

# The Curse of Dimensionality


State Reduction/Decomposition

Guillaume Lozenguez

[@imt-nord-europe.fr](mailto:@imt-nord-europe.fr)



**IMT Nord Europe**  
École Mines-Télécom  
IMT-Université de Lille

- 
1. **The Curse of Dimensionality**
  2. **Geometric Reduction**
  3. **State Decomposition**

# Before to go...

## Some Reminders on the Background here

- ▶ **Statistical Automata**
- ▶ **Dynamic Programming** (simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner)
  - Bellman and it's equation
- ▶ **Agent Based Modeling - Event-Driven Programming**

# 1. **The Curse of Dimensionality**

■ Example With 2 players 421

2. Geometric Reduction

3. State Decomposition

# System Difficulty

## Directly correlated to the state space:

**The number of states:** the Cartesian product of variable domains  $|S|$   
(minus some unreachable states)

▶ **421 game:** 3 dice-6 at the horizon 3:  $(3 \times 6^3 = 648)$  but 168  $(56 \times 3)$  effectives.

## Then the branching:

## Finally, the number of games:

# System Difficulty

Directly correlated to the state space

Then the branching:

The number of possible actions and actions' outcomes.

▶ **421 game:**  $2^3$  actions,  $6^r$  action outcomes ( $r$ , the number of rolled dice - *max. 3*).

Finally, the number of games:

The number of all game trajectories (possible succession of states) until an end is reached.

$|Branching|^h$  ( $h$  the horizon) - Potentially  $|S|^h$ .      **421 game:**  $(6^3)^3$  games

# Reminder over Combinatorics

**With a Classical 32-card game:** Possible distribution  $32! = 2.6 \times 10^{35}$



**Human life:** around  $5 \times 10^7$  seconds

Probability to play 2 times the same distribution in a human life is very close to 0

# Learning 2-players-421

State space ?

Branching ?

Estimation of the complexity and first ideas ?



# Learning in Combinatorics Context

## The root problem: handle large systems

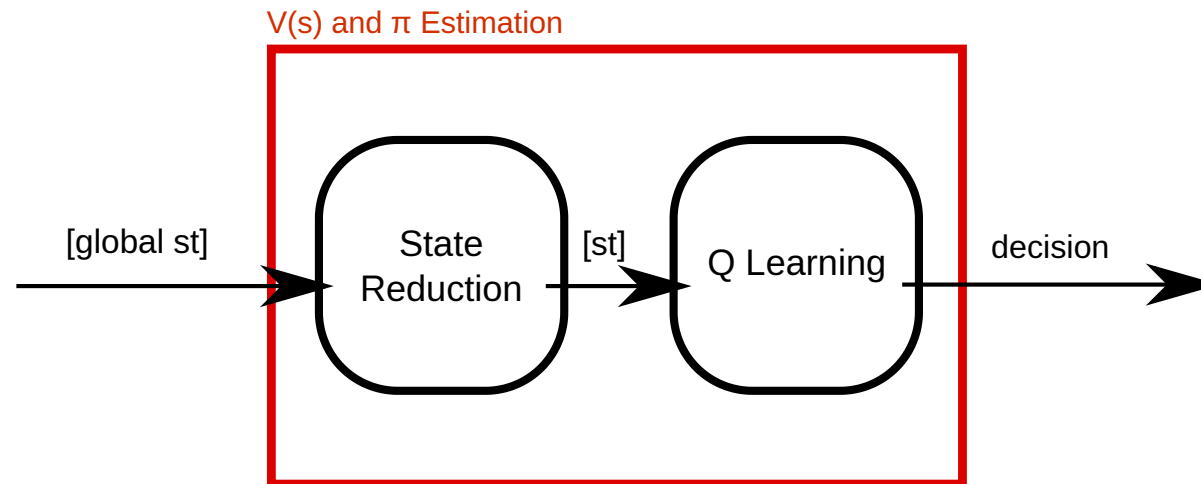
- ▶ Build a policy  $\pi : S \rightarrow A$
- ▶ Evaluate states  $V : S \rightarrow \mathbb{R}$  or  $Q : S \times A \rightarrow \mathbb{R}$
- ▶ Build a model  $T : S \times A \times S \rightarrow [0, 1]$

## A first basic solution:

- ▶ **Reduce the state space definition**

# State reduction in QLearning

Project the states in a smallest space (dimension and size)

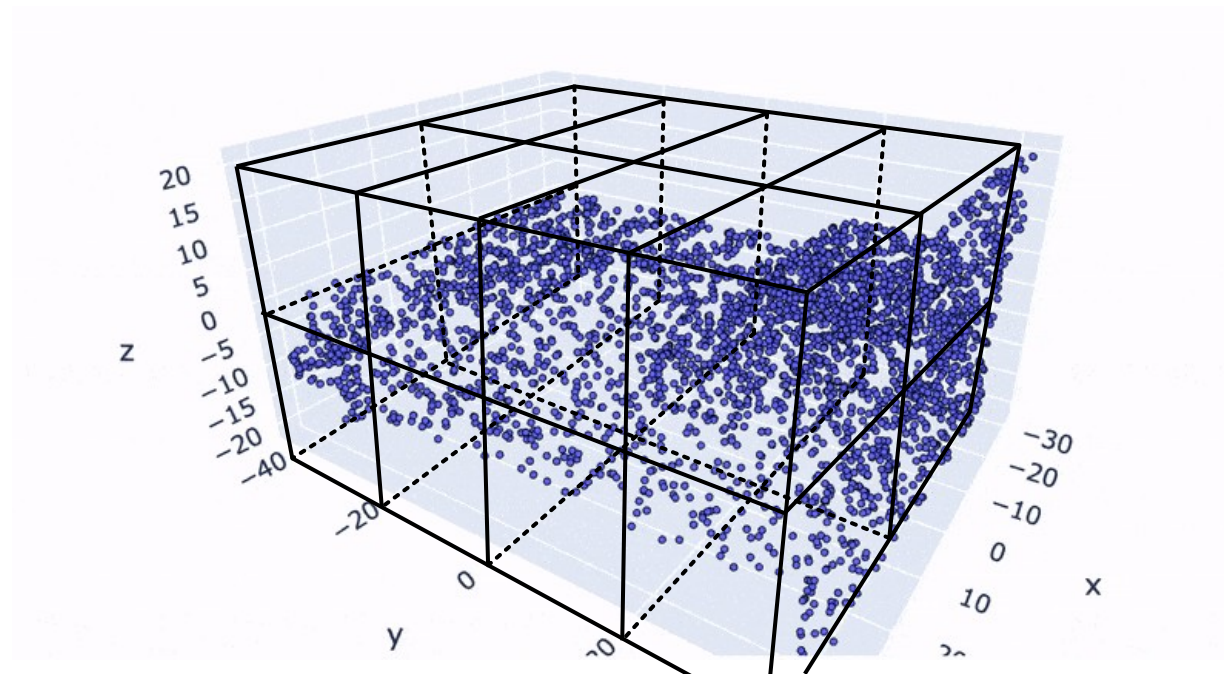


By mitigate the negative impact on the resulting built policy.

1. The Curse of Dimensionality
2. **Geometric Reduction**
  - Reduce the dimension (PCA)
  - Clustering (K-means)
3. State Decomposition
4. Quid of ...

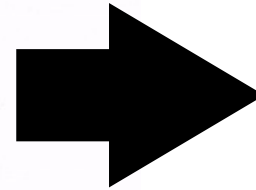
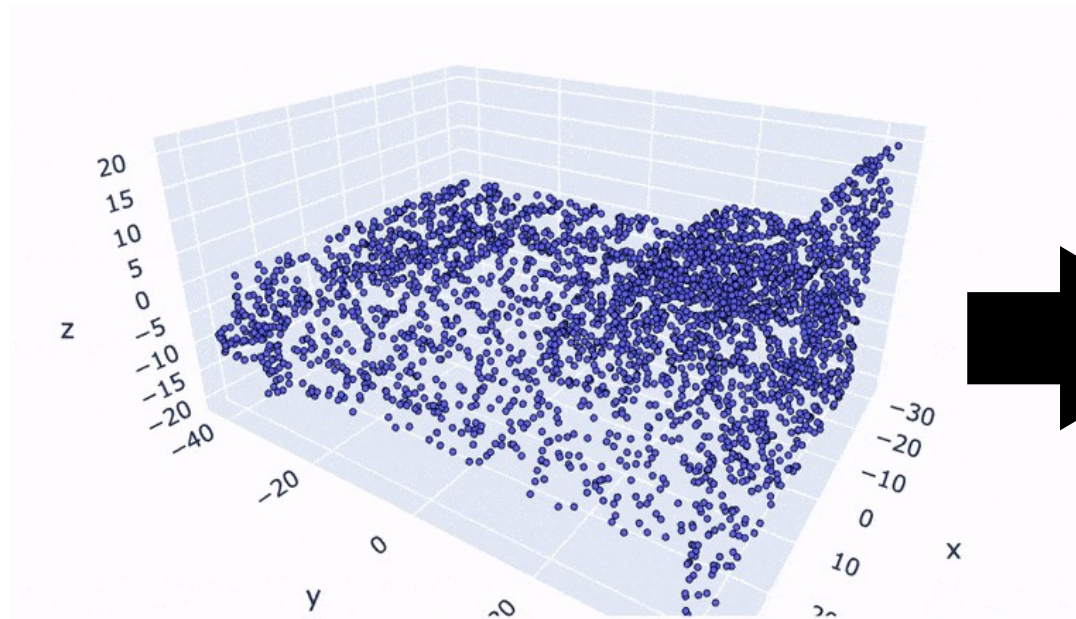
# Geometry Reduction

- ▶ Consider that **close** states are similar.
- ▶ Based on the assumption that: *it is possible to define a distance between States*
- ▶ By using regular discretization or adaptative clustering

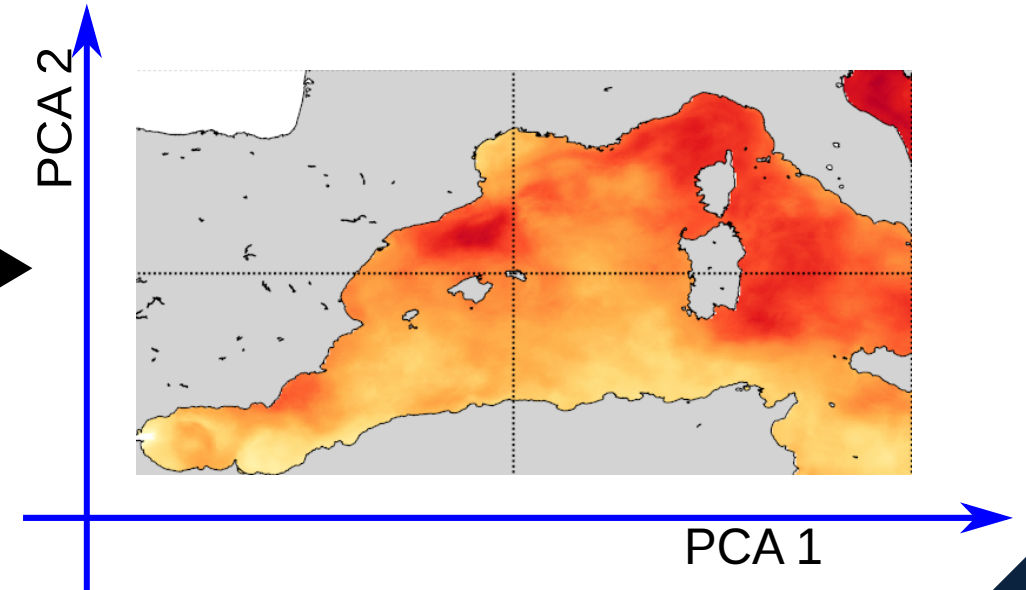


# Reduce the dimension - (Principal Component Analysis)

Searching the hyper-plan that better separate the data, in a given dimension.



State Space



# Reduce the dimension - (Principal Component Analysis)

## A Component $k$ :

- ▶ A vector of weight:  $w_{(k)} = (w_1, w_2, \dots, w_p)_{(k)}$  over the  $p$  dimensions
- ▶ That maximize the variance.

## Initialized with the first ( $k = 1$ ):

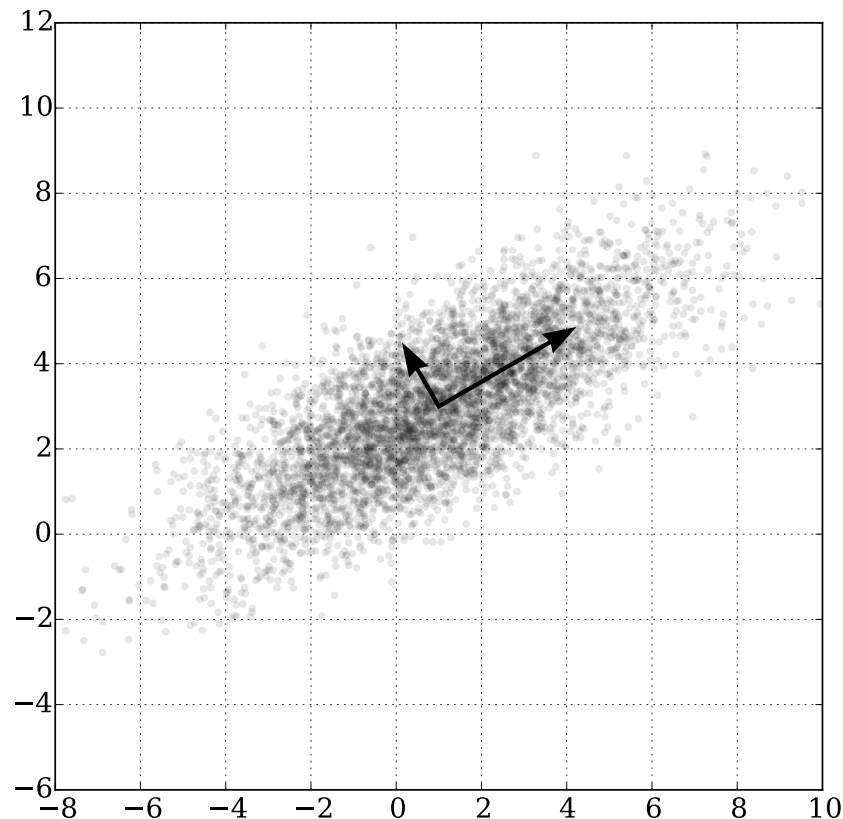
$$w_{(1)} = \arg \max_{||w||=1} \left( \sum_{x \in X} (x \cdot w)^2 \right)$$

## Then recursively:

Same equation but, the  $k$ -th component can be found by subtracting the first  $k - 1$  principal components from  $X$ .

# Reduce the dimension - (Principal Component Analysis)

## An example on generated values:



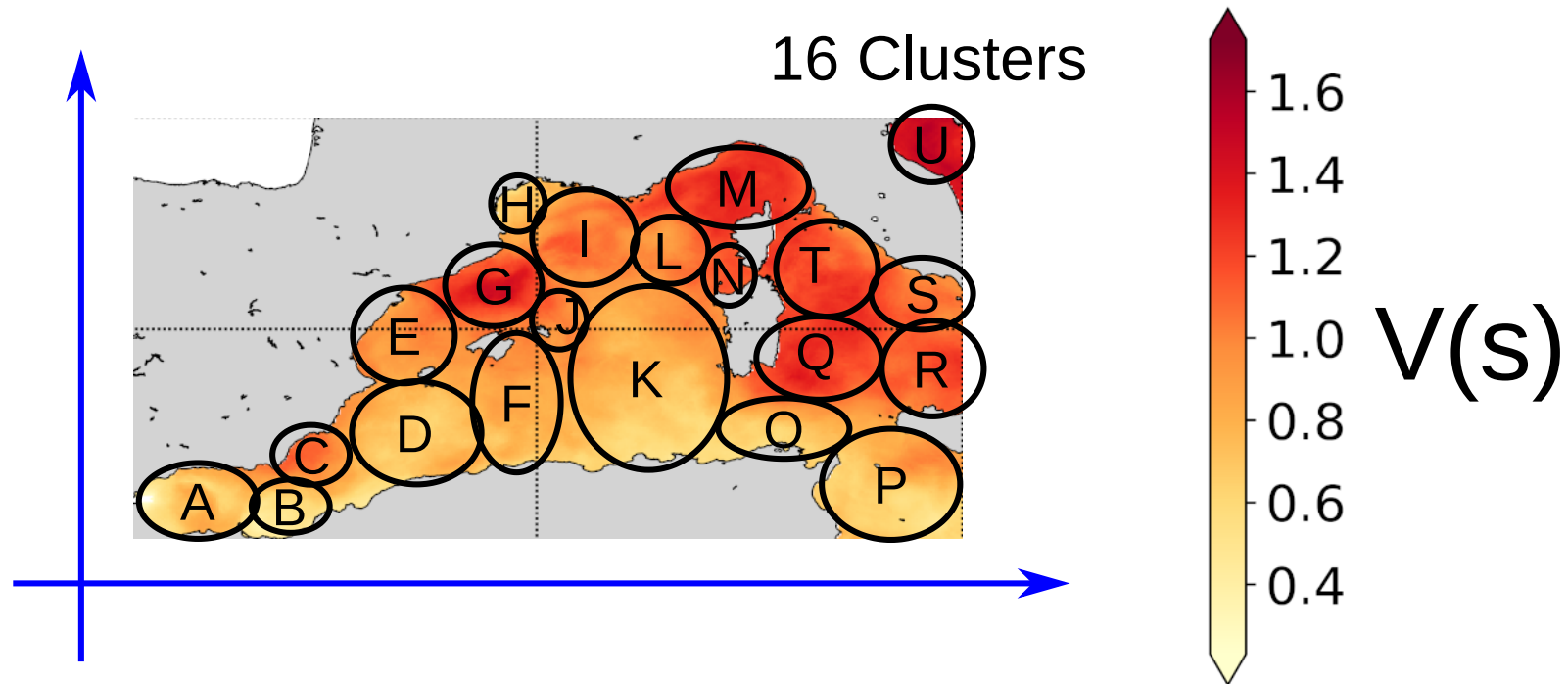
PCA of a multivariate Gaussian distribution centered at (1,3) with a standard deviation of 3 in roughly the (0.866, 0.5) direction and of 1 in the orthogonal direction.

The vectors shown are the eigenvectors of the covariance matrix scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean.

(Wikipedia)

# Clustering

Regroup the states in coherent sets



**Attention:** the action will be the same for all states in a same group.



# Clustering - (K-means)

## K-means:

Searching the optimal  $k$  center positions that better group/separate data

*Input:* Observations  $X$ ,  $k$  centers  $C = \{c_1, c_2 \dots c_k\}$

1. Repeat until: **stable**

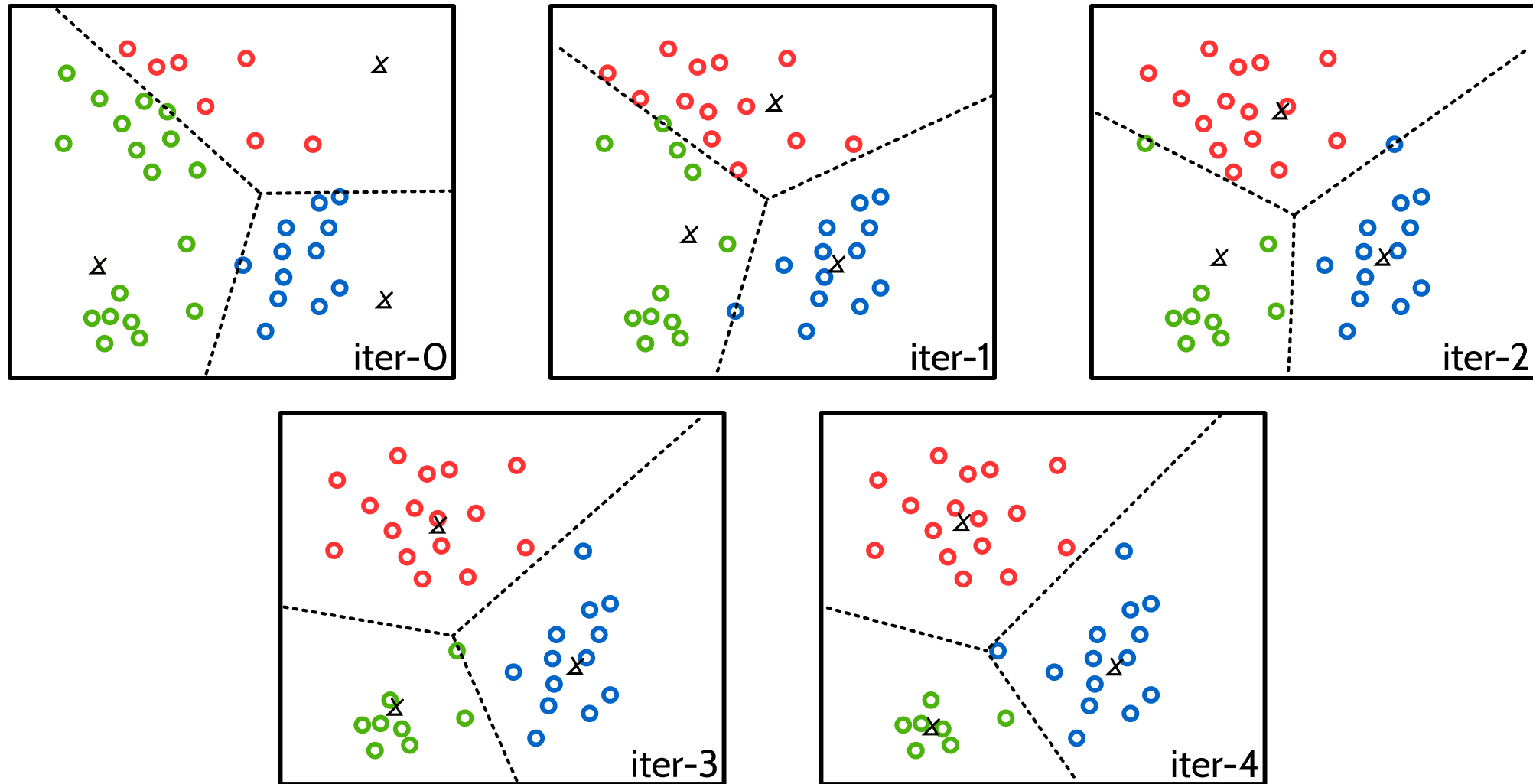
- Tags the observations  $tag(x) = \arg \min_{i \in [1 \dots k]} (distance(x, c_i))$
- Update the center  $c_i = average(X_i)$ ,  $X_i = \{x \in X, tag(x) = i\}$

*Output:* local optimal  $k$  centers.

## In real life:

Start k-means several times, with different random initialization.

# Clustering - Example on K-means



# Basic 'simple' classification method

## Principal Component Analysis (PCA)

Searching the hyper-plan that better separate the data, in a given dimension.

Python scikit-learn module: **sklearn.decomposition.PCA**

## K-means

Searching the optimal  $k$  center positions that better group the data together.

Python scikit-learn module: **sklearn.cluster.KMeans**

- ▶ Work well with 'linear state transitions' and variation in density.

# Geometric Reduction: Quid of ...

- ▶ The notion of distances
  - Several definitions: Euclidean, Manhattan, ...
  - Normalization.
  - Large dimension systems.
- ▶ The evaluation of the number of PCA, clusters, ...
  - precision vs noise
  - likelihood between a model and data: [Likelihood function](#)

1. The Curse of Dimensionality
  2. Geometric Reduction
  3. **State-Space Decomposition**
    - Decision Tree (on 421)
    - Extended Definition
- ▶ Learning Decision Tree
- Limits of the approach

# State-Space Decomposition

Factorized method: Based on state variable prevalence

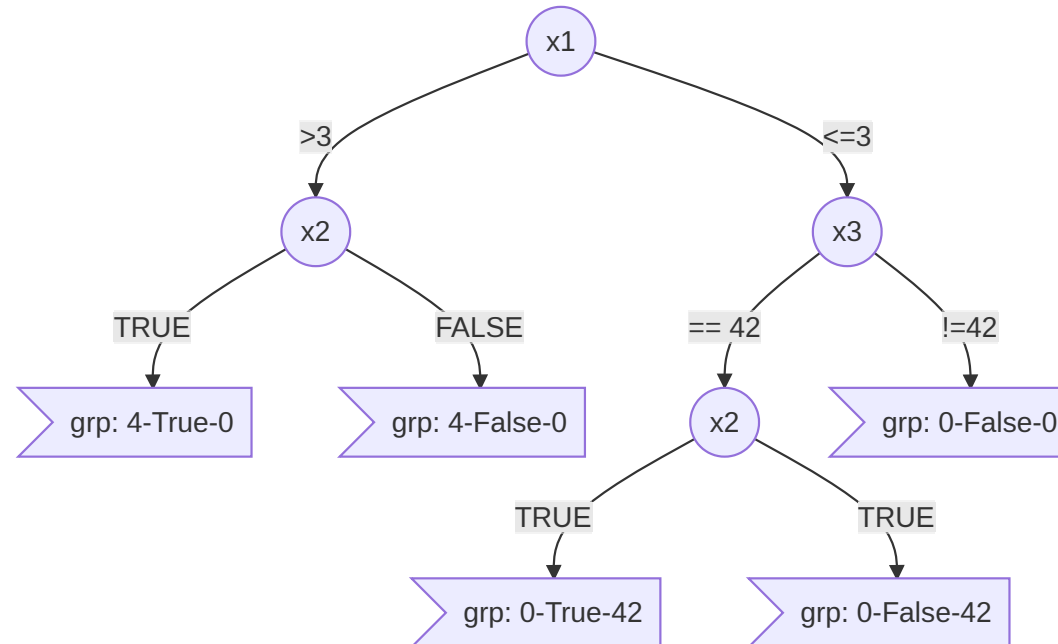
► *Decision tree*: **Nodes**: variables ; **Edges**: assignment ; **leaf**: group of states

Variable:

- x1 = [1, 2, 3, 4]
- x2 = [TRUE, FALSE]
- x3 = [1 - 50]

Groups:

- 4-True-X
- 4-False-X
- X-False-X
- X-True-42
- X-False-42

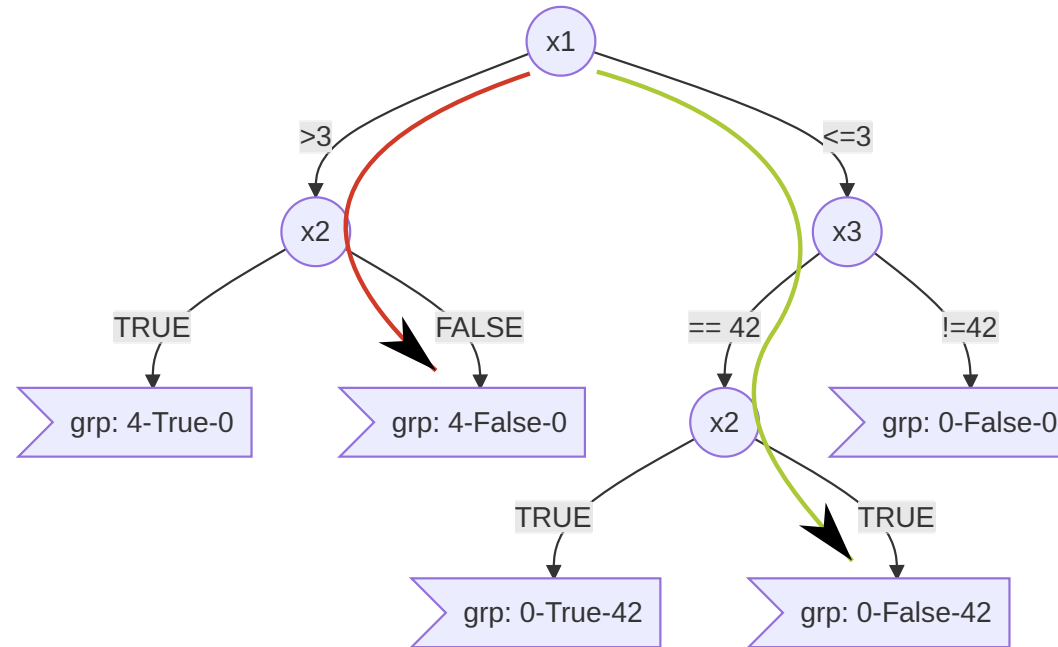


# State-Space Decomposition: A Tree

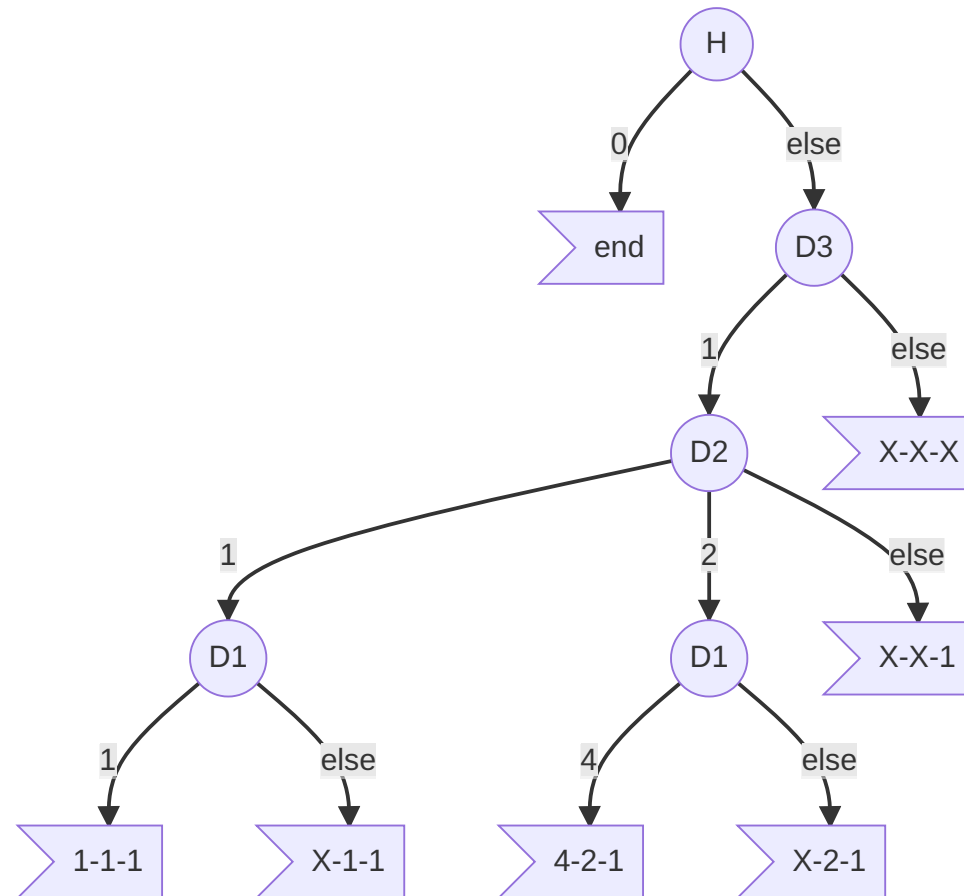
- ▶ Is a directed graph structure (with **Nodes** and **Oriented Edges**),
- ▶ Connected, with no loop and a unique path from any 2 nodes.

group( 4-**FALSE**-38 ) ?

group( 3-**TRUE**-42 ) ?



# Decision Tree On 421 Q-Learning





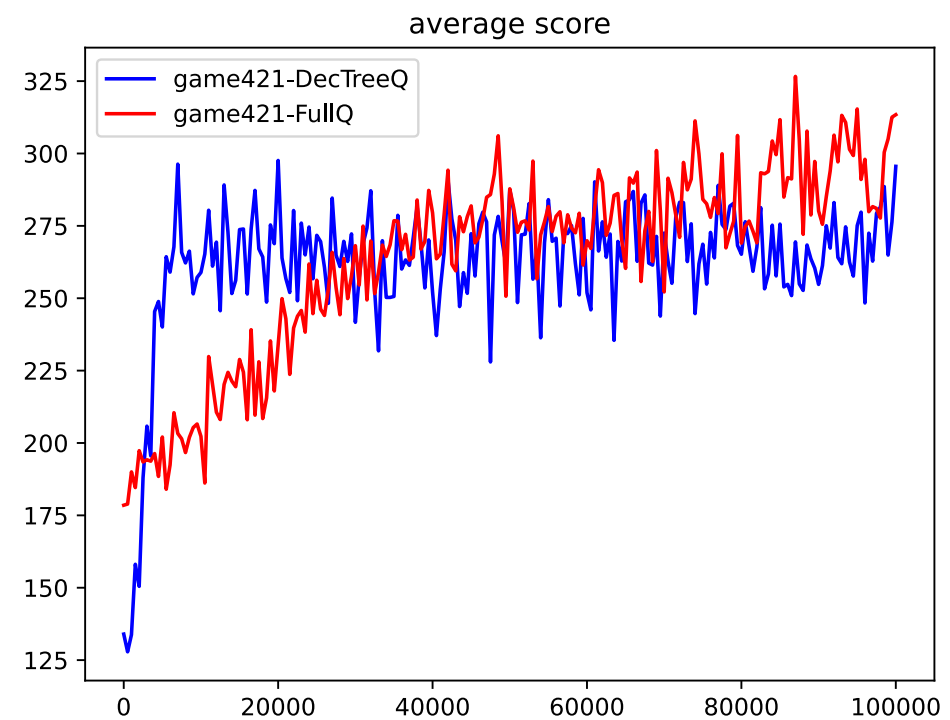
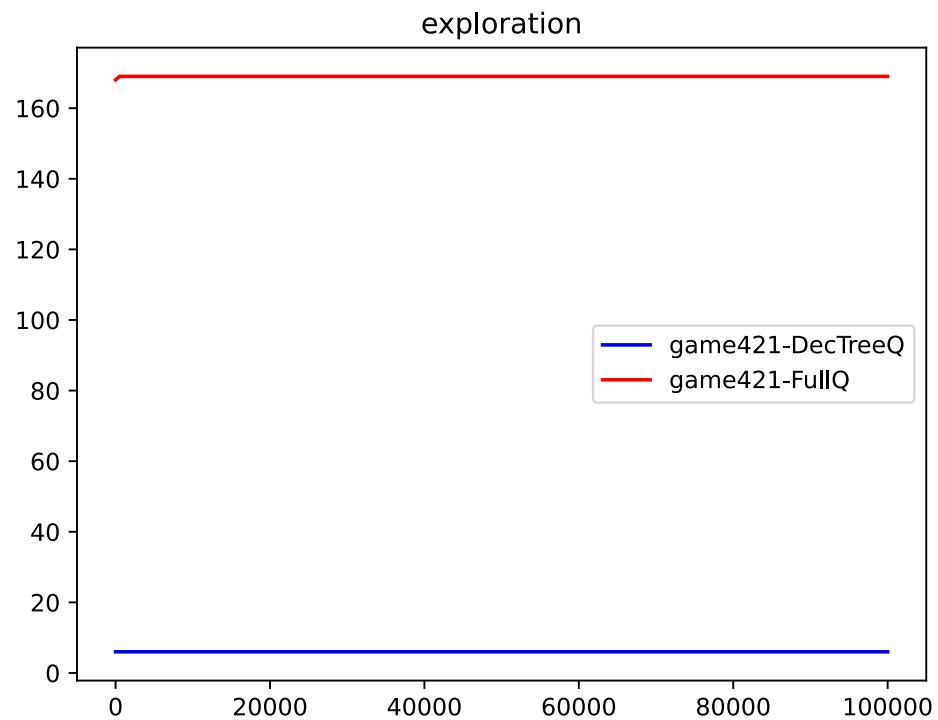
# Decision Tree On 421 Q-Learning

Simply reduce the state definition to 7 states...

```
def state(self):
    if self.turn == 0 :
        return 'end'
    if self.dices[2] == 1 :
        if self.dices[1] == 2 :
            if self.dices[0] == 4 :
                return "4-2-1"
            return "X-2-1"
        if self.dices[1] == 1 :
            return "X-1-1"
        return "X-X-1"
    return "X-X-X"
```

# Decision Tree On 421 Q-Learning

Results (Q-Learning over all the 168 states vs selected 7 states):



# Decision Tree: Extended Definition

## A tool for decision makers:

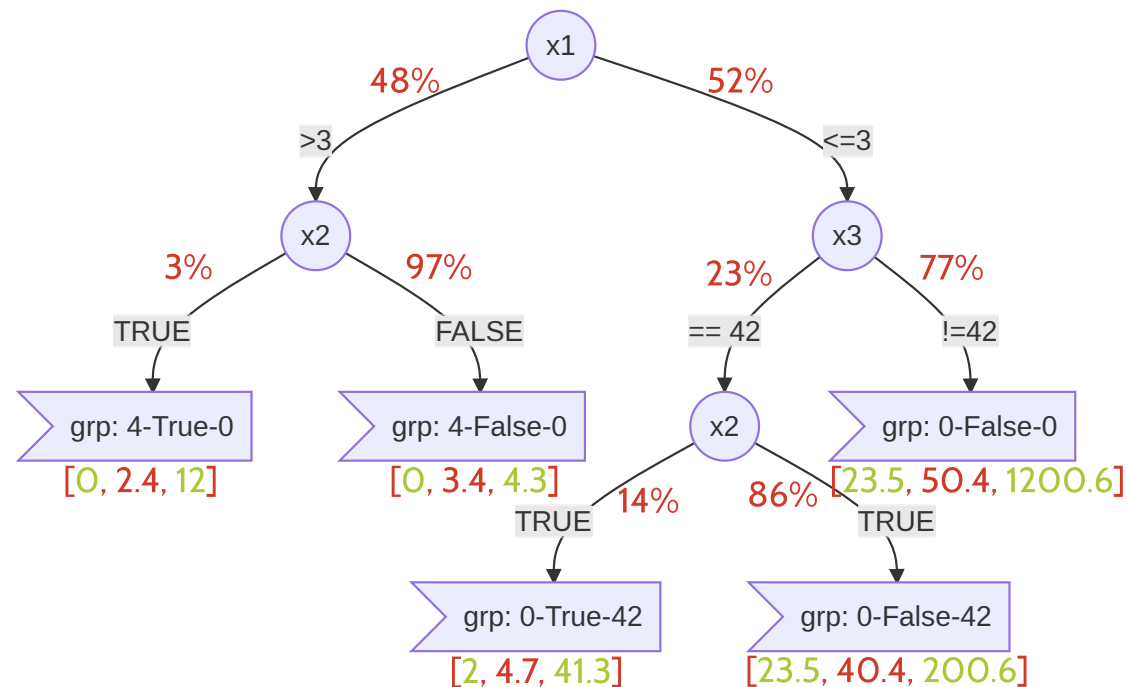
- ▶ A simple tool to depict decision rules historically draw by hand.
- ▶ As an opposition to black-box solution (deep learning for instance)

## An Analytic tool:

By integrating probabilities, cost, evaluations...

# Decision Tree: An Analytic tool:

- ▶ population proportion / experience evaluations



- ▶ **On 421:**  $H(X)$ -4-2-1 : weak proportion, very weak variation  
 $H(X)$ -X-X-1 : important proportion, huge variation

# Decision Tree: But require a tree...

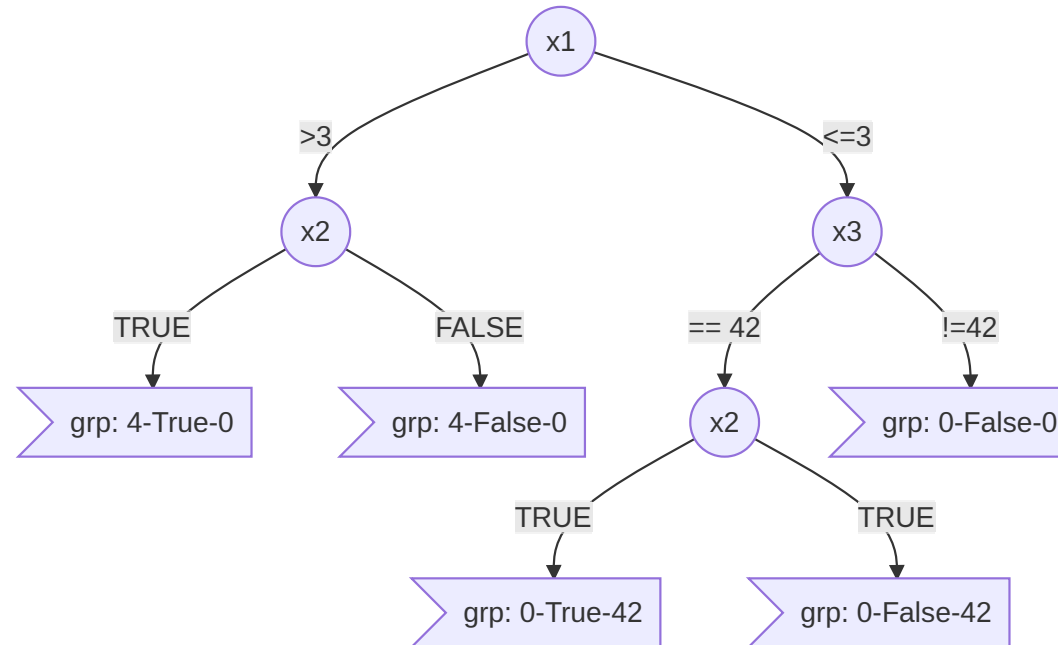
**Problem:** It is all about defining the appropriate variable prevalence

Variable:

- x1 = [1, 2, 3, 4]
- x2 = [TRUE, FALSE]
- x3 = [1 - 50]

Groups:

- 4-True-X
- 4-False-X
- X-False-X
- X-True-42
- X-False-42



**What-if:** X3 variable would be visited first ?

# ID3 Algorithm (Iterative Dichotomiser 3)

*Input:* Examples  $X$ , Attributes  $A$ , Labels  $[+, -]$  | *Output:* Decision Tree (local optimal)

- ▶ **If** All Examples positive : **return**  $leaf(+)$   
**If** All Examples negative : **return**  $leaf(-)$   
**If**  $A$  is empty : **return**  $leaf(\cdot)$ ,  $\cdot$  most common  $+$  or  $-$  in Examples
- ▶ **Otherwise** :
  1. define  $tree(a)$ , with  $a$ , the attribute in  $A$  that best classifies examples
  2. for each possible value  $vi$  of  $a$ 
    - add  $branch(vi)$  to  $tree(a)$
    - **If**  $X(vi)$  is empty ( the subset of  $X$  with  $A = vi$ ):  
 $branch(vi) = leaf(\cdot)$
    - **Otherwise** :  $branch(vi) = ID3(X(vi), A \setminus a)$
  3. return  $tree(a)$

# ID3 Algorithm - best classifier

## Select the attribute with the maximal Information Gain (IG):

$IG(X, a)$  is the measure of the difference in entropy from before after the set  $X$  is split on an attribute  $a$ .

$$IG(X, a) = H(X) - \sum_{vi \in V(a)} H(X(vi))$$

## Entropy:

$H(X)$  is a measure of the amount of uncertainty in the data set  $X$ .  
Computed from the proportion of *positive/negative* examples.

# ID3 Algorithm - properties

- ▶ *Local optimal* - no possibility to remove one of the nodes.
- ▶ *Greedy strategy* - not guarantee an optimal solution.
  - The best *IG* over 2 or more attributes do not necessarily include the best single attribute
- ▶ *Can overfit* training data. - no guarantee on smaller possible trees.
- ▶ *Not* easily applicable to *continuous* data.
- ▶ *Require examples* - no easy inclusion into reinforcement processes.
- ▶ *no robustness to noise* - noise will generate branches and leaves.

**But...**

- ▶ A lot of variations exist.



# Random Forest

## Idea:

**Genetic approach to Decision Tree.**

- ▶ Confront efficiency of different trees.
- ▶ Remove non-efficient ones
- ▶ Generate new one by merging efficient trees

## Decision:

**A vote over all the trees.**

# Decision Tree: Conclusion

It is all about defining the appropriate variable prevalence (Decision Tree Structure)

## Appreciate solution

- ▶ Quite simple
- ▶ Good compromise between : Efficiency / Explicability

## But...

- ▶ Requires labialized observations (Examples).

The evaluation of the structure of the tree is performed by deadly execution of Q-Learning !

- ▶ Build/use Decision-Tree in a reinforcement process...

# Decision Tree: Conclusion

It is all about defining the appropriate variable prevalence (Decision Tree Structure)

## Tools:

- ▶ On scikit learn: [module tree](#), [random forest](#).

# State-space Reduction/decomposition: Conclusion

## Reduce states ( $S$ ) definition to reduce core decision structure:

- ▶ Build a policy  $\pi : S \rightarrow A$
- ▶ Evaluate states  $V : S \rightarrow \mathbb{R}$  or  $Q : S \times A \rightarrow \mathbb{R}$
- ▶ Build a model  $T : S \times A \times S \rightarrow [0, 1]$

## Can be applied over ( $A$ )

- ▶ Implies necessarily a lost over **fine** decision-making (*i.e. only best decision on average*)