Julius-Maximilians-UNIVERSITÄT WÜRZBURG CAIDAS WüNLP

ALGORITHMS IN AI & DATA SCIENCE 1 (AKIDS 1)

Non-Parametric Classification

Prof. Dr. Goran Glavaš

5.2.2024

Content

- Non-Parametric Models
  - Decision Trees
  - K-Nearest Neighbours

Supervised ML

- Two important dimensions of division in supervised ML

1. **Parametric** vs. **Non-parametric** models
2. **Generative** vs. **Discriminative** models

- Today we will see some **non-parametric** models
  - Decision Trees
  - K-Nearest Neighbours

Recap: Supervised ML

Three components of a supervised **machine learning algorithm**

1. Model: a set of functions among which we're looking for the best

$$H = \{ h(\mathbf{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}}$$

- **hypothesis** = a concrete function obtained for some values $\boldsymbol{\theta}$
- Model is a set of hypothesis

2. Loss function L: used to compute the **empirical error** E on a dataset $D = \{(\mathbf{x}, y)_i\}$

$$E(h|D) = \frac{1}{N}\sum_{i=1}^{N} L(\, h(\boldsymbol{x}_i|\boldsymbol{\theta}), y_i)$$

3. Optimization procedure: procedure or algorithm with which we find the hypothesis h^* from the model H that **minimizes** the empirical error

- Equivalent to finding parameters $\boldsymbol{\theta}^*$ that minimize E

$$h^* = \text{argmin}_{h \in H} E(h|D)$$

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} E(h|D)$$

Parametric vs. Non-Parametric

Model: a set of functions among which we're looking for the best

$$H = \{\, h(\mathbf{x} \mid \boldsymbol{\theta}) \}_{\boldsymbol{\theta}}$$

- Parameters $\boldsymbol{\theta}$ estimated using the annotated dataset $D = \{(\mathbf{x}, y)_i\}$

Parametric vs. non-parametric models

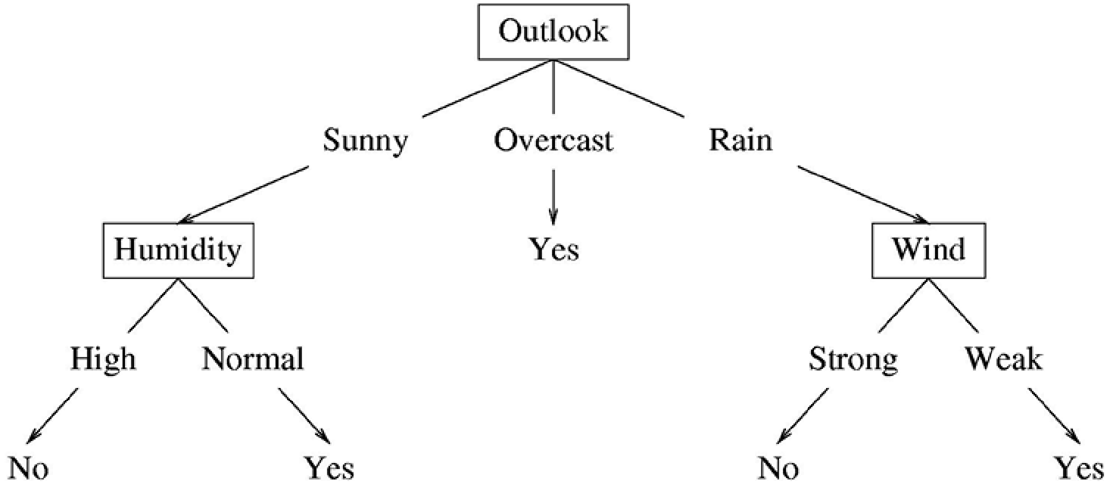
A model $H = \{\, h(\mathbf{x} \mid \boldsymbol{\theta}) \}_{\boldsymbol{\theta}}$ is **parametric** if its number of parameters n, $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_n]$ (estimated in model training) is **fixed** and **does not depend** on the size of the training dataset $D = \{(\mathbf{x}, y)_i\}$. Otherwise, the model is **non-parametric**.

Content

- Non-Parametric Models
  - Decision Trees
  - K-Nearest Neighbours

Decision Trees

- **Decision tree** refers to a **non-parametric** machine learning algorithm that builds a classifier as a **tree** of **if-then rules**
  - **Intermediate nodes**: features
  - **Leaf nodes**: classes



Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	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 Trees: Learning

- **Key questions**:
  (1) How do we select the features for the root and intermediate nodes?
  (2) When do we stop branching the tree?

- **Core algorithm**:

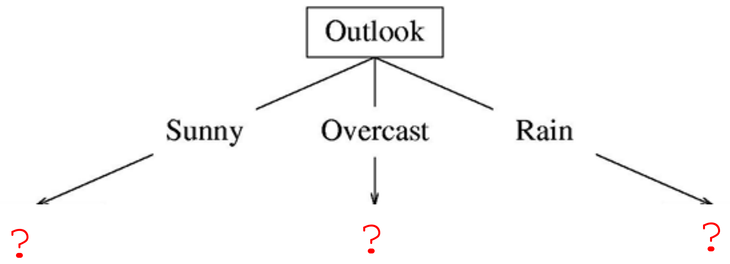
  Iterate the following steps:

  1. Select the „**best**" feature x_i for the next node
  2. Assign the selected feature x_i to the node
  3. For each value x_i create a new child node
  4. Filter „remaining" training examples for each new (child) node
  5. If the „remaining" examples for a node belong to a single class, **stop** further branching!

Decision Trees: Learning

- Let `select` be the function that picks the **„best”** feature
  - Based on some criteria we haven’t specified yet

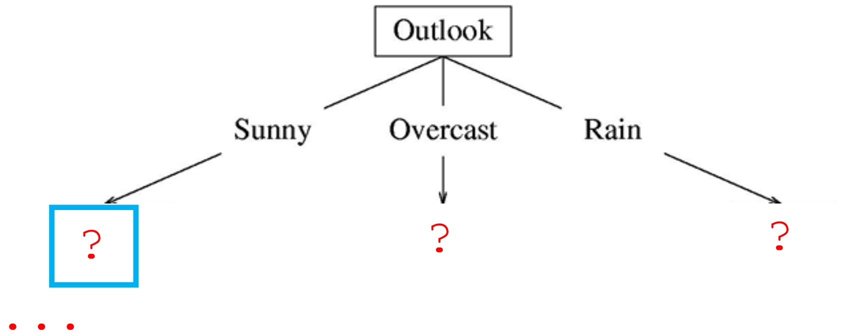
- Step #1:
  - `select` picks, e.g., **Outlook**
  - Since it’s a root node selection, **all instances** are considered for the selection criteria



Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	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 Trees: Learning

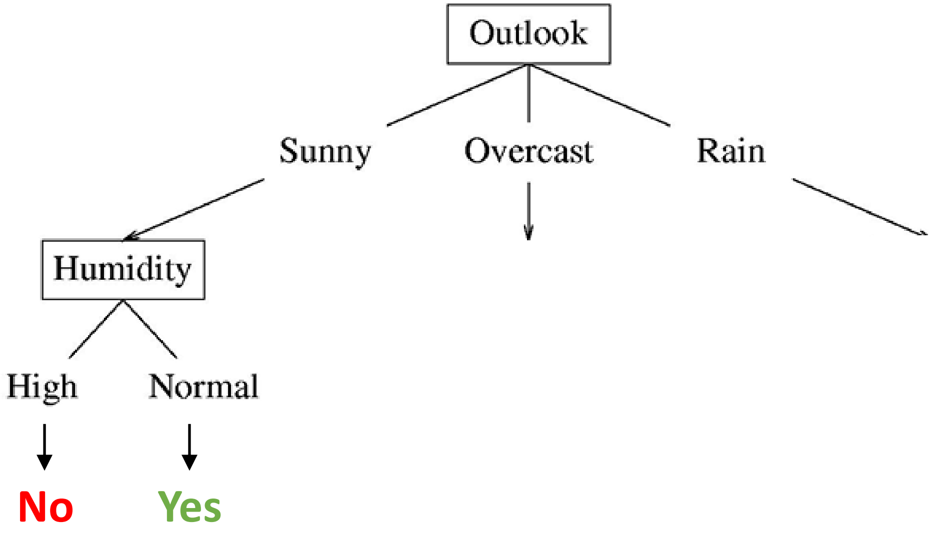
- Let `select` be the function that picks the „**best**” feature
  - Based on some criteria we haven’t specified yet
- Step #2:
  - `select` picks **Humidity**, by applying the selection criteria **only** over instances for which **Outlook** = sunny
  - And obviously chooses only between remaining features (Temp., Humid., Wind)
  - Remaining instances not all of the same label, fork the tree further



Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	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 Trees: Learning

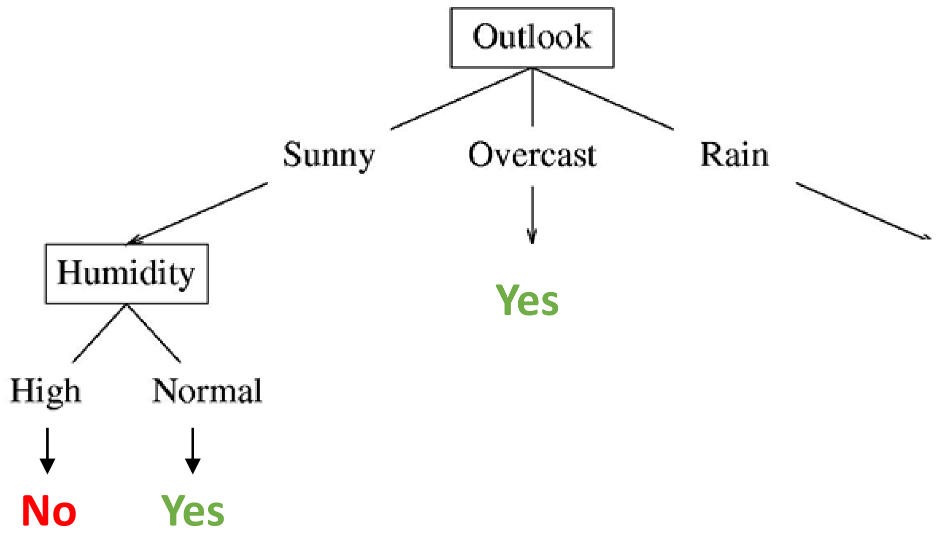
- Let `select` be the function that picks the „best" feature
  - Based on some criteria we haven't specified yet
- Steps #3 & #4:
  - All instances with **Outlook** = sunny and **Humidity** = high have the label **No**
  - All instances with **Outlook** = sunny and **Humidity** = high have the label **Yes**
  - **We stop for both branches!**



Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	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 Trees: Learning

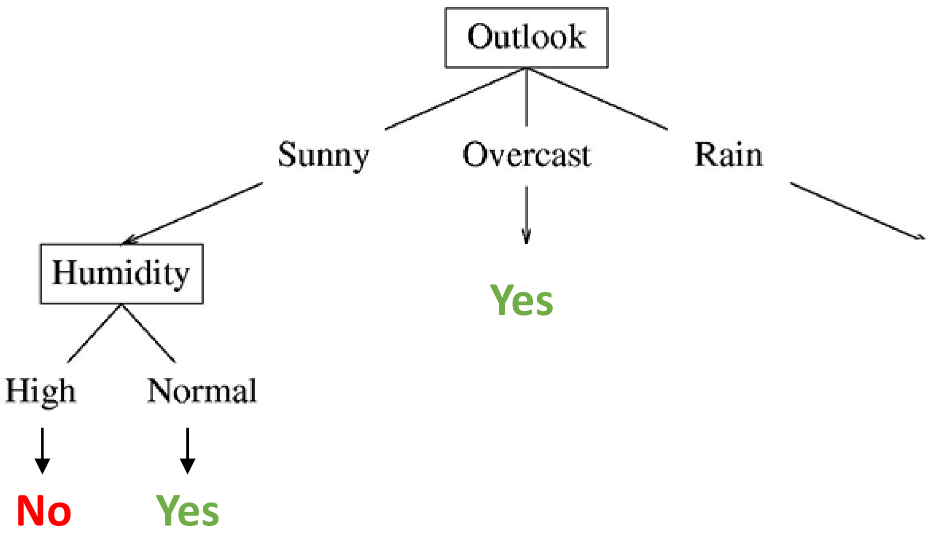
- Let `select` be the function that picks the „**best**” feature
  - Based on some criteria we haven't specified yet
- Step #5:
  - All instances with **Outlook** = overcast have the label Yes
  - **We stop for this branch!**



Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	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 Trees: Learning

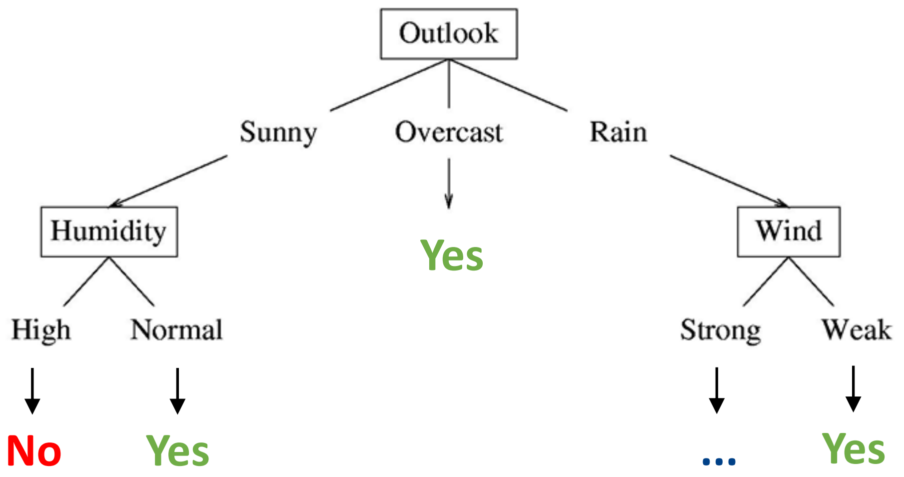
- Let `select` be the function that picks the „**best**" feature
  - Based on some criteria we haven't specified yet
- Step #6:
  - **Outlook** = Rain path (not all remaining instances have the same label)→ let's say `select` next chooses **Wind**



Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	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 Trees: Learning

- Let `select` be the function that picks the „**best**” feature
  - Based on some criteria we haven’t specified yet
- Step #7:
  - All instances for which **Outlook** = Rain and **Wind** = Weak have label Yes
  - Instances for which **Outlook** = Rain and **Wind** = Strong are not all of single label, we must continue that path...



Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	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 Trees: Learning

- We still need to define the **criteria** for choosing the feature for a node given a (sub)set of instances

- Some options:
  - **Random**: randomly pick one of the remaining features
  - **Least-Values**: choose the feature with the smallest number of values
  - **Most-Values**: choose the feature with the largest number of values

  - **Max-Gain**: choose the features that has the largest expected **information gain**
    - The one that reduces the **uncertainty (entropy :)** the most!

Entropy

- We will now introduce basic concepts from the **information theory**

- **Entropy** is a measure of **(un)certainty** or **„chaos”** (**order**)
  - Completely unpredictable things have **maximal entropy**
  - Completely predictable things have **minimal entropy**

- For a feature x_i for which the possible values are $\{v_1, v_2, \ldots, v_m\}$, with the corresponding probabilities $\{P(x_i = v_1), P(x_i = v_2), \ldots, P(x_i = v_m)\}$, entropy is defined as:

$$Entropy(x_i) = -\sum_{j=1}^{m} P(x_i = v_j) * \log_2 P(x_i = v_j) \quad \textit{bits}$$

- **Q:** When is entropy maximal?
- **Q:** When is it minimal (and how much is it)?

Entropy

- **Entropy** is a measure of **(un)certainty** or **„chaos”** (**order**)
  - Completely unpredictable things have **maximal entropy**
  - Completely predictable things have **minimal entropy**

$$Entropy(x_i) = -\sum_{j=1}^{m} P(x_i = v_j) * \log_2 P(x_i = v_j) \quad \textit{bits}$$

- $Entropy(x_i)$ = **0** (**minimal**) when some value v_j has **all of the probability mass** $P(x_i = v_j)$ = **1** and all other values have zero probability, $P(x_i = v_k)$ = **0**, $v_k \neq v_j$
  - **Certain** (completely predictable) that the feature x_i will have the value v_j
- $Entropy(x_i)$ = **$\log_2 m$** (**maximal**) when all values v_j have exactly the same probability, $P(x_i = v_j)$ = **1/m**
  - **Maximal uncertainty** (least predictable), as feature x_i is equally like to have any of the values

Information Gain

- **Information gain** a measure of **change** in entropy
  - For a node, where we need to assign one of remaining features, we measure:

1. Entropy of the node before assignment (taking into account all remaining training instances for the node)
2. (Weighted) average of entropies of children nodes to which we branch upon assignment of some feature to the parent node
3. Information gain, as the difference between 1) and 2)

- We then choose the feature for which the information gain, i.e., reduction in entropy is the **largest**

Information Gain

- *Entropy*(y|D): entropy of the label variable on a (sub)set of training instances D
- If D is the set of |D| instances covered by the parent node, and x_i with values $\{v_1, v_2, \ldots, v_m\}$ is the feature we consider to put into that parent node, then IG of x_i is:

$$IG(x_i|D) = Entropy(y|D) - \sum_{j=1}^{m} \frac{|D(x_i = v_j)|}{|D|} Entropy(y|D(x_i = v_j))$$

- $D(x_i = v_j)$ is the subset of instances in D for which the value of x_i is v_j

Information Gain: Example

- We choose the feature for the root node
- D = set of all instances, |D| = 14

  Entropy(y|D) = – 5/14*$\log_2$5/14 – 9/14*$\log_2$9/14 = **0.94**

- Let's first consider **Outlook**
  - If **Outlook** = sunny, $|D_{O=sunny}|$ = 5, y → 3 x **No**, 2 x **Yes**

    Entropy(y|$D_{O=sunny}$) = – 3/5*$\log_2$3/5 – 2/5*$\log_2$2/5 = **0.97**
  - If **Outlook** = overcast, $|D_{O=overc}|$ = 4, y → **0** x **No**, 4 x **Yes**

    Entropy(y|$D_{O=sunny}$) = – 0/4*$\log_2$0/4 – 4/4*$\log_2$4/4 = **0**
  - If **Outlook** = rain, $|D_{O=rain}|$ = 5, y → **2** x **No**, 3 x **Yes**

    Entropy(y|$D_{O=sunny}$) = – 2/5*$\log_2$2/5 – 3/5*$\log_2$3/5 = **0.97**

IG(**Outlook**|D) = 0.94 – (5/14 * **0.97** + 4/14***0** + 5/14 * **0.97**)
= 0.94 – 0.69
= **0.25**

Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	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 Trees

- **ID3** = decision tree algorithm that builds the tree based on information gain, proposed by Ross Quinlan
  - Slightly more advanced extension of ID3 is widely used C4.5
- It just recursively builds the tree by choosing features for nodes as shown
  - **Q**: guaranteed that this procedure finishes?
- **Pseudocode:** recursive function, each `node` gets
  - The remaining instances `D` (filtered by the path to that node)
  - List of remaining features `feats` between which to choose
    - A hashtable: feature name as *key* and list of feature values as *value*
  - parent node `p`
  - Value of feature of parent `v` that leads to the node

```
id3(D, feats, p, v)
  n = new node
  n.parent = p
  p.children[v] = n

  e = entropy(D)
  if e = 0
    n.content = get_class(D)
  else
    maxig = -inf
    best_f = null
    for f in feats:
      ig = inf_gain(e, f, D)
      if ig > maxig
        maxig = ig
        best_f = f
    n.content = best_f
    for val in feats[best_f]
      D_val = filter(D, best_f, val)
      # recursive call
      id3(D_val, feats/{best_f}, n, val)
```

Decision Trees: Numerical Features

- Our entropy and IG computation kind of assumed discrete features
  - **Q:** Can decision trees operate on numeric features?
- For numeric features, we need to „discretize" into two values „> t" and „< t" where t is some threshold value.
  - **Q:** but how do we choose t (infinite number of choices)?
- Some strategies for choosing the **discretization threshold** t:
  - **Mean** value of the feature on D
  - **Median** value of the feature on D
  - Search for „best" t (among a predefined set of candidate values): one that that gives the **largest Information Gain** for the feature

Decision trees: **overfitting**

- Following **IG** (or any similar measure) as a splitting criterion for building a tree can sometimes result in **overly specific trees**
  - Last-level non-leaf nodes cover as few as 1-2 instances from the training set
  - A splitting rule is less reliable the fewer instances it was made based on

  →Likely leads to a **tree** that **overfits** to the noise in the training set

- **Solution #1**:
  - **Search**: Build multiple trees (with different choices for features in nodes) and choose the simplest among them (**Q**: what is the „simplest"?)

- **Solution #2:**
  - **Pruning**: stop trees from becoming deeper than some fixed depth *d*

Decision Trees: **overfitting**

Occam's razor

A **philosophical principle** that favors **simplicity**: for any phenomenon, a simple explanation – one that introduces fewer assumptions – should be preferred over more complex ones – those that introduce more assumptions.

- The simplest „explanation" for our training data D is the tree with least rules: the **shallowest** of the possible decision trees
- **Search**:
  - If we have a small number of features, we can build many (all) possible trees (e.g., try all possible assignments of features to nodes)
  - **Choose:** (i) the **shallowest one** or (ii) one that performs best on our development/validation dataset

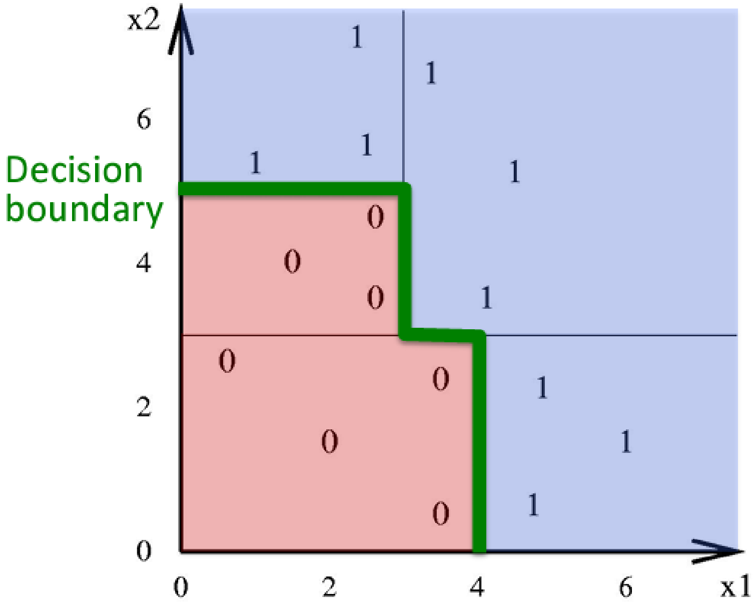
Decision Trees: **overfitting**

- If the number of features is large, search is not possible
  - **Q:** Why? If we have N features, what's the time complexity of building all possible trees (i.e., how many different trees are there)?

- **Pruning**: we simply stop building the tree after some maximal depth *d*
  - But the nodes at this maximal depth *d* may still have some Entropy > 0
    - In other words, remaining (filtered) instances corresponding to that node may not all belong to the same class (classification still ambiguous)
    - Yet, we **must make a classification decision** at this depth (as we're not allowed to build the tree further). **Q:** How then?
    - Prediction for nodes at *d*: **the most frequent class** among the remaining instances

Decision Trees: Decision Boundary

- **Q:** Is DT **discriminative** or **generative** ML model?
- How do decision trees actually divide the classes?
  - They divide the feature space into axis-parallel **hyper-rectangles**



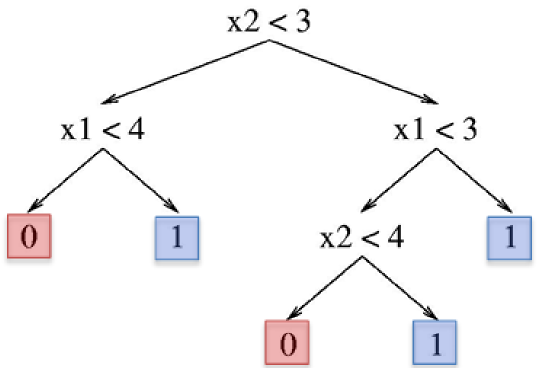


Image from: https://www.seas.upenn.edu/~cis5190/fall2017/lectures/02_DecisionTrees.pdf

Content

- Non-Parametric Models
  - Decision Trees
  - K-Nearest Neighbours

K-Nearest Neigbours

- **K-Nearest Neighbours** (K-NN) is a **non-parametric ML algorithm** that doesn't really have a training procedure, it merely classifies new instances based on the distance to (or similarity to) „training" examples

- Training instances (**x** = [x_1, x_2, ..., x_n], y) from D_{tr} are simply **stored in memory**

- We then **classify** new example **x'** as follows:

    1. We identify *k* examples from D_{tr} that are **most similar (closest)** to **x'**

    2. Among these *k* **nearest neighbours**, we identify the most freqent class label and assign that class to **x'**

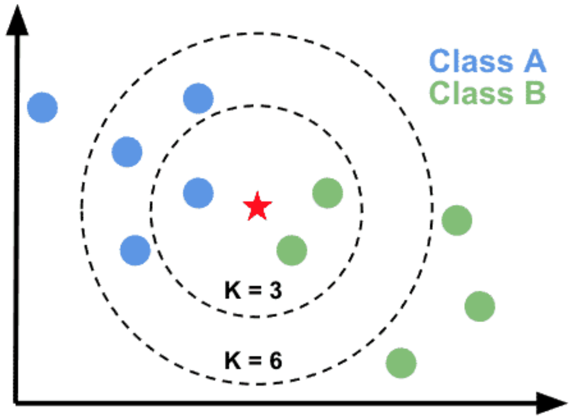


Image from: https://www.jcchouinard.com/k-nearest-neighbors/

K-Nearest Neigbours

- Obviously we need to specify two things:

1. The number of nearest neighbours *k* to consider
   - *k* is a hyperparameter, we find its optimal value with a validation dataset D_{val}
2. Measure of distance (or similarity) between two vectors
   - Many different distance/similarity measures, depending on the nature of the vectors/features (discrete, numeric, or a combination)

Euclidean distance

$$d_E(\mathbf{x}, \mathbf{x'}) = \sum_{i=1}^{n} \sqrt{(x_i - x_i')^2}$$

Cosine similarity

$$cos(\mathbf{x}, \mathbf{x'}) = \frac{\sum_{i=1}^{n} x_i * x_i'}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} x_i'^2}}$$

Questions?

Awọn ibeere?
Turite klausimų?
ਸਵਾਲ?
Küsimusi?
Pitanja?
Sorusu olan?
Dúvidas?
質問は？
有问题吗？
Fragen?
¿Preguntas?
Frågor?
Questions?
Pytania?
Domande?
Питання?
Vragen?
Ερωτήσεις;
Porandukuéra?
คำถาม?
أسئلة؟