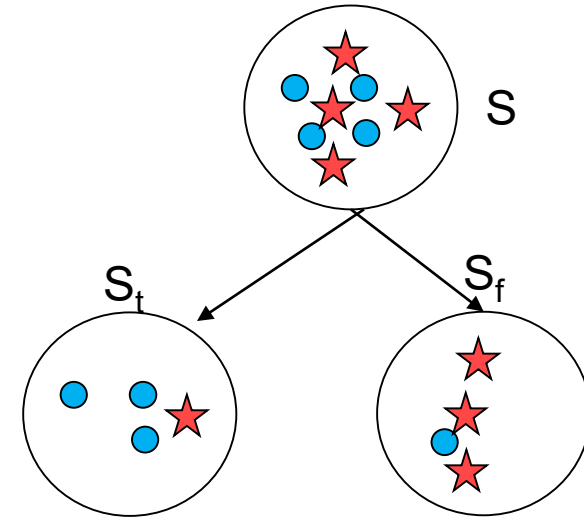
$$Entropy(S) = -\frac{7}{14} \log_2\left(\frac{7}{14}\right) - \frac{7}{14} \log_2\left(\frac{7}{14}\right) = 1$$

$$Entropy(S) = -\frac{14}{14} \log_2\left(\frac{14}{14}\right) - \frac{0}{14} \log_2\left(\frac{0}{14}\right) = 0$$

# Attribute selection measure: Information gain

- Used in ID3 (Quinlan, 1986)
- It uses entropy, a measure of pureness of the data
- The **Information Gain**  $Gain(S, A)$  of an attribute  $A$  relative to a collection of examples  $S$  measures the *entropy reduction* in  $S$  due to splitting on  $A$ :

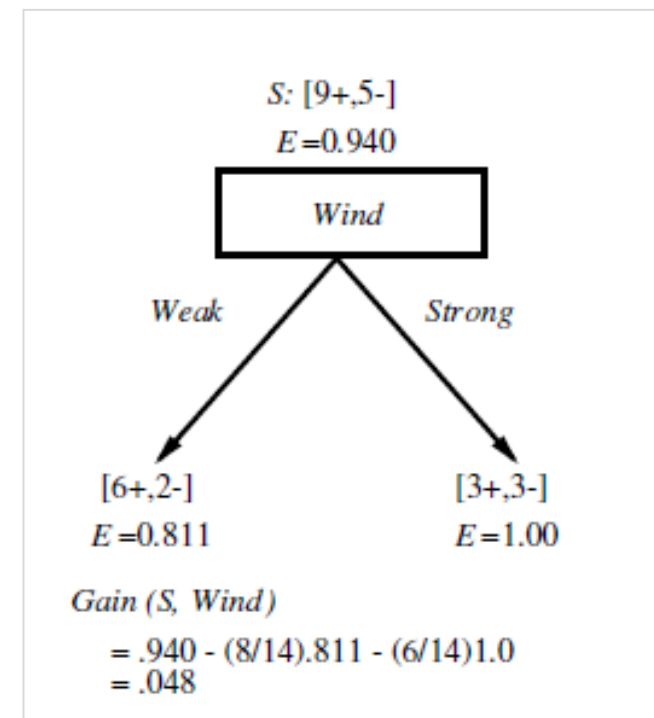
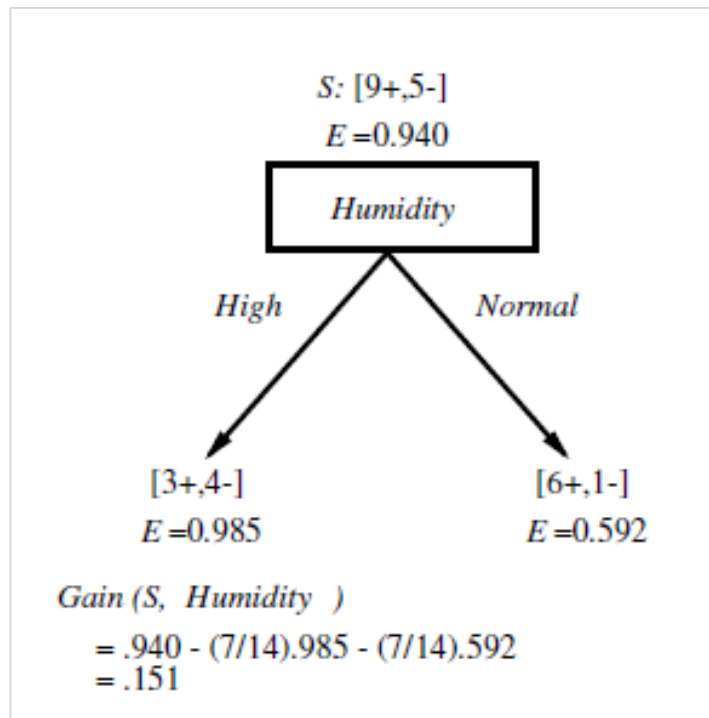
$$Gain(S, A) = \underbrace{Entropy(S)}_{\text{Before splitting}} - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \underbrace{Entropy(S_v)}_{\text{After splitting on A}}$$



- Information Gain measures the expected reduction in entropy due to splitting on  $A$
- The attribute with the higher entropy reduction is chosen for splitting

# Information Gain example 1

- Two options for splitting: “Humidity” and “Wind”?



Which attribute to choose for splitting?

# Information Gain example 2

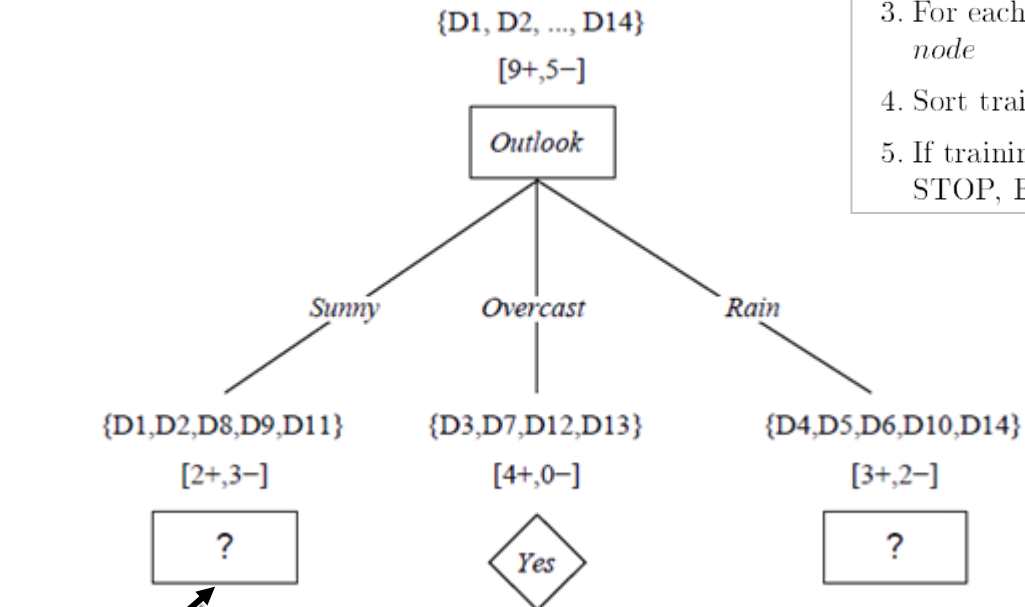
- Repeat recursively

Training set

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
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



Which attribute is chosen?



Which attribute should we choose for splitting here?

$$S_{\text{sunny}} = \{D1, D2, D8, D9, D11\}$$

$$\text{Gain}(S_{\text{sunny}}, \text{Humidity}) = .970 - (3/5) 0.0 - (2/5) 0.0 = .970$$

$$\text{Gain}(S_{\text{sunny}}, \text{Temperature}) = .970 - (2/5) 0.0 - (2/5) 1.0 - (1/5) 0.0 = .570$$

$$\text{Gain}(S_{\text{sunny}}, \text{Wind}) = .970 - (2/5) 1.0 - (3/5) .918 = .019$$

Main loop:

1.  $A \leftarrow$  the “best” decision attribute for next *node*
2. Assign  $A$  as decision attribute for *node*
3. For each value of  $A$ , create new descendant of *node*
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

# Attribute selection measure: Information Gain

- Information gain is biased towards attributes with a large number of distinct values.

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- Consider unique identifiers like ID or credit card
- Such attributes have a high information gain, because they uniquely identify each instance, but we do not want to include them in the decision tree
  - E.g., deciding how to treat a customer based on their credit card number is unlikely to generalize to customers we haven't seen before.
- Measures have been proposed that “correct” this issue
  - Quinlan suggested information gain in his ID3 system and later the gain ratio, both based on entropy.
  - Gini index

# Attribute selection measure: Gain ratio

- C4.5 (a successor of ID3) uses gain ratio which overcomes the problem by normalizing the gain of splitting on attribute  $A$  by the split information of  $A$ :

$$\text{GainRatio}(S, A) = \frac{\text{Gain}(S, A)}{\text{SplitInfo}(S, A)}$$

Measures the information  
w.r.t. classification

Measures the information  
generated by splitting  $S$  into  
 $|Values(A)|$  partitions

$$\text{SplitInfo}(S, A) = - \sum_{v \in Values(A)} P_v \cdot \log_2(P_v) = - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot \log_2\left(\frac{|S_v|}{|S|}\right)$$

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

$$\text{Entropy}(S) = \sum_{i=1}^k -p_i \log_2(p_i)$$

# Example: Gain ratio - Split information

## ■ Example:

□ Humidity={High, Normal}

$$SplitInformation(S, Humidity) = -\frac{7}{14} \times \log_2\left(\frac{7}{14}\right) - \frac{7}{14} \times \log_2\left(\frac{7}{14}\right) = 1$$

□ Wind={Weak, Strong}

$$SplitInformation(S, Wind) = -\frac{8}{14} \times \log_2\left(\frac{8}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) = 0.9852$$

□ Outlook = {Sunny, Overcast, Rain}

$$SplitInformation(S, Outlook) = -\frac{5}{14} \times \log_2\left(\frac{5}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{5}{14} \times \log_2\left(\frac{5}{14}\right) = 1.5774$$

$$SplitInfo(S, A) = - \sum_{v \in Values(A)} P_v \bullet \log_2(P_v) = - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \bullet \log_2\left(\frac{|S_v|}{|S|}\right)$$

Training set

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
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

# Gain ratio characteristics

- Gain ratio normalizes the gain by splitting on A by the split information of A:

$$\text{GainRatio}(S, A) = \frac{\text{Gain}(S, A)}{\text{SplitInfo}(S, A)}$$

Measures the information  
w.r.t. classification

Measures the information  
generated by splitting S into  
|Values(A)| partitions

$$\text{SplitInfo}(S, A) = - \sum_{v \in \text{Values}(A)} P_v \cdot \log_2(P_v) = - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \cdot \log_2\left(\frac{|S_v|}{|S|}\right)$$

- **Low split info**: few partitions hold most of the tuples (peaks)
- **High split info**: partitions have more or less the same size (uniform)
- If an attribute produces many splits  $\rightarrow$  high SplitInfo()  $\rightarrow$  low GainRatio().
  - This is the case for e.g., the ID attribute
- The attribute with the **maximum gain ratio** is selected as the splitting attribute



# Attribute selection measure: Gini Index

- Used in CART (Breiman et al, 1984)
- Let a dataset  $S$  containing examples from  $k$  classes. Let  $p_j$  be the probability of class  $j$  in  $S$ . The **Gini Index** of  $S$  is given by:

$$\text{Gini}(S) = 1 - \sum_{j=1}^k p_j^2$$

- Gini index considers a **binary split** for each attribute  $A$ . Let  $S$  is split based on  $A$  into two subsets  $S_1$  and  $S_2$ . Then:

$$\text{Gini}(S, A) = \frac{|S_1|}{|S|} \text{Gini}(S_1) + \frac{|S_2|}{|S|} \text{Gini}(S_2)$$

- We want to evaluate the reduction in the impurity of  $S$  based on  $A$ :

$$\Delta \text{Gini}(S, A) = \text{Gini}(S) - \text{Gini}(S, A)$$

- The attribute  $A$  that provides the smallest  $\text{Gini}(S, A)$  (or the **largest reduction in impurity**) is chosen to split the node

# Attribute selection measure: Gini Index – a small example

- Let  $D$  has 14 instances
  - ▣ 9 instances in the class: buys\_computer = “yes”
  - ▣ 5 instances in the class: buys\_computer = “no”
- The Gini Index of  $D$  is:



What is the Gini Index of  $D$ ?

$$\text{Gini}(D) = 1 - \sum_{j=1}^k p_j^2 \quad \Rightarrow \quad \text{Gini}(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

# Attribute selection measure: Gini Index for non-binary data

- Gini index considers a **binary split** for each attribute  $A$
- How to find the binary splits for non-binary attributes?
  - For a **categorical attribute  $A$** , we consider all possible subsets that can be formed based on the values of  $A$  (next slides)
  - For a **numerical attribute  $A$** , we find the split points of  $A$  (next slides)

# Attribute selection measure: Gini index for categorical attributes

- Let the categorical attribute “Income” = {low, medium, high} .
- To generate the binary splits for “Income”, we check **all possible subsets**:
  - ({low,medium} and {high})
  - ({low,high} and {medium})
  - ({medium,high} and {low})
- **For each subset**, we check the Gini Index of setting up a split in that subset
  - $\text{Gini}\{\text{low,medium}\} \text{ and } \{\text{high}\}(D) = ?$
  - $\text{Gini}\{\text{low,high}\} \text{ and } \{\text{medium}\}(D) = ?$
  - $\text{Gini}\{\text{medium,high}\} \text{ and } \{\text{low}\}(D) = ?$
- The split that provides the smallest  $\text{Gini}(S, \text{Asplit})$  (or **the largest reduction in impurity**) is chosen to split the node

# Attribute selection measure: Gini index for categorical attributes

- Let us compute the Gini Index for the ( $\{low, medium\}$  and  $\{high\}$ ) split.
- Let us assume that this split results in  $D_1$  (#10 instances: 6+, 4-) and  $D_2$  (#4 instances: 1+, 3-)

$$\begin{aligned} Gini_{\{low, medium\} and \{high\}}(D) &= \left(\frac{10}{14}\right)Gini(D_1) + \left(\frac{4}{14}\right)Gini(D_2) \\ &= \frac{10}{14}\left(1 - \left(\frac{6}{10}\right)^2 - \left(\frac{4}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{1}{4}\right)^2 - \left(\frac{3}{4}\right)^2\right) \\ &= 0.450 \end{aligned}$$

- Similarly, for the remaining binary split partitions:

$$Gini_{\{low, high\} and \{medium\}}(D) = 0.315$$

$$Gini_{\{medium, high\} and \{low\}}(D) = 0.300$$



Which split should we choose?

$$Gini(S) = 1 - \sum_{j=1}^k p_j^2$$

$$Gini(S, A) = \frac{|S_1|}{|S|}Gini(S_1) + \frac{|S_2|}{|S|}Gini(S_2)$$

$$\Delta Gini(S, A) = Gini(S) - Gini(S, A)$$

# Attribute selection measure: Gini index for numerical attributes

- Let attribute  $A$  be a continuous-valued attribute
- Must determine the best split point  $t$  for  $A$
- Methodology

<i>Temperature:</i>	40	48	60	72	80	90
---------------------	----	----	----	----	----	----

- Sort the values of  $A$  in increasing order
- Identify adjacent examples that differ in their target classification
  - Typically, every such pair suggests a potential split threshold  $t = (a_i + a_{i+1})/2$
- Select the threshold  $t$  that yields the best value of the splitting criterion.

<i>Temperature:</i>	40	48	60	72	80	90
<i>PlayTennis:</i>	No	No	Yes	Yes	Yes	No

<i>Temperature:</i>	40	48	60	72	80	90
<i>PlayTennis:</i>	No	No	Yes	Yes	Yes	No

$t = (48 + 60)/2 = 54$

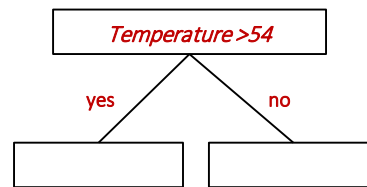
$t = (80 + 90)/2 = 85$

- 2 potential thresholds:  $\text{Temperature}_{>54}$ ,  $\text{Temperature}_{>85}$
- Compute the attribute selection measure for both
- Choose the best one ( $\text{Temperature}_{>54}$  here)

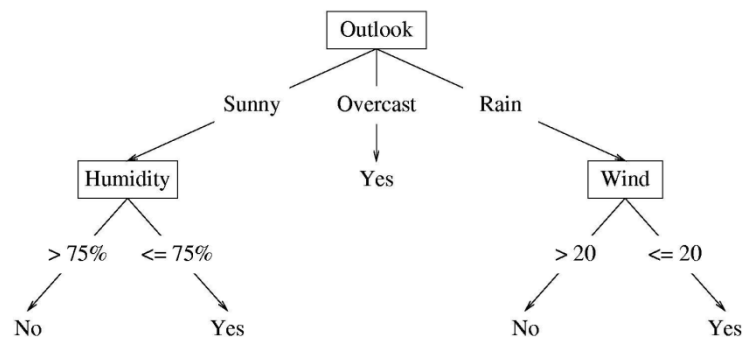
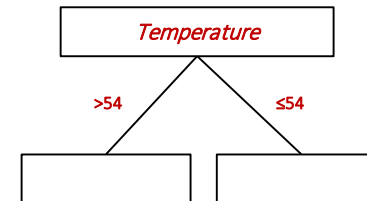
# Attribute selection measure: Gini index for numerical attributes

- Let  $t$  be the threshold chosen from the previous step ( $t=54$ )
- Create a Boolean attribute based on  $A$  and threshold  $t$  with two possible outcomes: “yes”, “no”
  - $S_1$  is the set of tuples in  $S$  satisfying  $(A > t)$ , and  $S_2$  is the set of tuples in  $S$  satisfying  $(A \leq t)$

How it looks



or



An example of a tree for the play tennis problem when attributes Humidity and Wind are continuous

# Comparing Attribute Selection Measures

- The three measures, are commonly used and in general, return good results but
  - Information gain  $\text{Gain}(S,A)$ :
    - biased towards multi-valued attributes
  - Gain ratio  $\text{GainRatio}(S,A)$  :
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - Gini index:
    - biased to multivalued attributes
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions
- Other measures also exist
  - «most previously published empirical results concluded that it is not possible to decide which one of the two tests to prefer», Theoretical Comparison between the Gini Index and Information Gain Criteria, Raileanu and Stoffel, 2004.  
<https://link.springer.com/article/10.1023/B:AMAI.0000018580.96245.c6>

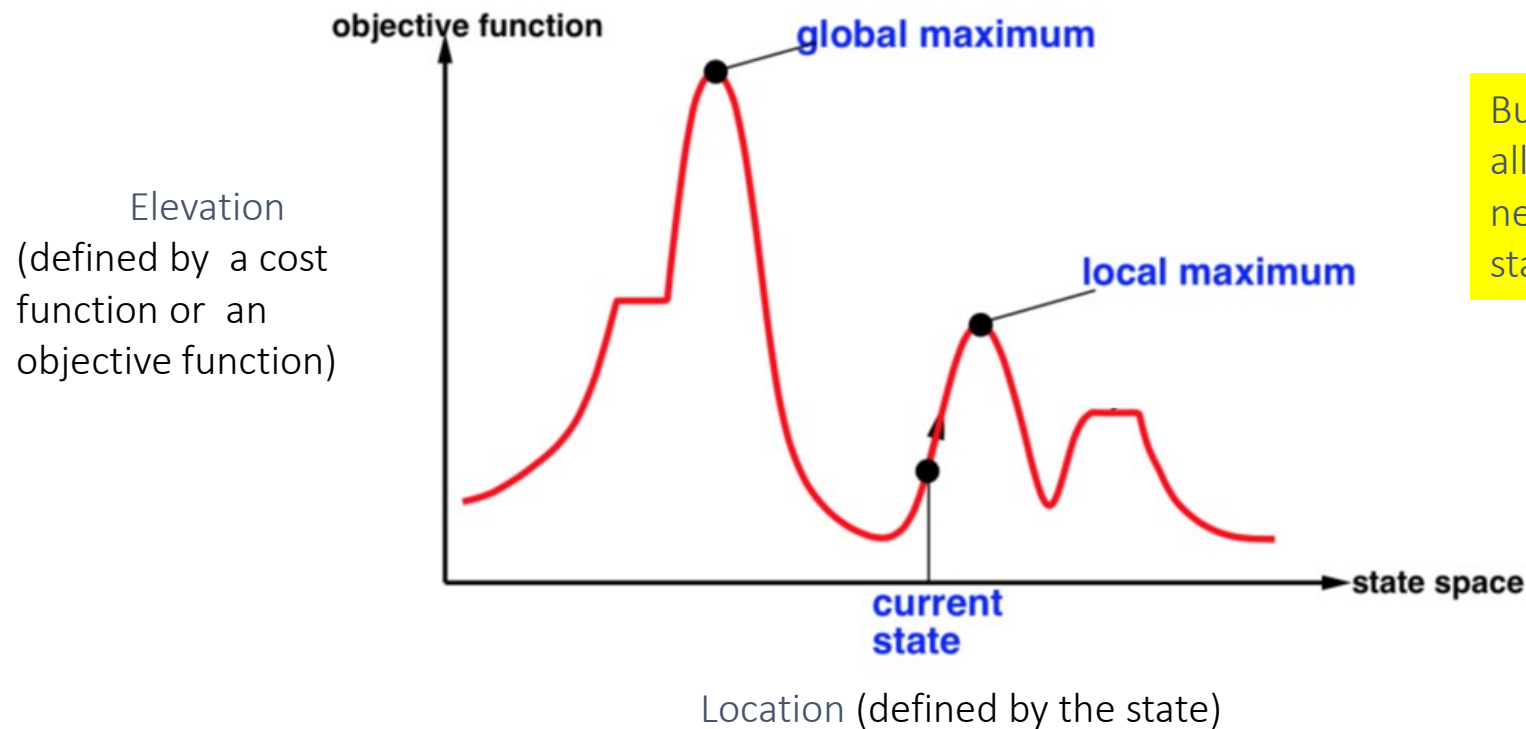


# Decision tree classifiers outline

- In this lecture, we will cover
  - Representation
  - How to build a tree/ Splitting attributes
  - Hypothesis space and inductive bias
  - Decision boundary
  - When to consider decision trees
  - Overfitting

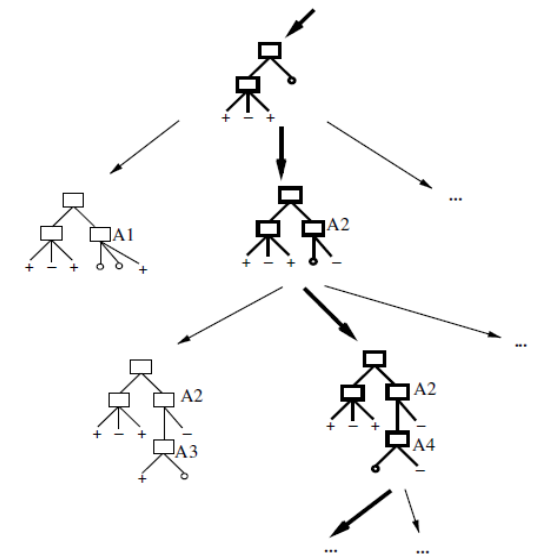
# Hypothesis space and inductive bias in decision trees

- In case of decision trees,  $h()$  is represented by a decision tree
- **Hypothesis class/space**: the set of possible decision trees
- How do we search in  $H \rightarrow$  local search



# Hypothesis space and inductive bias in decision trees

- In classification we want to learn a target function/ mapping  $h(): X \rightarrow Y$ 
  - In case of decision trees,  $h()$  is represented by a decision tree
- **Hypothesis class/space**: the set of possible decision trees
- **Search method**: How do we search in  $H$ 
  - hill-climbing
    - Looks only to immediate good neighbors and not beyond
  - greedy approach
  - No backtracking: split attributes are fixed
  - from simple to complex (top-down)
  - Only a single current hypothesis is maintained
- Evaluation function: Information gain/Gini Index etc
- Batch learning: use all training data



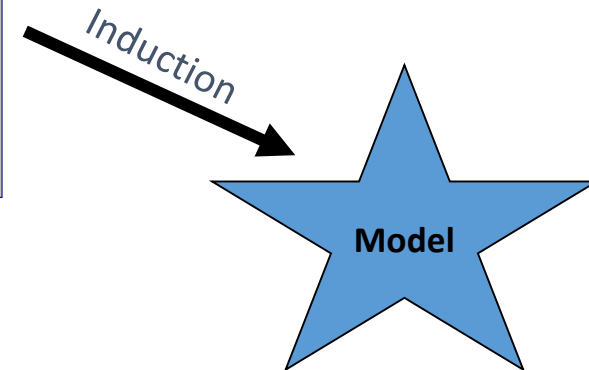
# Inductive bias in decision tree learning

- **Inductive bias**: the set of assumptions that, together with the training data, deductively justify the classifications assigned by the learner to future instances.
- **Inductive bias in ID3**: What is the policy by which ID3 generalizes from observed training examples to classify unseen instances?
  - It chooses the first acceptable tree it encounters in its simple-to-complex, hill climbing search through the space of possible trees.
  - trees that place high information gain attributes close to the root are preferred over those that do not.
  - shorter trees are preferred over larger trees.


- **Induction**: makes broad generalizations from specific observations
  - Generates new theory emerging from the data

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set



# Why prefer shorter hypotheses?

- **Occam's Razor principle**: Prefer the simplest hypothesis that fits the data
- Scientists seem to do that: Physicists, for example, prefer simple explanations for the motions of the planets, over more complex explanations.
- Why shorter hypotheses are preferred? 
  - Since there are fewer short hypotheses than long ones, it is less likely that one will find a short hypothesis that coincidentally fits the training data
  - In contrast, there are often many very complex hypotheses that fit the current training data but fail to generalize correctly to subsequent data.

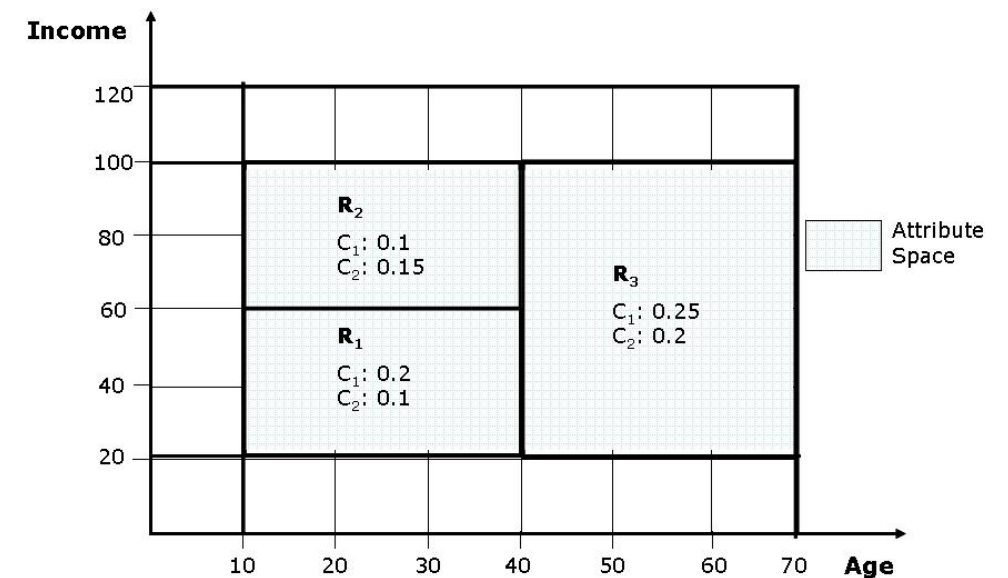
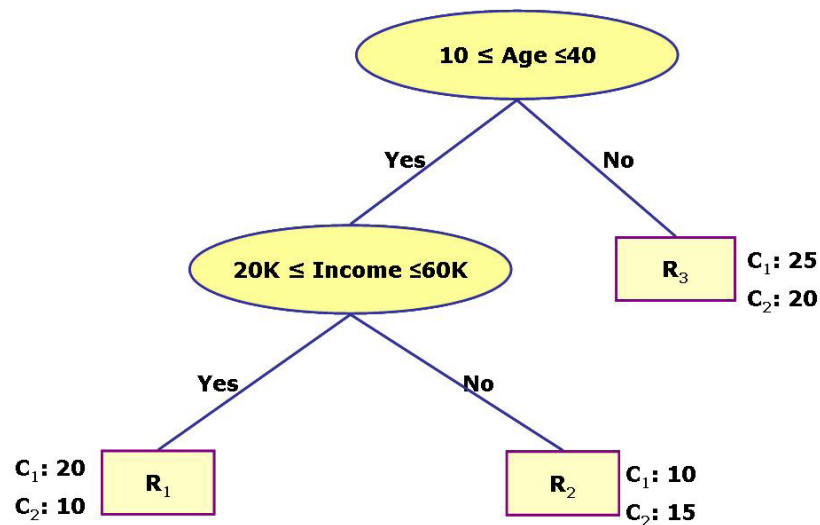
# Decision tree classifiers outline

- In this lecture, we will cover
  - Representation
  - How to build a tree/ Splitting attributes
  - Hypothesis space and inductive bias
  - **Decision boundary**
  - When to consider decision trees
  - Overfitting

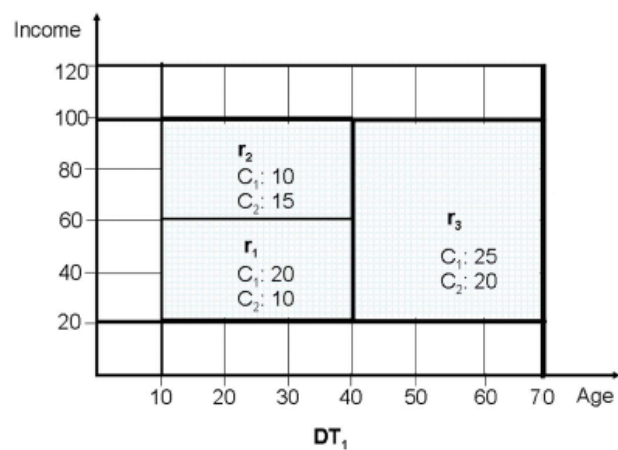
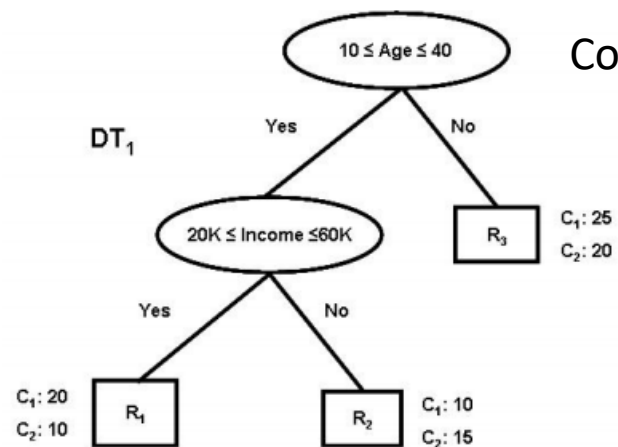
# Decision tree decision boundary



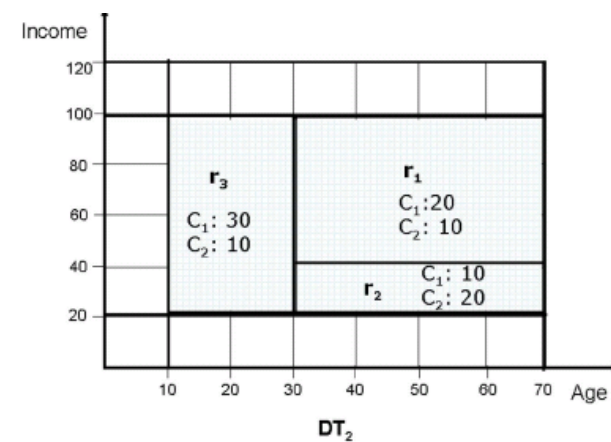
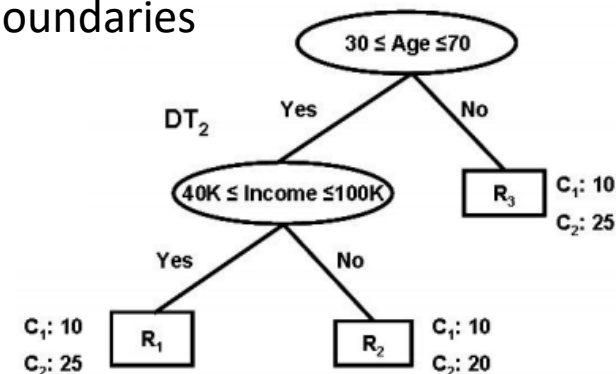
- DTs partition the feature space into **axis-parallel hyper-rectangles**
  - the so-called decision regions
  - A region can be labeled based on (majority) class and/or the region class distribution.
- **Decision boundary**: the border line between two neighboring regions of different classes



# Comparing decision trees



Comparing decision tree boundaries

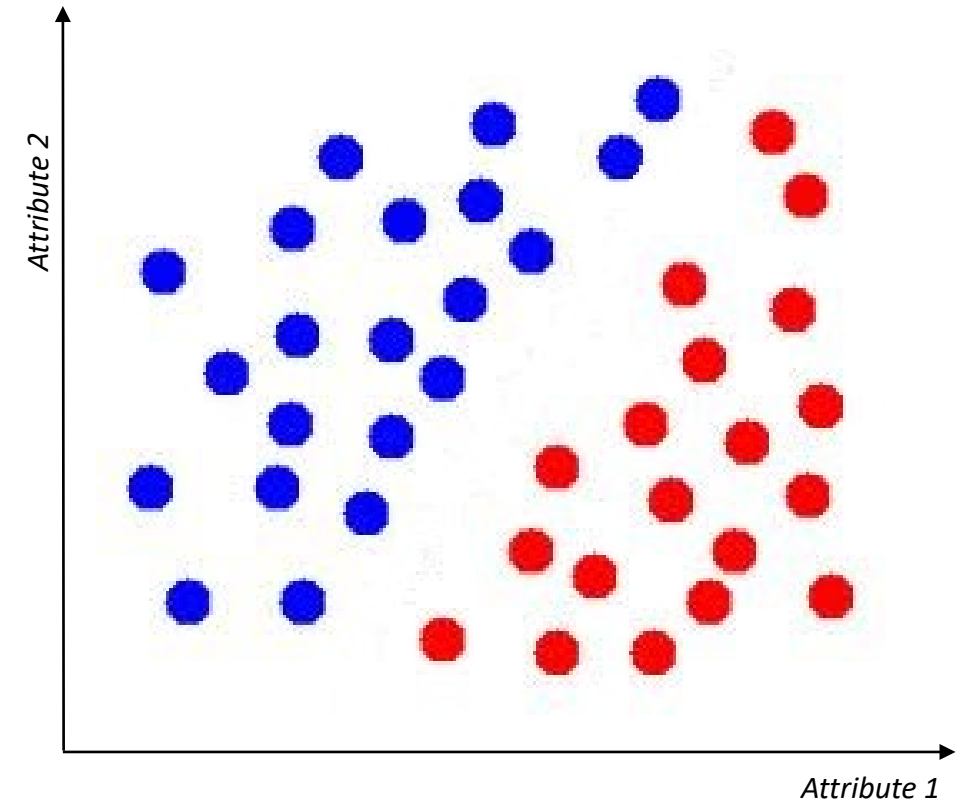
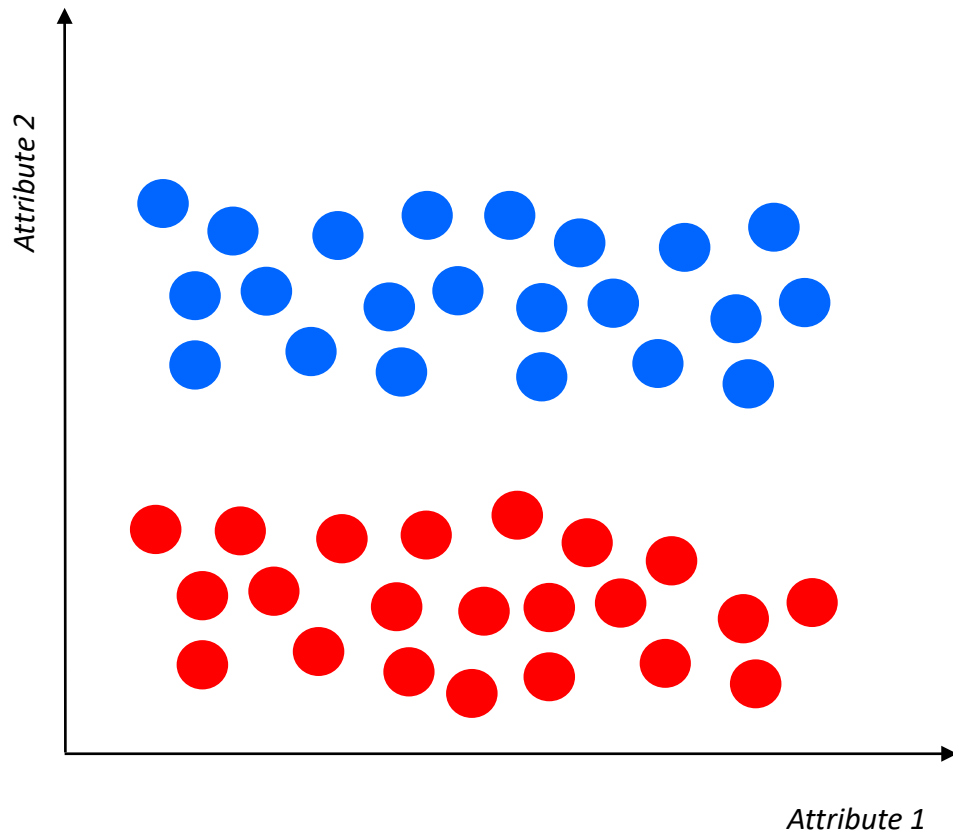




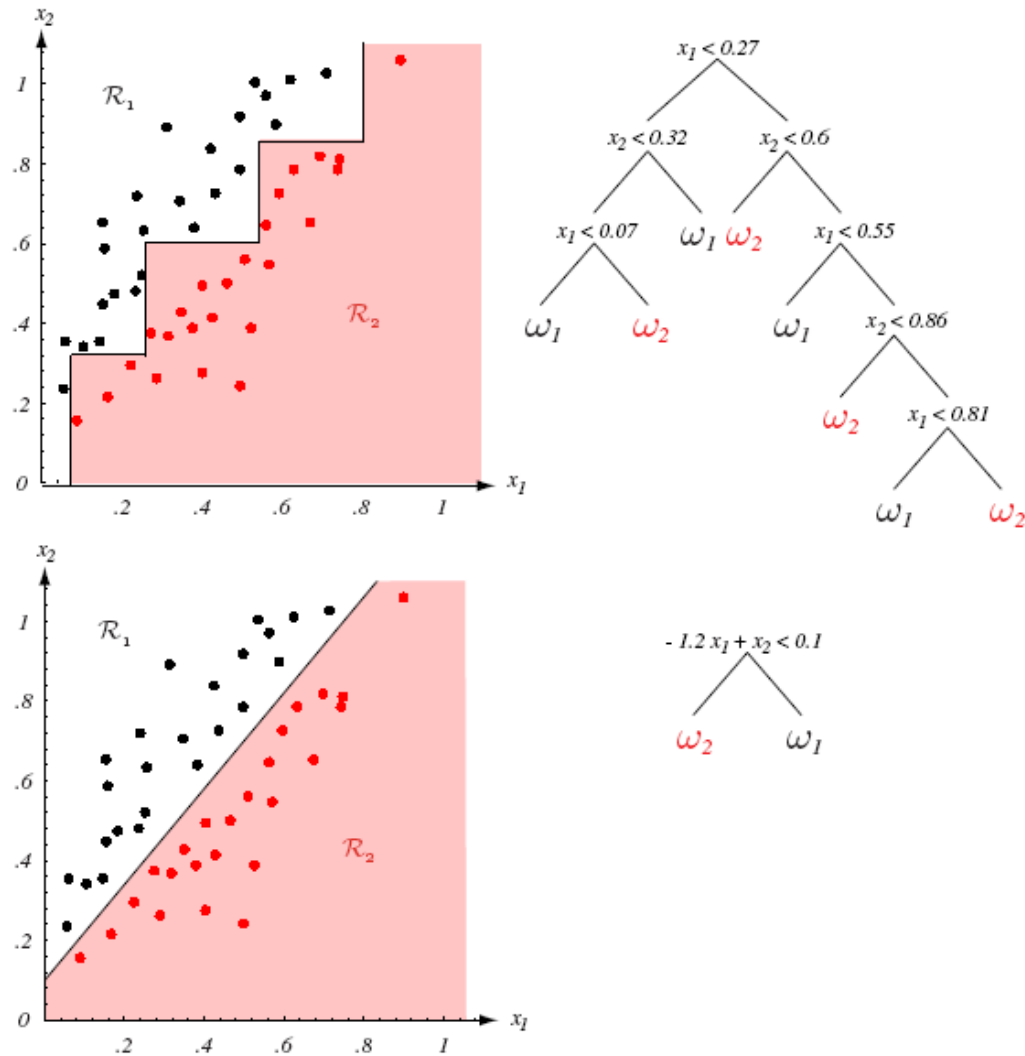
## Short break (5')



- How a decision tree based decision boundary would look like for the examples below? Why?



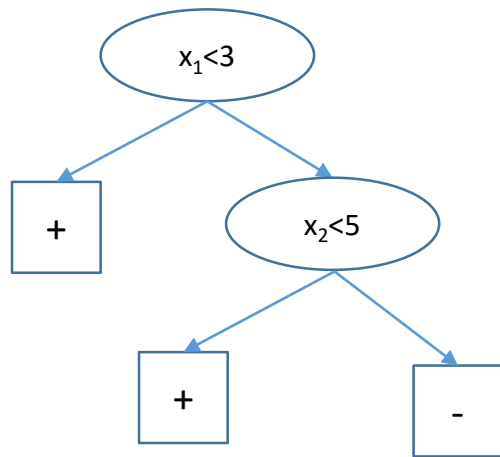
# Limitations of decision trees



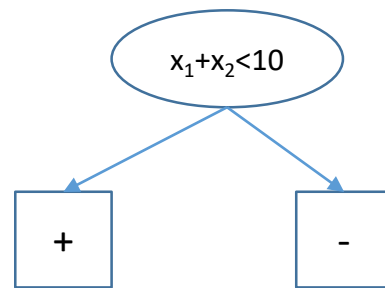
Source: <http://slideplayer.com/slide/5806873/>

# Orthogonal vs oblique decision trees

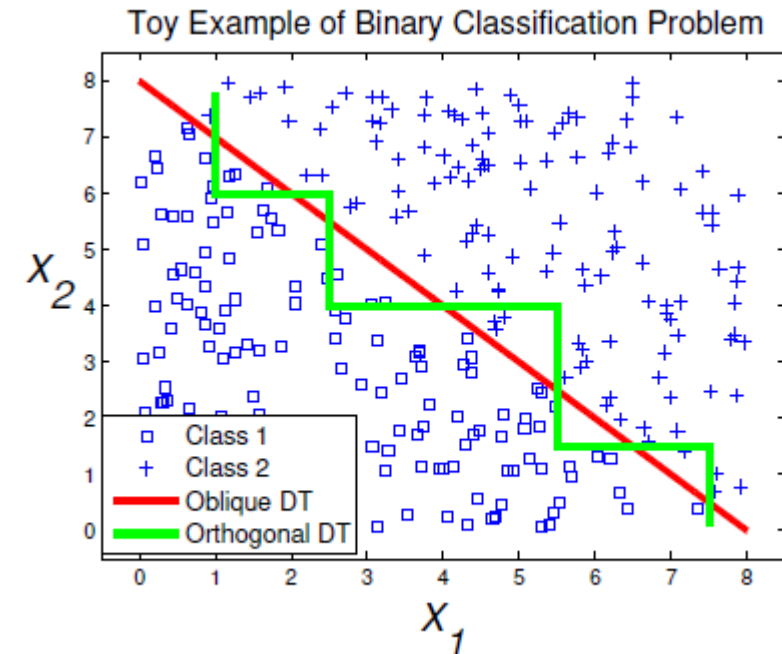
- There exist solutions to simplify the boundary structure, like the so-called **oblique decision trees**
  - In oblique DTs, the splitting criterion might involve  $>1$  attributes
  - More expressive representation
  - Finding optimal test condition is computationally more expensive



Orthogonal split



Oblique split



Source: <https://sites.google.com/site/zhangleuestc/incremental-oblique-random-forest>

# Decision tree classifiers outline

- In this lecture, we will cover
  - Representation
  - How to build a tree/ Splitting attributes
  - Hypothesis space and inductive bias
  - Decision boundary
  - When to consider decision trees
  - Overfitting

# When to consider decision trees

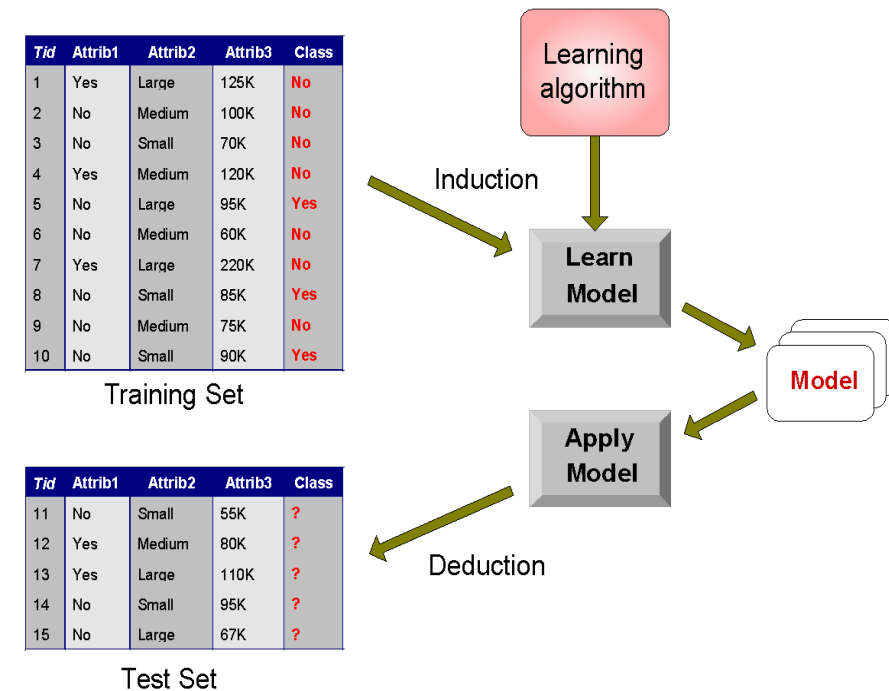
- Instances are represented by attribute-value pairs
  - Instances are represented by a fixed number of attributes, e.g. outlook, humidity, wind and their values, e.g. (wind=strong, outlook =rainy, humidity=normal)
  - The easiest situation for a DT is when attributes take a small number of disjoint possible values, e.g. wind={strong, weak}
  - There are extensions for numerical attributes also, e.g. temperature, income.
- The class attribute has discrete output values
  - Usually binary classification, e.g. {yes, no}, but also for more class values, e.g. {pos, neg, neutral}
- The training data might contain errors
  - DTs are robust to errors: both errors in the class values of the training examples and in the attribute values of these examples
- The training data might contain missing values
  - DTs can be used even when some training examples have some unknown attribute values

# Decision tree classifiers outline

- In this lecture, we will cover
  - Representation
  - How to build a tree/ Splitting attributes
  - Hypothesis space and inductive bias
  - Decision boundary
  - When to consider decision trees
  - Overfitting

# Training vs generalization errors

- The errors of a classifier are divided into
  - **Training error:**
    - errors committed in the training set
  - **Generalization error:**
    - the expected error of the model on previously unseen examples
    - We use a test set to estimate this (test error)
- A good classifier must
  - Fit the training data &
  - Accurately classify records never seen before
- So, low training error & low generalization error



# Model overfitting

Definition: Given a hypothesis class  $H$ , a hypothesis  $h \in H$  is said to overfit the training data if there exists some alternative hypothesis  $h' \in H$ , such that  $h$  has smaller error than  $h'$  over the training examples, but  $h'$  has a smaller error than  $h$  over the entire distribution of instances.

- Consider the error of  $h$  over:
  - The training set:  $L_D(h)$
  - The entire distribution:  $L_{P(X),f}(h)$
- According to the definition,  $h$  overfits the training data if there is an alternative  $h'$  in  $H$  such that:

$$L_D(h) < L_D(h') \quad \& \quad L_{P(X),f}(h) > L_{P(X),f}(h')$$

- How to check for overfitting:
  - A model that fits the training data well (low training error) but has a poor generalization power (high generalization error, in the test set)



# Overfitting in decision trees

- How it is manifested
  - Overfitting results in decision trees that are more complex than necessary

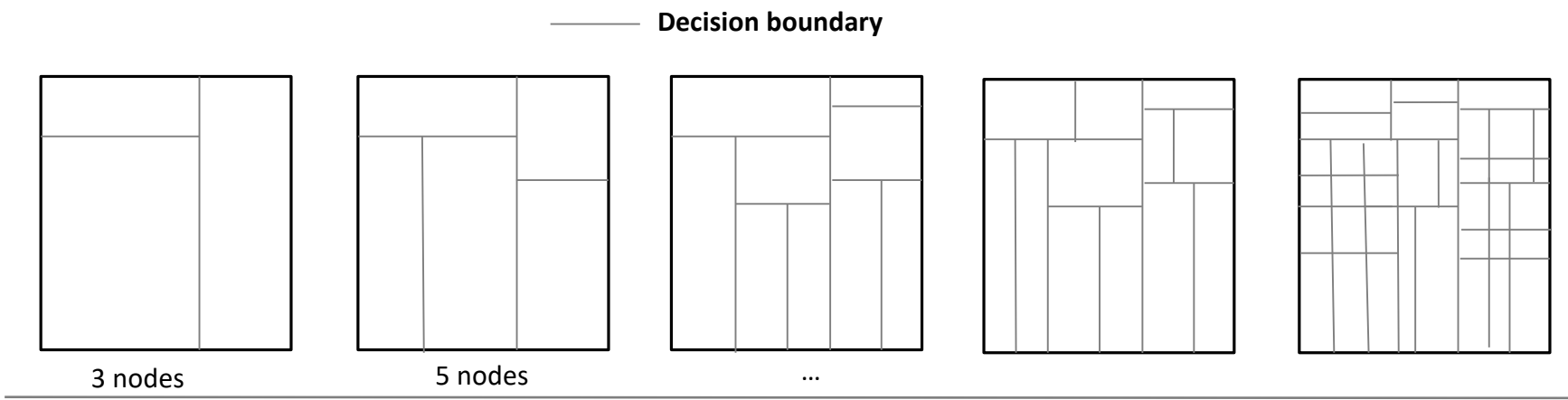
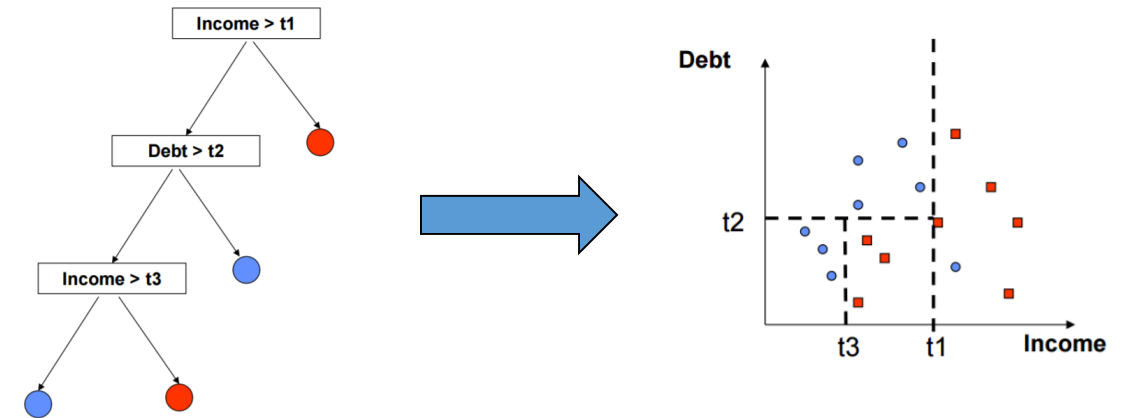


Source: <http://www.learnbymarketing.com/tutorials/rpart-decision-trees-in-r/>

- and of course,
  - Very good performance in the training (already seen) samples
  - Poor accuracy for unseen samples

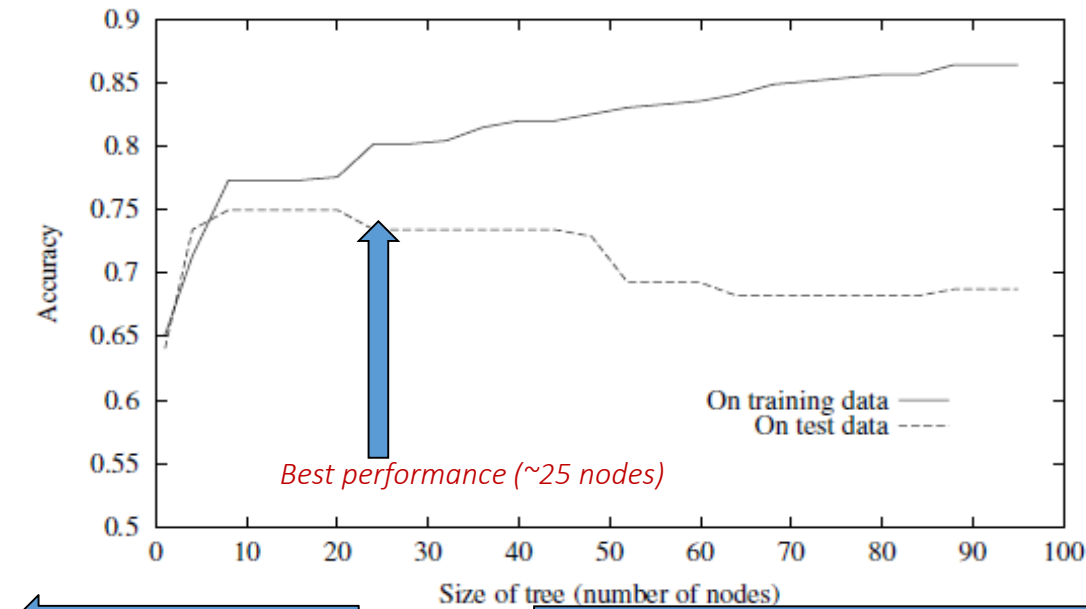
# Another way to look at overfitting: The effect of overfitting on DT partitioning

- Recall that a DT partitions the space into axis-parallel hyper-rectangles
- With overfitting, the decision regions become very small



# Overfitting & Underfitting

- Remember: A good classifier must have low training error and low generalization error
  - The training error can be decreased by increasing the model complexity
  - But, a complex, tailored to the training data model, will also have a high generalization error
- An example of how the decision tree complexity affects the performance of the classifier



*The model has yet to learn the true structure from the training data.*

*The model overspecializes to the training data*

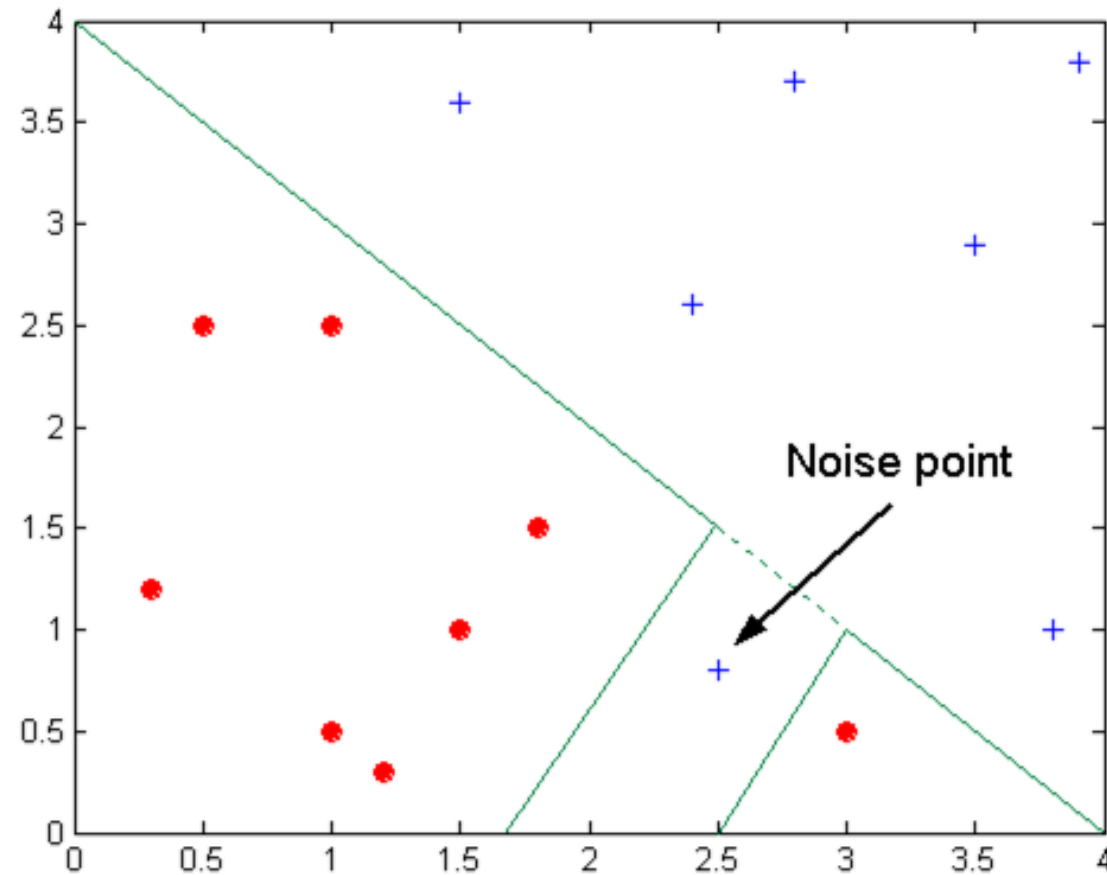
Model underfitting

Model overfitting

# Potential causes of model overfitting

- Overfitting due to presence of noise
- Overfitting due to lack of representative samples
- ...

# Overfitting due to presence of noise



The decision boundary is distorted by the noise point.

# Overfitting due to presence of noise – an example in DTs

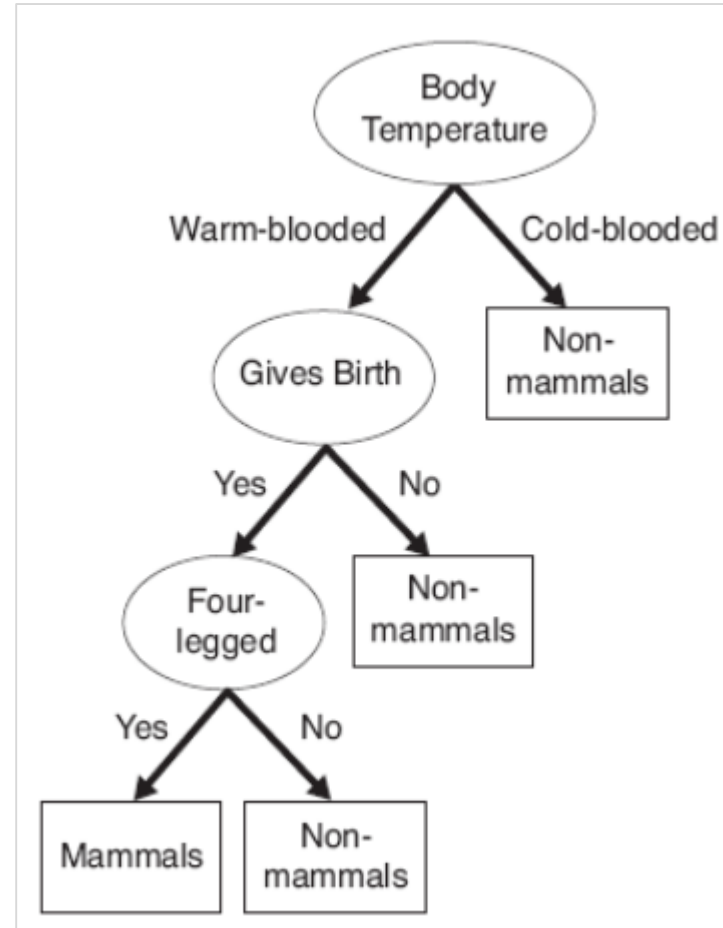
Training set

(\* stands for missclassified instances)

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
porcupine	warm-blooded	yes	yes	yes	yes
cat	warm-blooded	yes	yes	no	yes
bat	warm-blooded	yes	no	yes	no*
whale	warm-blooded	yes	no	no	no*
salamander	cold-blooded	no	yes	yes	no
komodo dragon	cold-blooded	no	yes	no	no
python	cold-blooded	no	no	yes	no
salmon	cold-blooded	no	no	no	no
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no

Test set

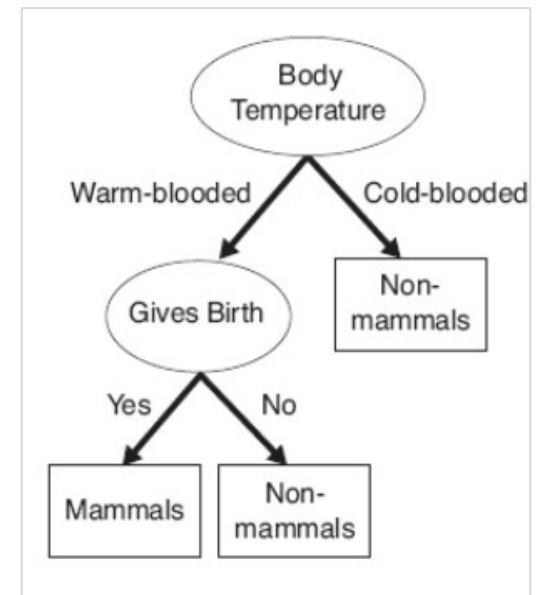
Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
human	warm-blooded	yes	no	no	yes
pigeon	warm-blooded	no	no	no	no
elephant	warm-blooded	yes	yes	no	yes
leopard shark	cold-blooded	yes	no	no	no
turtle	cold-blooded	no	yes	no	no
penguin	cold-blooded	no	no	no	no
eel	cold-blooded	no	no	no	no
dolphin	warm-blooded	yes	no	no	yes
spiny anteater	warm-blooded	no	yes	yes	yes
gila monster	cold-blooded	no	yes	yes	no



$M_1$

Training error: 0

Test error: 30%

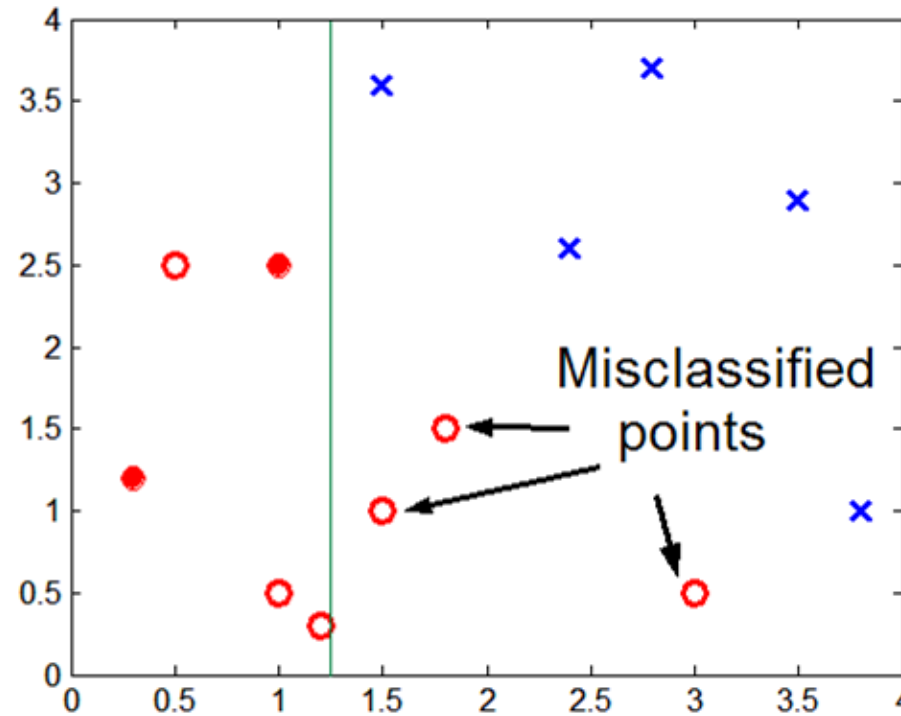


$M_2$

Training error: 20%

Test error: 10%

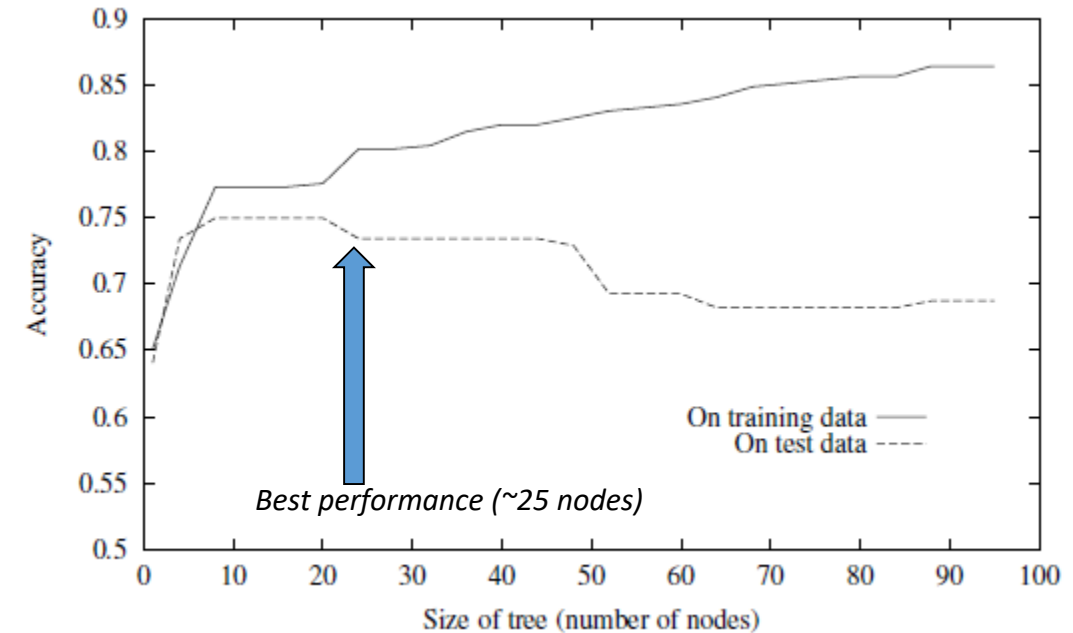
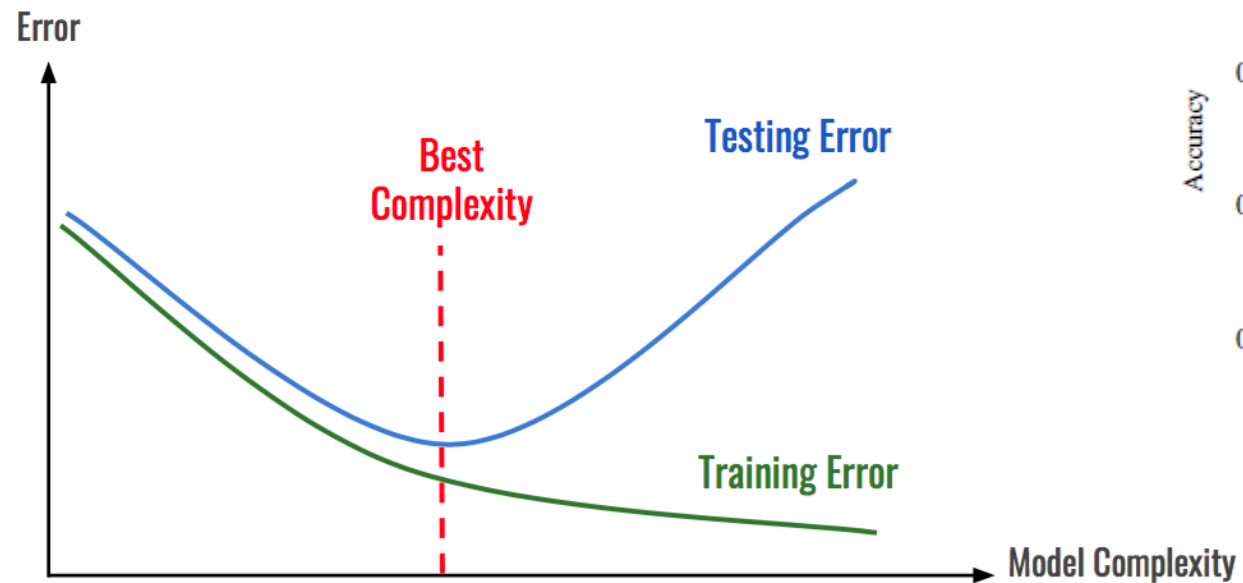
# Overfitting due to lack of representative samples



- Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region
  - Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task

# How to prevent overfitting (in general)?

- **Regularization**: Control model complexity, i.e, make your model simpler (Recall Occam's razor)
- **Feature selection**: Remove irrelevant features. Some models like decision trees have built-in feature selection mechanisms.
- **More data**: sometimes the training set is not enough
- ...





# Avoiding overfitting in Decision Trees - Pruning

- Prevent overfitting the training data → prune the decision tree
- Two strategies:
  - **Pre-pruning**  
stop growing a branch if it does not provide much information
  - **Post-pruning**  
take a fully-grown decision tree and discard parts that do not contribute much information
- Postpruning is preferred in practice—prepruning can “stop early”

# Avoiding overfitting in Decision Trees: Pre-pruning

- **Pre-pruning** (early stopping rule)
  - Stop the algorithm before it becomes a fully-grown tree
- **Pre-pruning techniques**
  - Based on a **minimum number of instances per node threshold**
    - C4.5 uses a simpler strategy, but combines it with post-pruning
    - Each parent of a leaf node must contain at least  $m$  examples
  - Based on a **statistical significance test**
    - Stop growing the tree when there is no *statistically significant association* between any attribute and the class at a particular node
      - Most popular test: chi-square test
    - ID3 used chi-squared test in addition to information gain
      - Only statistically significant attributes were allowed to be selected by the information gain procedure
- Risk: Pre-pruning may stop the growth process prematurely (early stopping)

# Avoiding overfitting in Decision Trees: Post-pruning



- First grow a full tree to capture all possible attribute interactions
- Later remove those that are due to chance

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

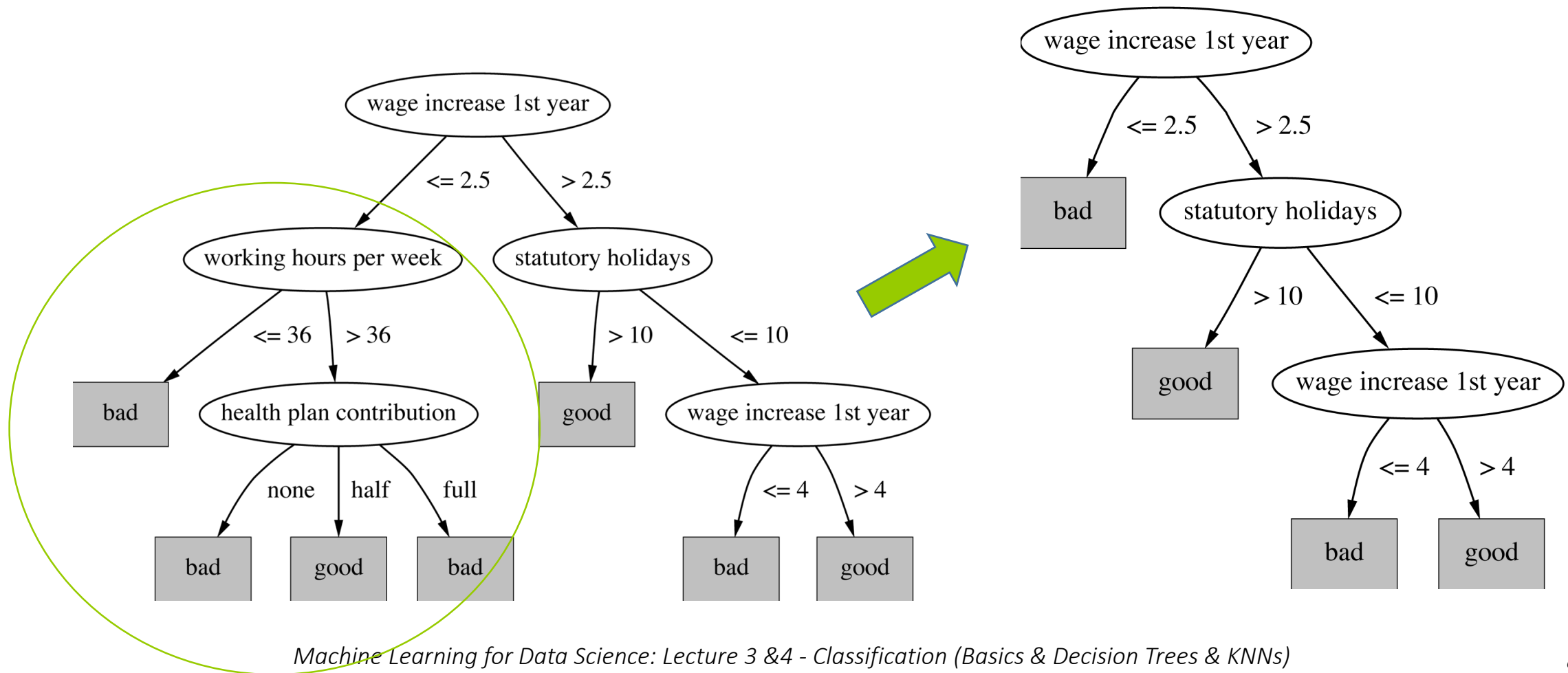
# Avoiding overfitting in Decision Trees: Post-pruning

- Two subtree simplification operations:
  - Subtree replacement
  - Subtree raising
- Possible performance evaluation strategies:
  - Error-based pruning
  - Complexity-based pruning
  - ...



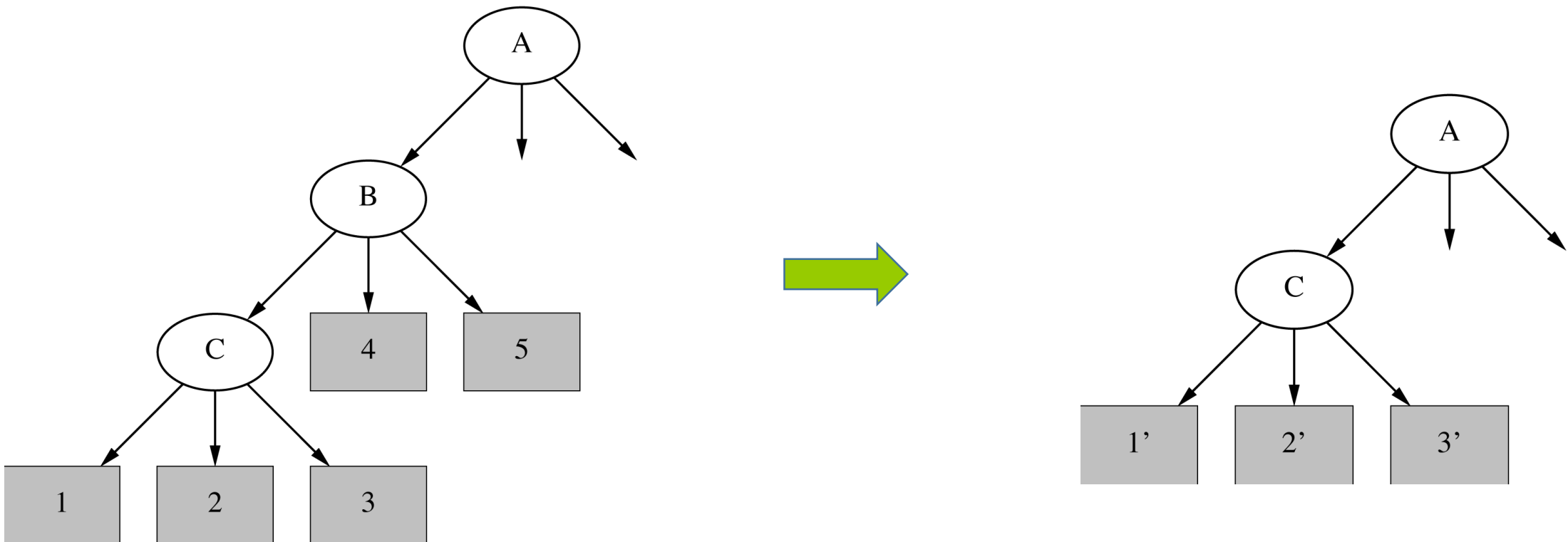
# Post-pruning simplification operations: Subtree replacement

- Works bottom-up
  - Consider replacing a tree only after considering all its subtrees



# Post-pruning simplification operations: Subtree raising

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



# Pruning strategies



- Error-based pruning

- Prune only if it does not increase the estimated error
  - Error on the training data is NOT a useful estimator (would result in almost no pruning)
  - Error is evaluated on an independent dataset (validation set/pruning set, more on validation set next week)

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

# Pruning strategies

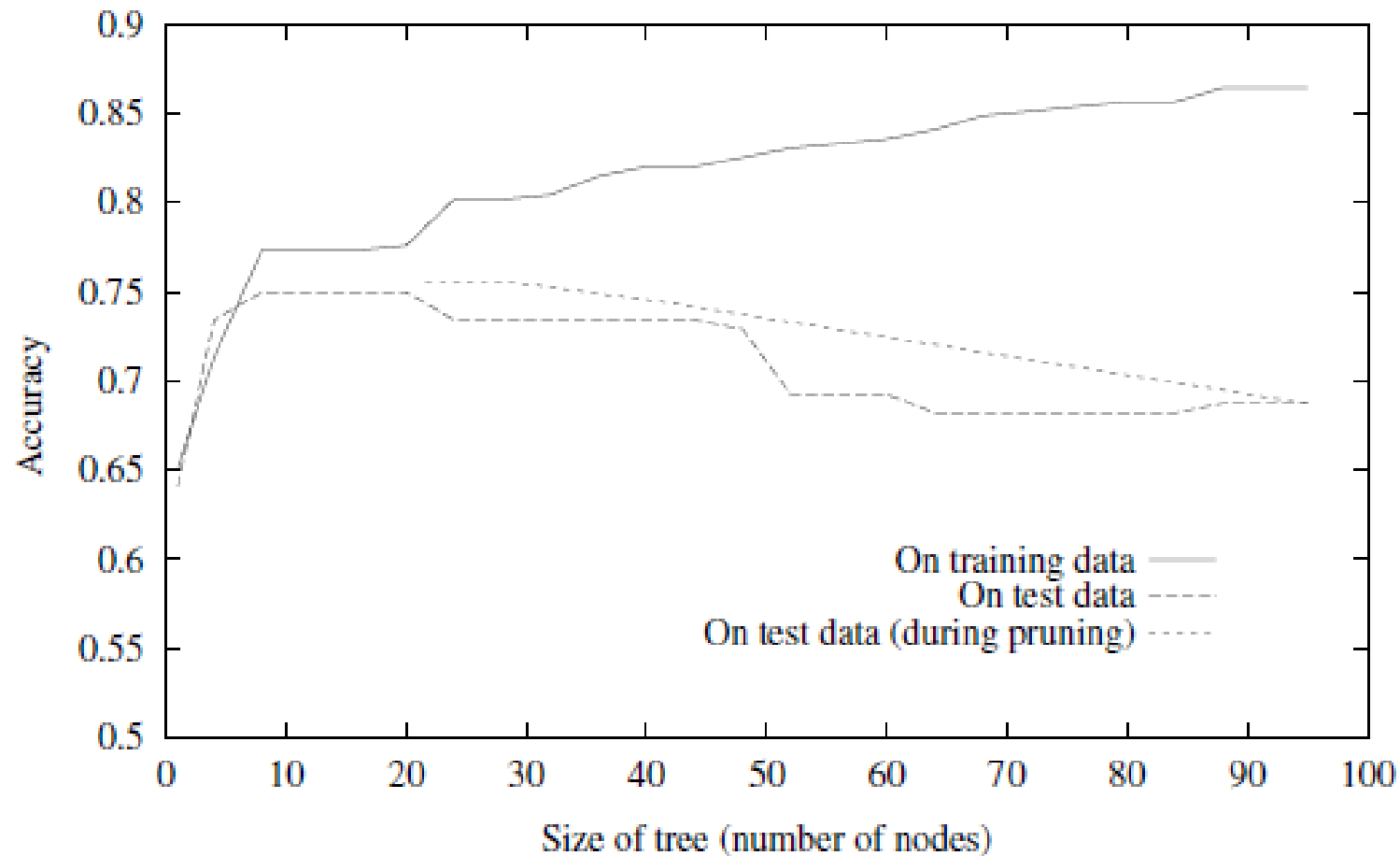


- Complexity-based pruning
  - Add complexity penalty to the performance measure
- For example, using Minimum Description Length (MDL)
  - [The MDL Principle] is based on the following insight: any regularity in a given set of data can be used to [compress the data](#), i.e. to describe it using fewer symbols than needed to describe the data literally. (Grünwald, 1998), from Wikipedia
  - MLD for pruning: the shortest description of the data (code length) is the best model
  - Improved cost function: minimize (size(tree) + size(misclassifications(tree)))



# Effect of reduced-error pruning on decision trees

- How the error in both training and test data evolves with the tree complexity; with and without pruning



# Overview and Reading w.r.t. basics and DTs

- Overview

- Classification basics
- Decision tree classifiers

- Reading

- Chapter 2: A gentle start, Understanding Machine Learning book by Shai Shalev-Schwartz and Shai Ben-David
- Chapter 3: Decision tree learning, Machine Learning book by Tom Mitchel
- Chapter 18: Decision trees, Understanding Machine Learning book by Shai Shalev-Schwartz and Shai Ben-David

# Acknowledgements

- The slides are based on
  - ❑ KDD I lecture at LMU Munich (Johannes Aßfalg, Christian Böhm, Karsten Borgwardt, Martin Ester, Eshref Januzaj, Karin Kailing, Peer Kröger, Eirini Ntoutsi, Jörg Sander, Matthias Schubert, Arthur Zimek, Andreas Züfle)
  - ❑ Introduction to Data Mining book slides at <http://www-users.cs.umn.edu/~kumar/dmbook/>
  - ❑ Pedro Domingos Machine Lecture course slides at the University of Washington
  - ❑ Machine Learning book by T. Mitchel slides at <http://www.cs.cmu.edu/~tom/mlbook-chapter-slides.html>
  - ❑ (DTs) J. Fürnkranz slides from TU Darmstadt (<https://www.ke.tu-darmstadt.de/lehre/archiv/ws0809/mlbm/>)
  - ❑ Thank you to all TAs contributing to their improvement, namely Vasileios Iosifidis, Damianos Melidis, Tai Le Quy, Han Tran.

# Outline

- Supervised learning
- Classification basics
- Decision trees
- Lazy vs Eager learners
- KNNs
- Things you should know from this lecture & reading material

# Lazy vs Eager learners

## ■ Eager learners

- Construct a classification model (based on a training set)
- Learned models are ready and eager to classify previously unseen instances
- e.g., decision trees, SVMs, MNB, neural networks

## ■ Lazy learners

- Simply store training data (with labels) and wait until a new instance arrives that needs to be classified
- No model is constructed.
- Known also as **instance-based learners**, because they store the training set
- e.g., *k*NN classifier

### Eager learners

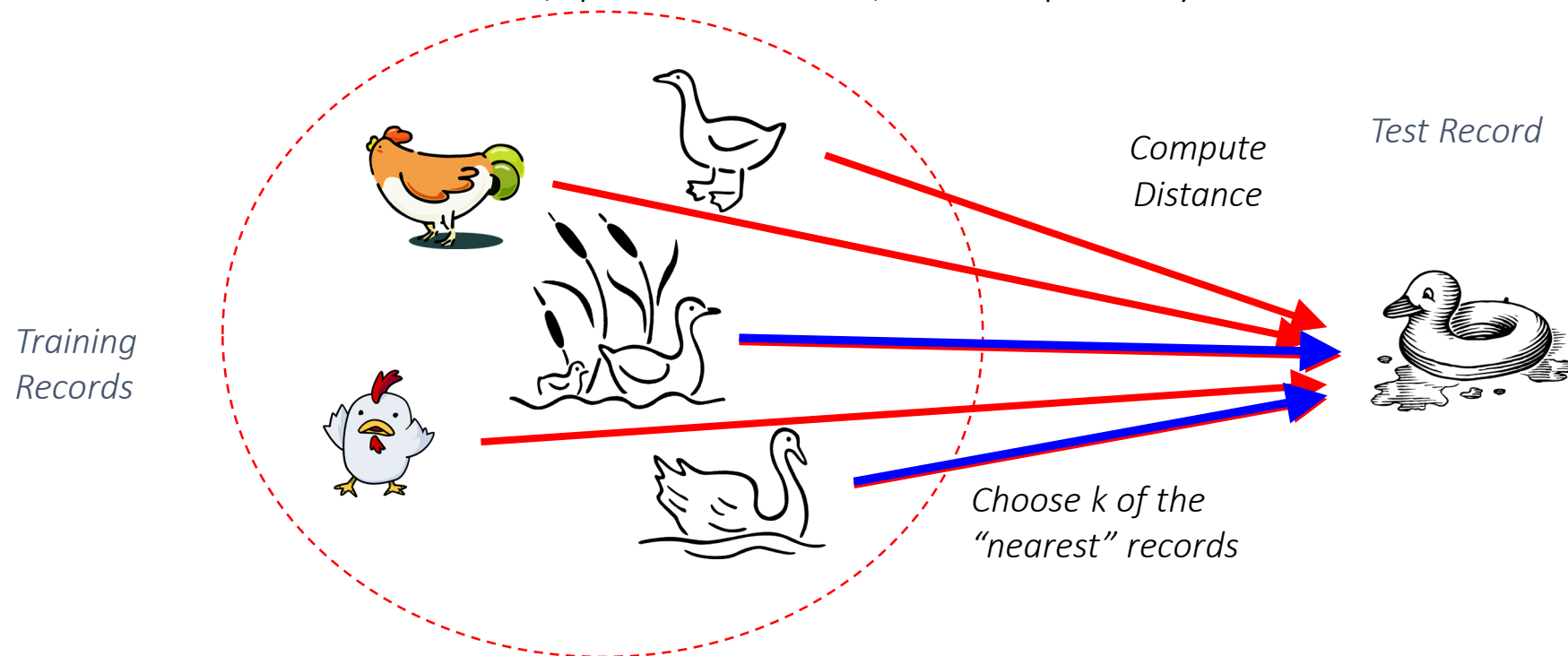
- Do lot of work on training data
- Do less work on classifying new instances

### Lazy learners

- Do less work on training data
- Do more work on classifying new instances

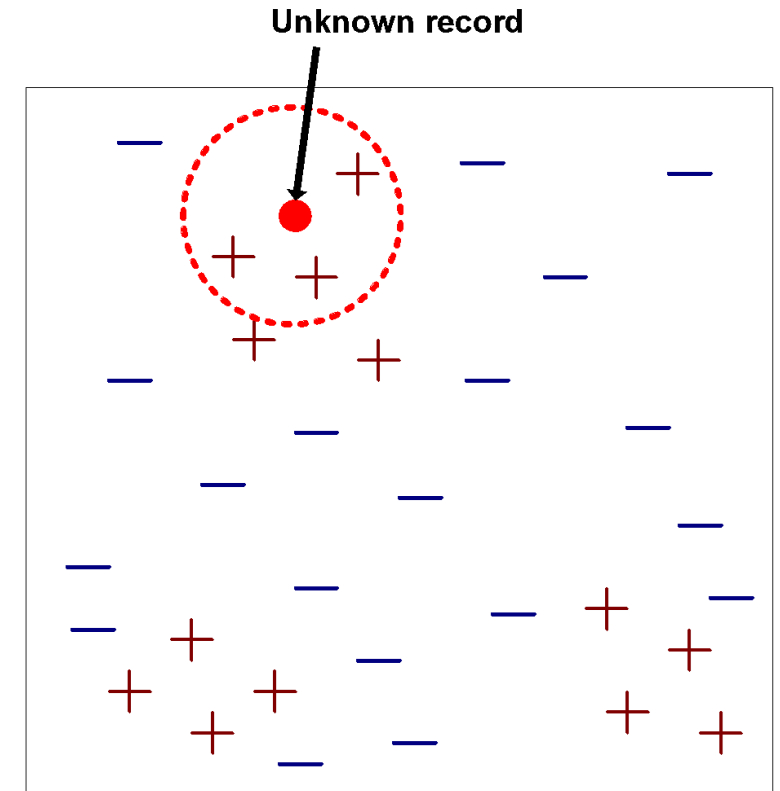
# k-Nearest Neighbor classifiers idea

- Let  $D$  be a training set of instances (i.e., with labels)
- Let  $X$  be a new instance for which we cant to predict its class label
- Nearest-neighbor classifiers compare  $X$  with its **most similar training instances** to decide on class
  - Basic idea: If it walks like a duck, quacks like a duck, then it's probably a duck



# k-Nearest Neighbor (kNN) classifiers

- Input:
  - A **training set  $D$**  (with known class labels)
  - A **distance measure** to compute the distance between two instances
  - The **number of neighbors  $k$**
- Classification: Given a new unknown instance  $X$ 
  - Compute distance to the training records, i.e.,  $\text{dist}(X, d), d \in D$
  - Identify the  $k$  nearest neighbors
  - Use the class labels of the  $k$  nearest neighbors to determine the class label of  $X$  (e.g., by taking majority vote)
- Complexity: It requires  $O(|D|)$  for each new instance

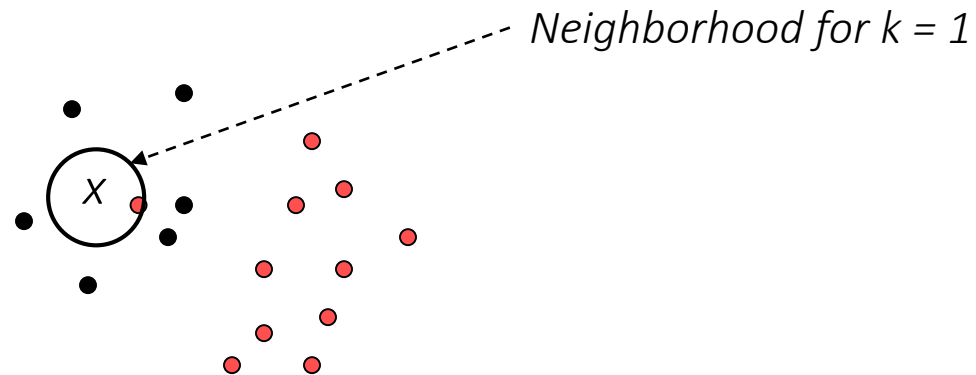


# Definition of nearest neighbors: parameter $k$

- An example with a too small  $k$ 
  - ▣ Extreme case:  $k=1$

❓ Why a small  $k$  is problematic?

too small  $k$ : high sensitivity to outliers



$X$ : unknown instance

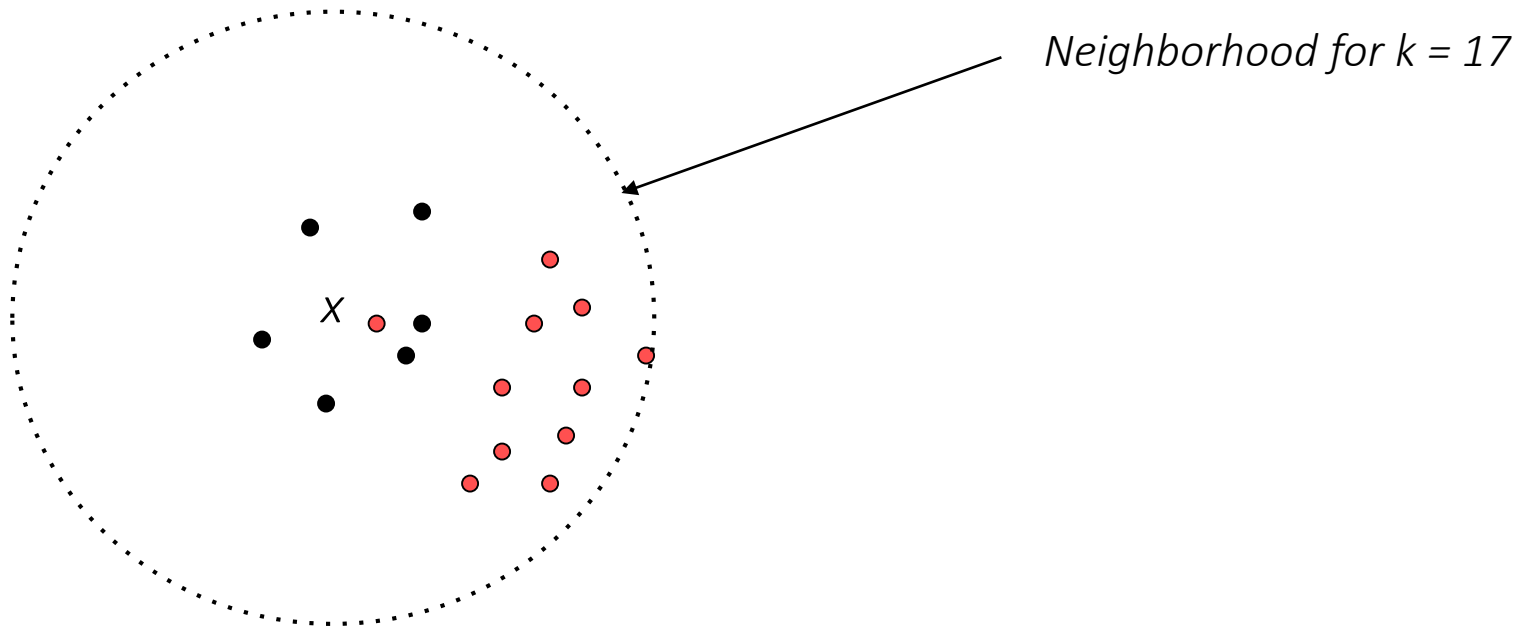


# Definition of nearest neighbors: parameter $k$

- An example with a too large  $k$ 
  - ▣ Extreme case:  $k=|D|$

⓪ Why a large  $k$  is problematic?

too large  $k$ : many objects from other classes in the resulting neighborhood



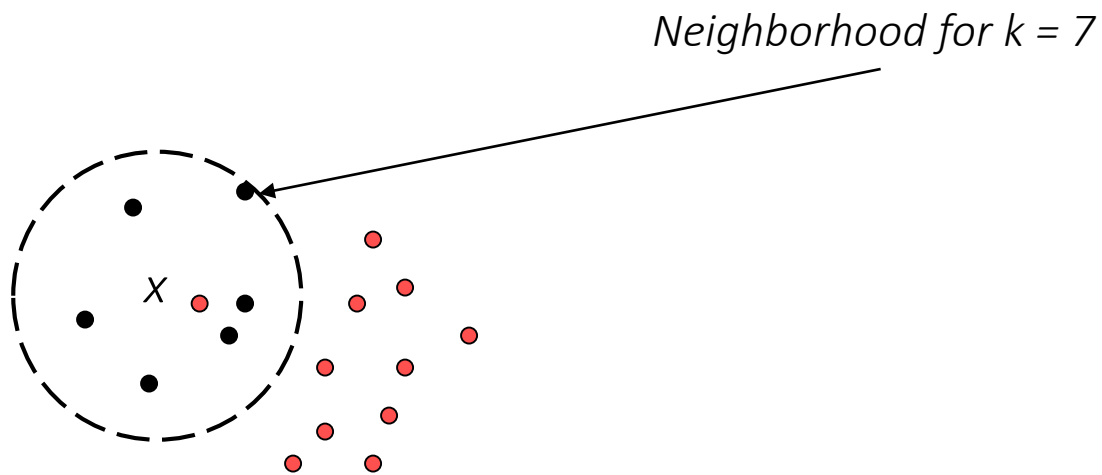
$X$ : unknown instance

# Definition of nearest neighbors: parameter $k$

- An example with an average  $k$

② Why an average  $k$  is best?

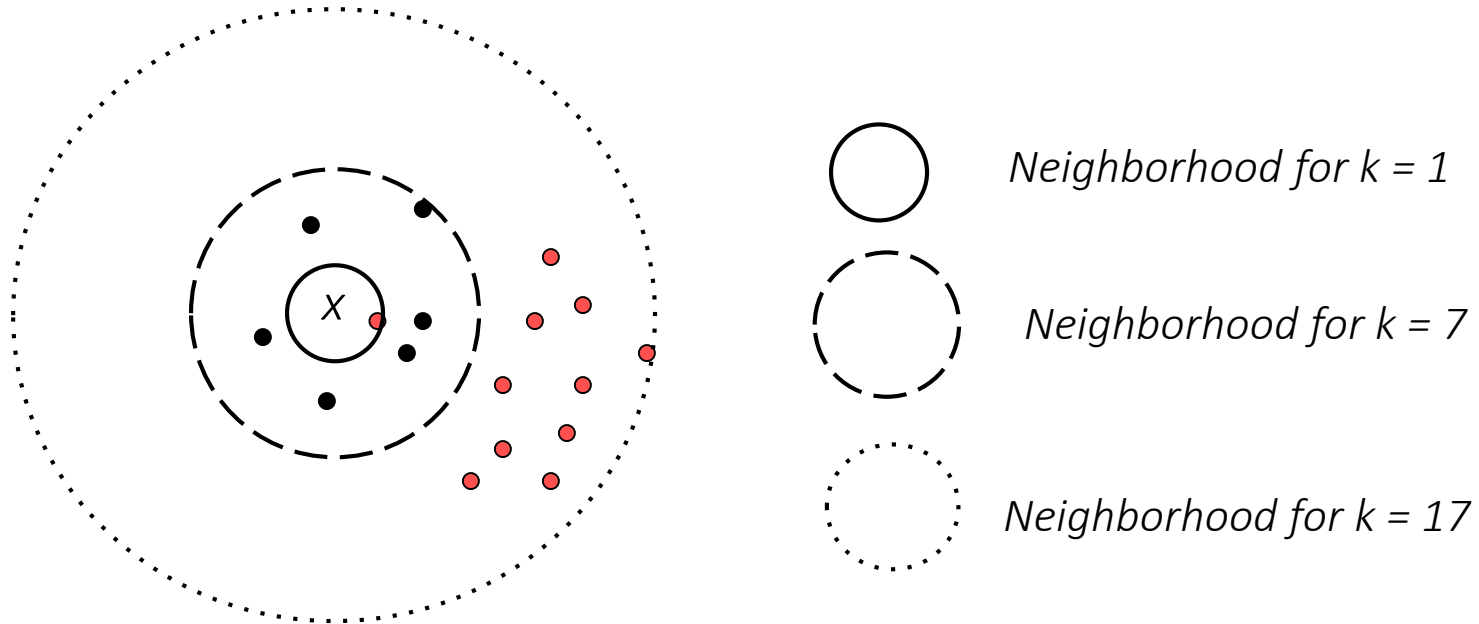
average  $k$ : highest classification accuracy



$X$ : unknown instance

# Definition of nearest neighbors: parameter $k$

- Overview of the effect of  $k$



$X$ : unknown instance

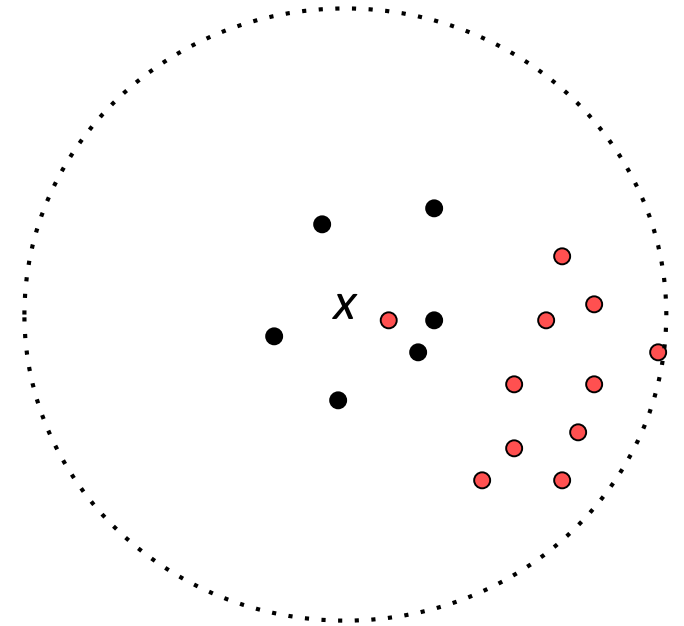
# Definition of nearest neighbors: distance function

- The  $k$ -nearest neighbors are selected among instances of the training set  $D$
- “Closeness” is defined in terms of a distance/similarity measure
- For example, Euclidean distance (or  $L_2$  norm)

$$\text{dist}(x, y) = \sqrt{\sum_{i=1}^d (p_i - q_i)^2}$$

# Nearest neighbor classification: voting in the neighborhood

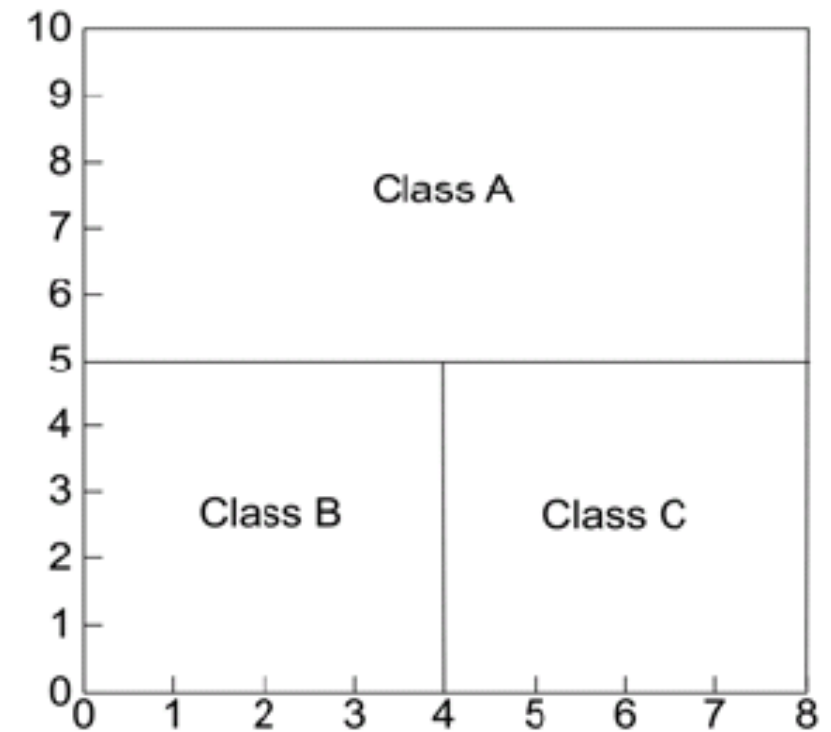
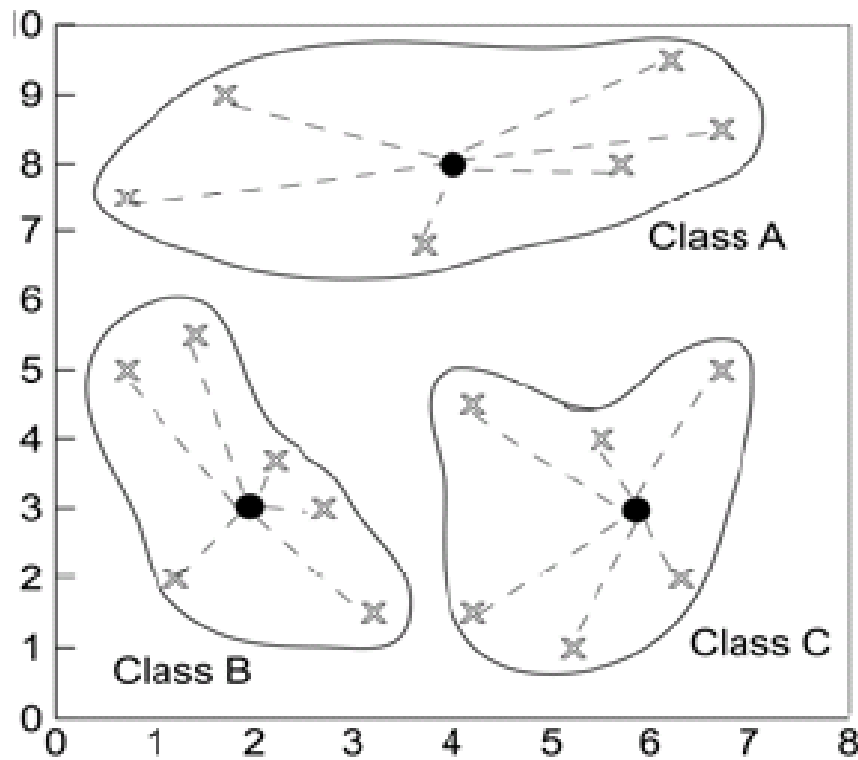
- The class of an unknown instance  $X$  is determined from its neighbors
- If  $k=1$ , the class is that of the closest instance
- In the general case ( $k>1$ )
  - **Majority voting**: take the majority vote of class labels among the neighbors
    - All neighbors contribute equally to the classification
    - The algorithm is very sensitive to the choice of  $k$
  - **Weighted voting**: each neighbor  $d$  contributes with a weight proportional to its distance from the unknown instance  $X$ 
    - e.g., a possible weight factor:  $w=1/dist(X,d)^2$



Compare majority to weighted voting in the above figure. What do you expect?

# Decision boundaries

- Nearest-neighbor classifiers can produce arbitrarily shaped decision boundaries
  - in contrary to e.g. decision trees that result in axis parallel hyper rectangles

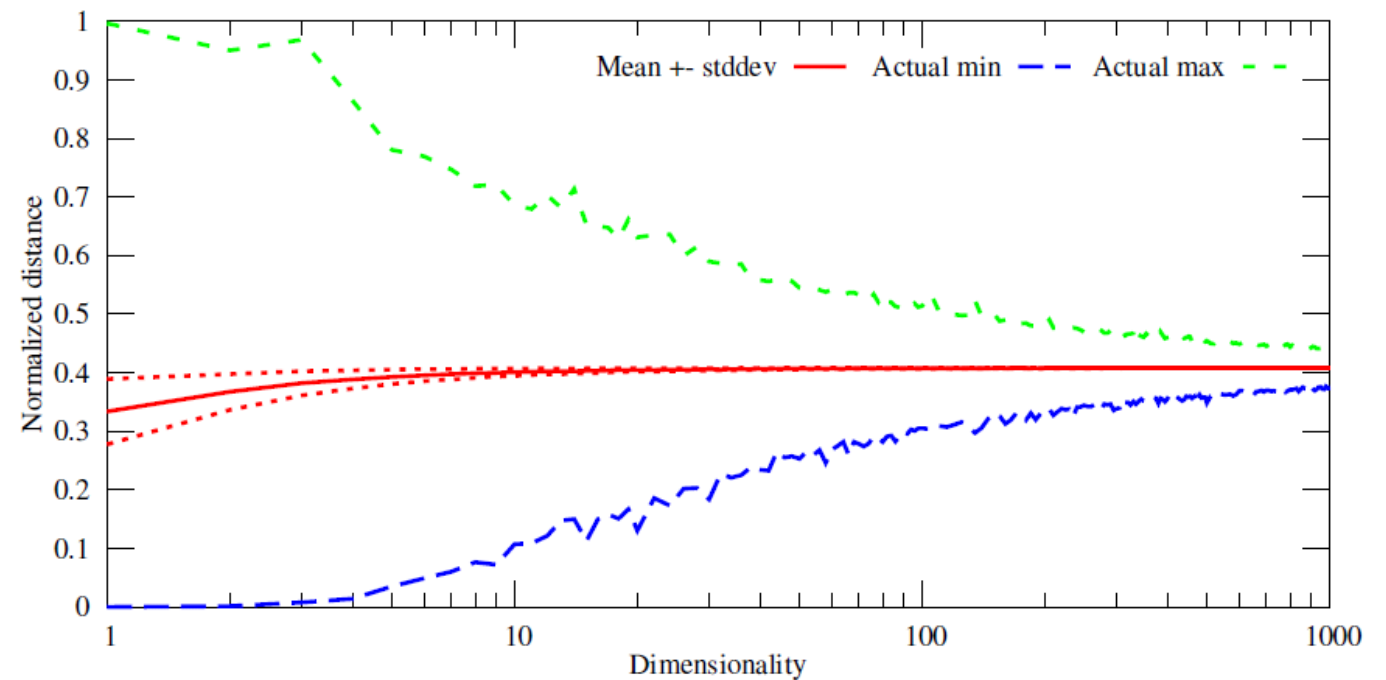


# Nearest neighbor classification issues 1

- Efficiency of kNN classifiers
  - Lazy learners: no model is built explicitly, like in eager learners
  - Classifying unknown records is relatively expensive
  - Possible solutions:
    - Use index structures to speed up the nearest neighbors computation
    - Partial distance computation (projected distances) based on a subset of attributes
- Different attributes have different value ranges
  - e.g., height in [1.5m-1.8m]; income in [\$10K-\$1M]
  - Distance measures might be dominated by one of the attributes
  - Solution: normalization

# Nearest neighbor classification issues 2

- Curse of dimensionality
  - (traditional) Distance functions that give same weight to all dimensions loose their discriminative power in high-dimensional spaces
  - Possible solutions:
    - Dimensionality reduction (e.g., PCA)
    - Work in a subspace



Source: Tutorial on Outlier Detection in High-Dimensional Data, Zimek et al, ICDM 2012



## k-NN classifiers: overview

- (+-) Lazy learners: Do not require building a model, but testing is more expensive
- (-) Classification is based on local information in contrast to e.g. DTs that try to find a global model that fits the entire input space: Susceptible to noise
- (+) Incremental classifiers → training set can be updated
- (-) The choice of distance function and parameter  $k$  is important
- (+) Nearest-neighbor classifiers can produce arbitrarily shaped decision boundaries, in contrary to e.g. decision trees that result in axis parallel hyper rectangles

# Overview and Reading w.r.t. KNNs

- Overview
  - KNNs (distance function, voting function, parameter  $k$ )
- Reading
  - Chapter 19: Nearest Neighbor, Understanding Machine Learning book by Shai Shalev-Schwartz and Shai Ben-David

Thank you

Questions/Feedback/Wishes?

# Acknowledgements

- The slides are based on
  - ❑ KDD I lecture at LMU Munich (Johannes Aßfalg, Christian Böhm, Karsten Borgwardt, Martin Ester, Eshref Januzaj, Karin Kailing, Peer Kröger, Eirini Ntoutsi, Jörg Sander, Matthias Schubert, Arthur Zimek, Andreas Züfle)
  - ❑ Introduction to Data Mining book slides at <http://www-users.cs.umn.edu/~kumar/dmbook/>
  - ❑ Pedro Domingos Machine Lecture course slides at the University of Washington
  - ❑ Machine Learning book by T. Mitchel slides at <http://www.cs.cmu.edu/~tom/mlbook-chapter-slides.html>
  - ❑ Thank you to all TAs contributing to their improvement, namely Vasileios Iosifidis, Damianos Melidis, Tai Le Quy, Han Tran.