# Introduction to Machine Learning

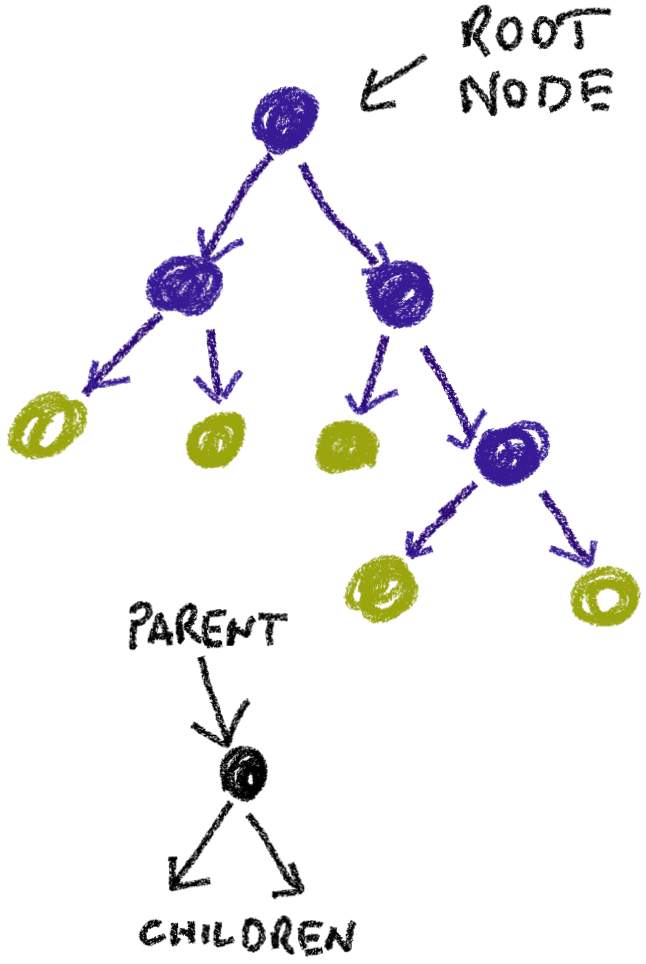
## Decision Trees & Random Forest & Feature Selection

Cigdem Beyan

UNIVERSITÀ DI TRENTO

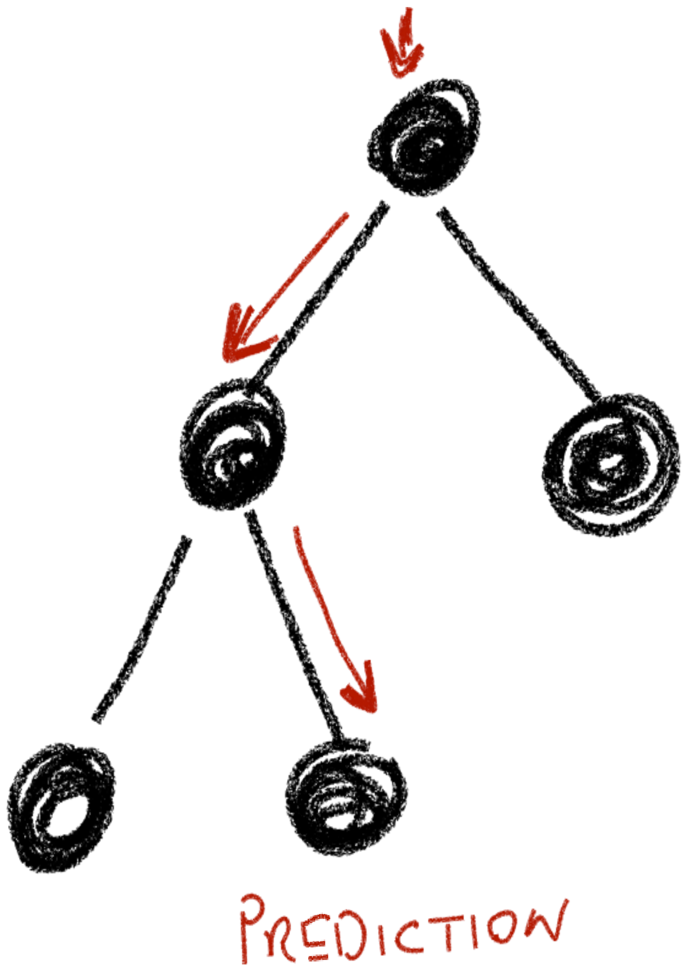
# Decision trees

- A tree-structured, prediction model.
- It is composed of terminal (or leaf) nodes and non-terminal nodes.
- **Non-terminal** nodes have 2+ children and implement a routing function.
- **Leaf nodes** have no children (i.e. terminal) and implement a prediction function.
- There are no cycles, all nodes have at most 1 parent excepting one a.k.a. **root node**



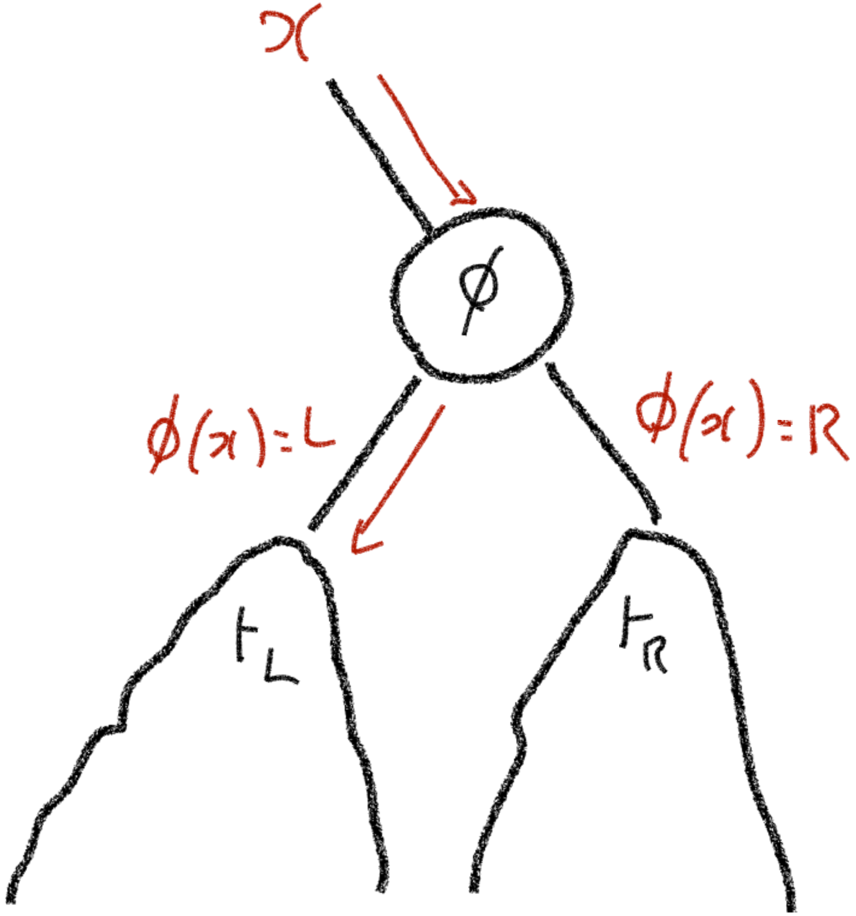
# Decision trees

- A decision tree takes an input $x \in \mathcal{X}$ and routes it through its nodes until it reaches a leaf node.
- In the leaf a prediction takes place



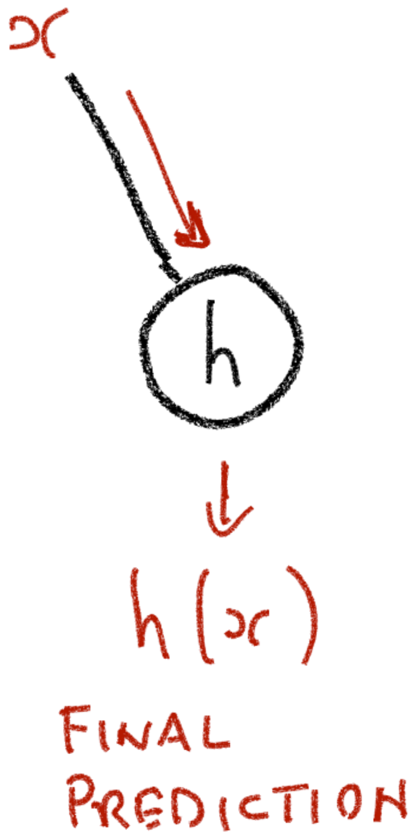
# Decision trees - inference

- Each non-terminal node $\mathtt{Node}(\phi, t_L, t_R)$ holds a **routing function** $\phi \in \{L, R\}^{\mathcal{X}}$ such that there exist a left child $t_L$ and right child $t_R$
- When $x$ reaches the node it will go to the left child $t_L$ or the right child $t_R$ depending on the value of $\phi(x) \in \{L, R\}$
- Here we assume a binary tree, thus there exist only 2 childs: left and right.



# Decision trees - inference

- Each leaf node $\mathtt{Leaf}(\mathtt{h})$ holds a **prediction** function $h \in \mathcal{F}_{\text{task}}$ (typically a constant)
- Depending on the task we want to solve it can be $h \in \mathcal{Y}^{\mathcal{X}}$, e.g., **classification** or **regression**.
- Once $x$ reaches the leaf the final prediction is given by $h(x)$.



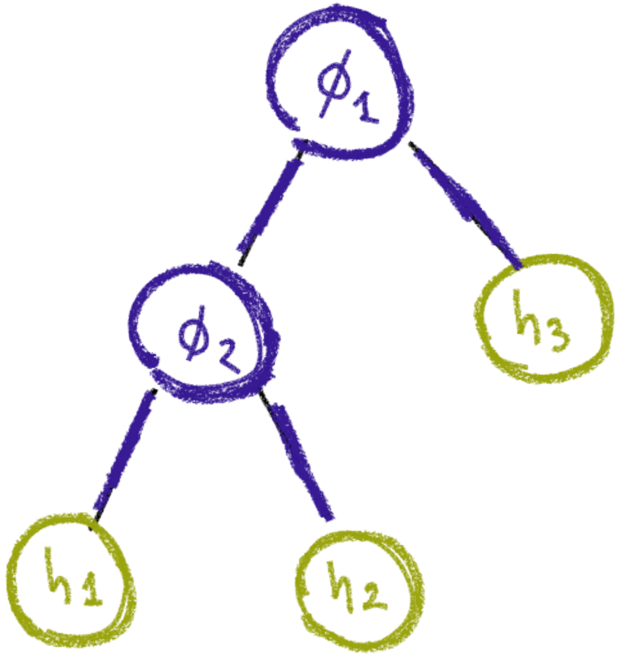
# Decision trees - inference

- The decision tree function:

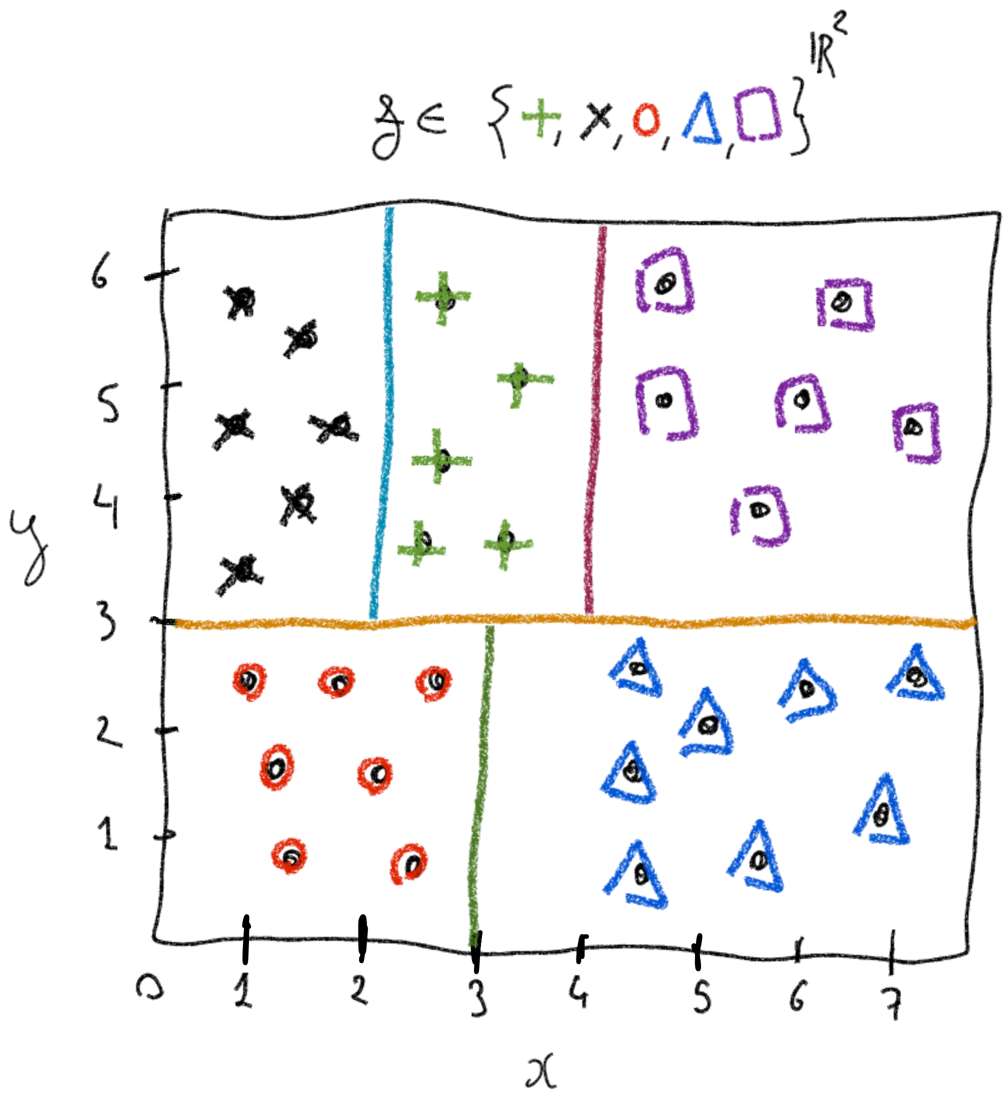
$$f_t(x) = \begin{cases} h(x) & \textbf{if } t = \texttt{Leaf}(h) \\ f_{t_{\phi(x)}}(x) & \textbf{if } t = \texttt{Node}(\phi, t_L, t_R) \end{cases}$$

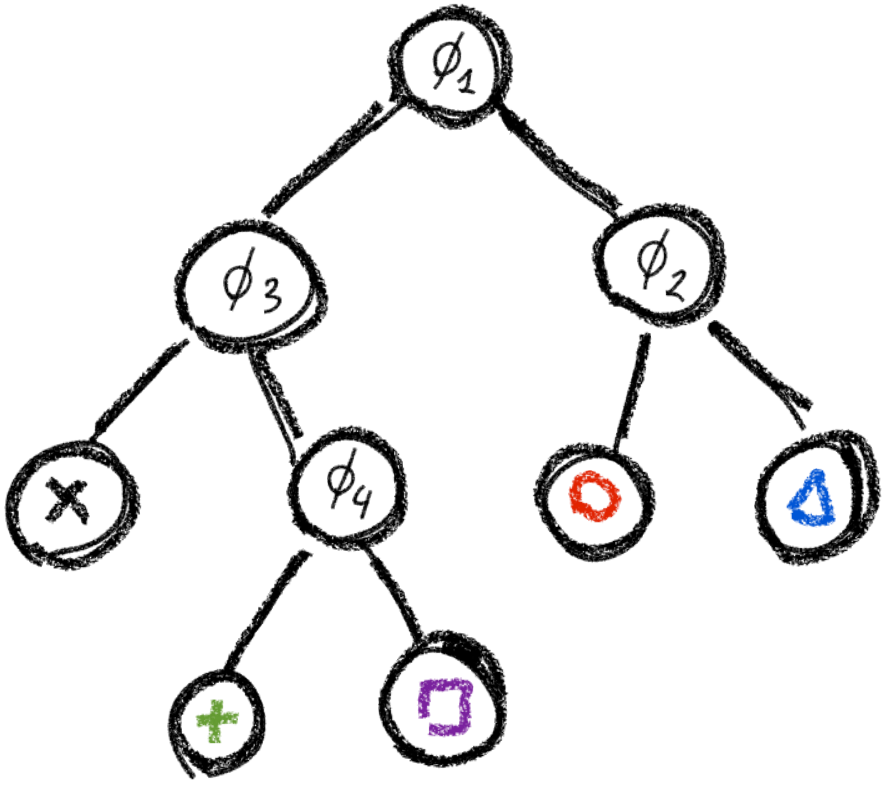
Where to go

Making prediction



# Decision trees- inference example




$$\phi_1(x, y) = \begin{cases} L & \textbf{if } y \geq 3 \\ R & \textit{else} \end{cases}$$

$$\phi_2(x, y) = \begin{cases} L & \textbf{if } x \leq 3 \\ R & \textit{else} \end{cases}$$

$$\phi_3(x, y) = \begin{cases} L & \textbf{if } x \leq 2 \\ R & \textit{else} \end{cases}$$

$$\phi_4(x, y) = \begin{cases} L & \textbf{if } x \leq 4 \\ R & \textit{else} \end{cases}$$

# Decision trees- learning algorithm

- Given a training set: $\mathscr{D}_n = \{z_1, \ldots, z_n\}$, we want to find a tree $t^\star$

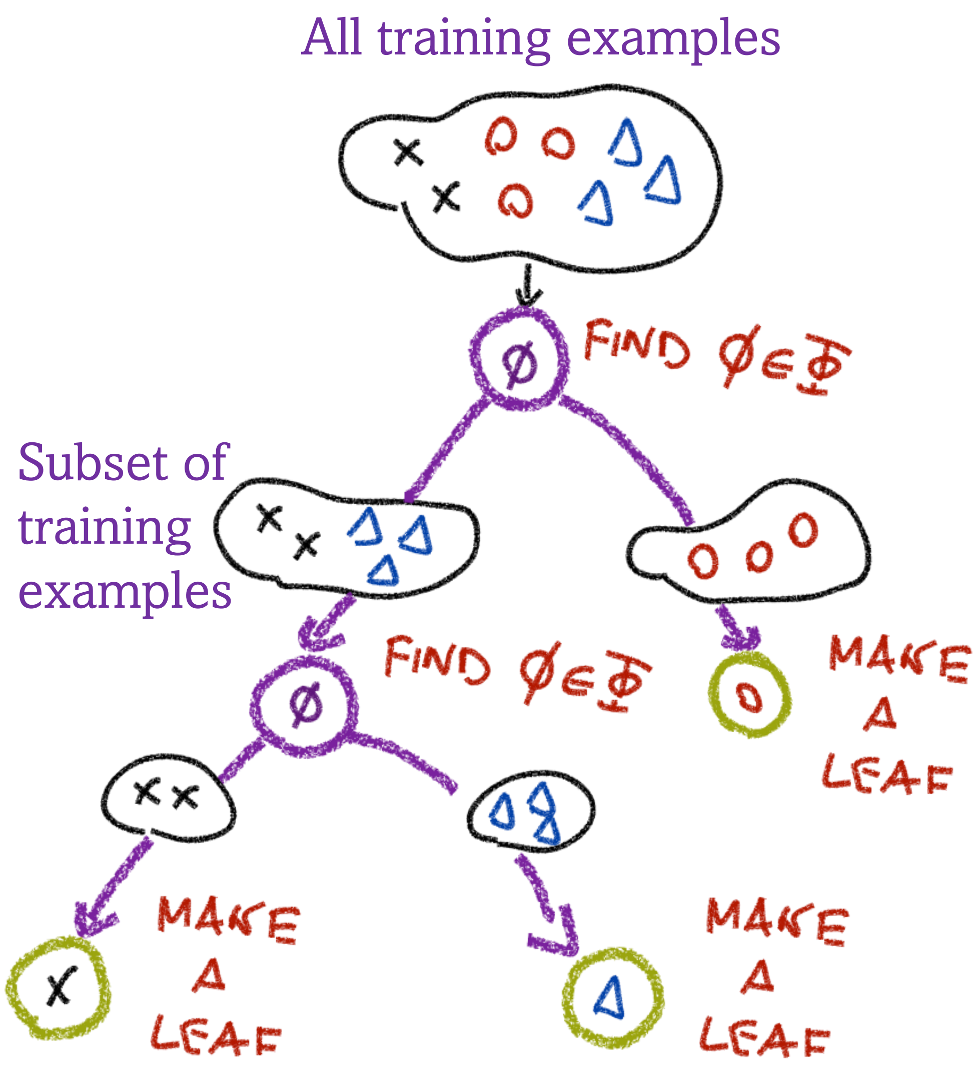
$$t^\star \in \underset{t \in \mathscr{T}}{\arg\min}\, E(f_t; \mathscr{D}_n)$$

set of decision trees

- This optimization problem has many solutions
  - Thus, we need to impose constraints, e.g., most compact tree, otherwise it could be NP-hard

# Decision trees- learning algorithm

- We need to fix a set of leaf predictions $\mathcal{H}_{\text{leaf}} \subset \mathcal{F}_{\text{task}}$ (e.g., constant functions)
- Fix a set of possible splitting functions $\Phi \subset \{L, R\}^{\mathcal{X}}$
- Tree-growing strategy recursively partition the training set and decides whether to grow the leaves or non-terminal nodes.
  - ID3 algorithm by Ross Quilan
  - CaRT by Breiman et al.



# ID3/CART ALGORITHM

Split (node, {examples} ):

Operates on each node in the tree. For each node you get a subset training examples that falls into that node

1. A ← the best attribute for splitting the {examples}

   features

   How to find the best? SOON!

2. Decision attribute for this node ← A
3. For each value of A, create new child node

   If A has 3 possible values, then there are 3 children

4. Split training {examples} to child nodes
5. For each child node / subset:

   if subset is pure: STOP

   else: Split (child_node, {subset} )

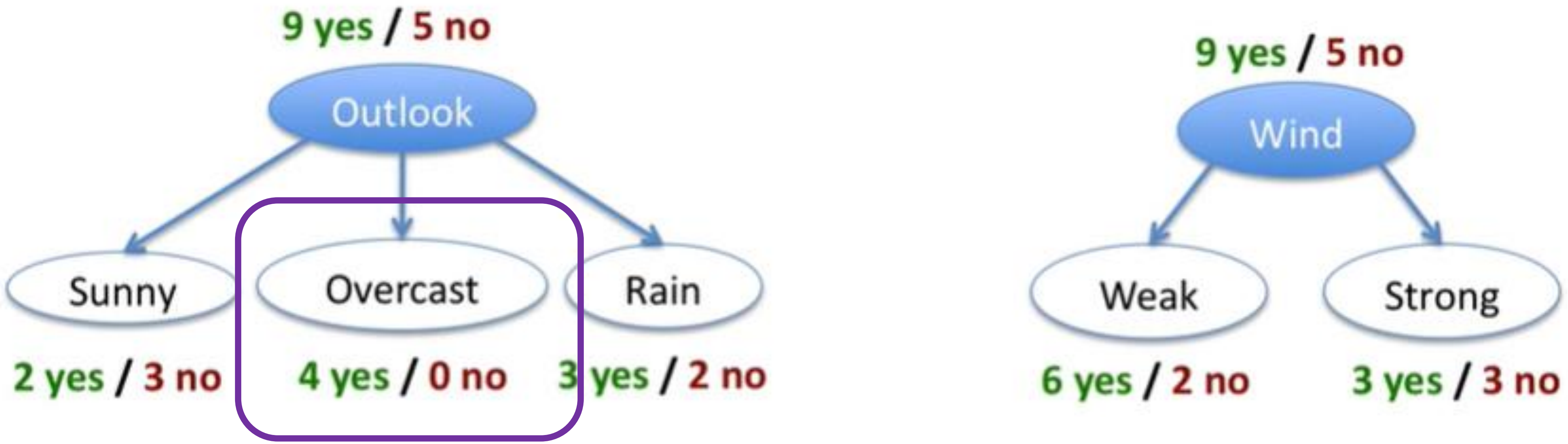
recursive

# Id3/CART algorithm- how do you pick the best attribute (feature)

| Day | Outlook | Humidity | Wind | Play |
|---|---|---|---|---|
| D1 | Sunny | High | Weak | No |
| D2 | Sunny | High | Strong | No |
| D3 | Overcast | High | Weak | Yes |
| D4 | Rain | High | Weak | Yes |
| D5 | Rain | Normal | Weak | Yes |
| D6 | Rain | Normal | Strong | No |
| D7 | Overcast | Normal | Strong | Yes |
| D8 | Sunny | High | Weak | No |
| D9 | Sunny | Normal | Weak | Yes |
| D10 | Rain | Normal | Weak | Yes |
| D11 | Sunny | Normal | Strong | Yes |
| D12 | Overcast | High | Strong | Yes |
| D13 | Overcast | Normal | Weak | Yes |
| D14 | Rain | High | Strong | No |

- 14 training samples
- 4 attributes /features
- 9 of them class: ”Yes”
- 5 of them class: “NO”

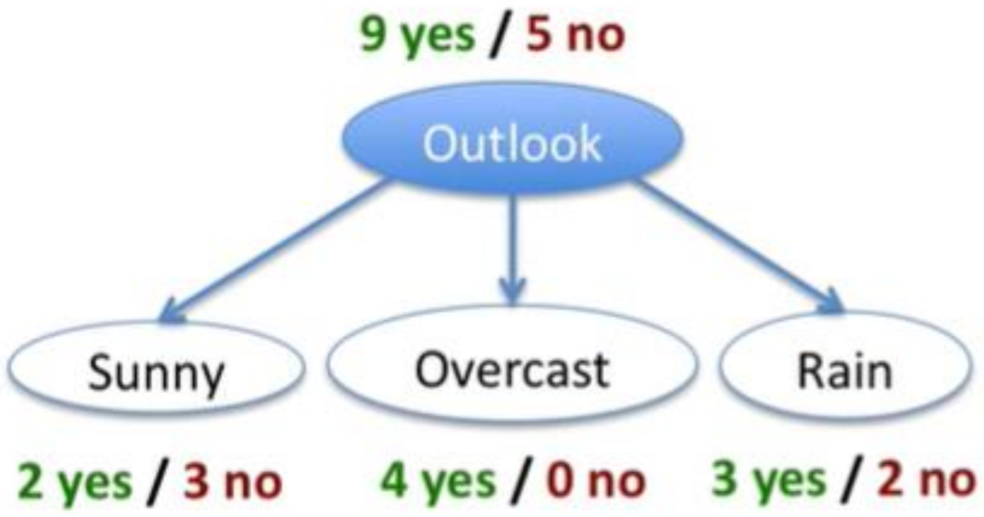
# ID3/CART algorithm- how do you pick the best attribute (feature)




- Which split do you find better and why?

- Outlook, because it has a pure subset

# Id3/CART algorithm- how do you pick the best attribute (feature)





- If you look at “wind”, which subset of the wind is better than other? Why?

- “Weak” is better, even though it is not pure.

# ID3/CART algorithm- how do you pick the best attribute (feature)

- We want to measure "purity" of the split.
  - We want to have more certain about Yes/No after the split.
  - Pure set (4 yes / 0 no) ➔ completely certain 100%
  - Impure set (3 yes / 3 no) ➔ completely uncertain 50%

- This measure should be symmetric!
  - 4 yes / 0 no is as pure as 0 yes / 4 no

- A lot of different ways to measure the purity (e.g., entropy)

# Entropy

- Assume that we have binary classes (positives and negatives)
- p(+) is the proportion of positives in a subset
- p(-) is the proportion of negatives in a subset
- Entropy of a subset ➔ $H(S) = - p_{(+)} \log_2 p_{(+)} - p_{(-)} \log_2 p_{(-)}$
- Impure (3 yes / 3 no): $H(S) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1$
- Pure (4 yes / 0 no): $H(S) = -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0$

*Higher numbers to the subset that are less pure!!!*

# Information gain

- We want to have many items in the pure sets.
- We calculate the expected drop in entropy after each split:

$$Gain(S,A) = H(S) - \sum_{V \in Values(A)} \frac{|S_V|}{|S|} H(S_V)$$

V: possible values of A
S: set of examples {X}
Sv: subset where $X_A$= V

$-\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14}$

H(S) = 0.94


$-\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8}$

$H(S_{weak}) = 0.81$

$-\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6}$

$H(S_{strong}) = 1.0$

Mutual information between A and class labels of S:

Gain (S, Wind)

$= H(S) - {}^{8}/_{14}\, H(S_{weak}) - {}^{6}/_{14}\, H(S_{weak})$

$= 0.94 - {}^{8}/_{14} * 0.81 - {}^{6}/_{14} * 1.0$

$= 0.049$

# Information gain

$-\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14}$

H(S) = 0.94


$-\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8}$

$H(S_{weak}) = 0.81$

$-\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6}$

$H(S_{strong}) = 1.0$

Mutual information between A and class labels of S:

Gain (S, Wind)

$= H(S) - {}^{8}/_{14}\, H(S_{weak}) - {}^{6}/_{14}\, H(S_{weak})$

$= 0.94 - {}^{8}/_{14} * 0.81 - {}^{6}/_{14} * 1.0$

$= 0.049$

We use information gain to decide which attribute to pick. We want to maximize the “information gain”.

# Information gain

- We use information gain to decide which attribute to pick.
  - Take every attribute that you have in your data
  - Compute gain for that attribute
  - Select the attribute that has the highest information gain.
  - Highest? That's the attribute which reduce the uncertainty the most, aka lead the purest possible split out of all attributes.
- We consider one level split at a time, but remember the procedure is recursive.

# Other measures?

Maximize the Gain

| | | | |
|---|---|---|---|
| Entropy | Classification | $\sum_{i=1}^{C} -f_i log(f_i)$ | $f_i$ is the frequency of label $i$ at a node and $C$ is the number of unique labels. |
| Gini impurity | Classification | $\sum_{i=1}^{C} f_i(1-f_i)$ | $f_i$ is the frequency of label $i$ at a node and $C$ is the number of unique labels. |
| Variance | Regression | $\frac{1}{N}\sum_{i=1}^{N}(y_i - \mu)^2$ | $y_i$ is label for an instance, $N$ is the number of instances and $\mu$ is the mean given by $\frac{1}{N}\sum_{i=1}^{N} y_i$. |

Minimize the variance

# Decision trees and overfitting

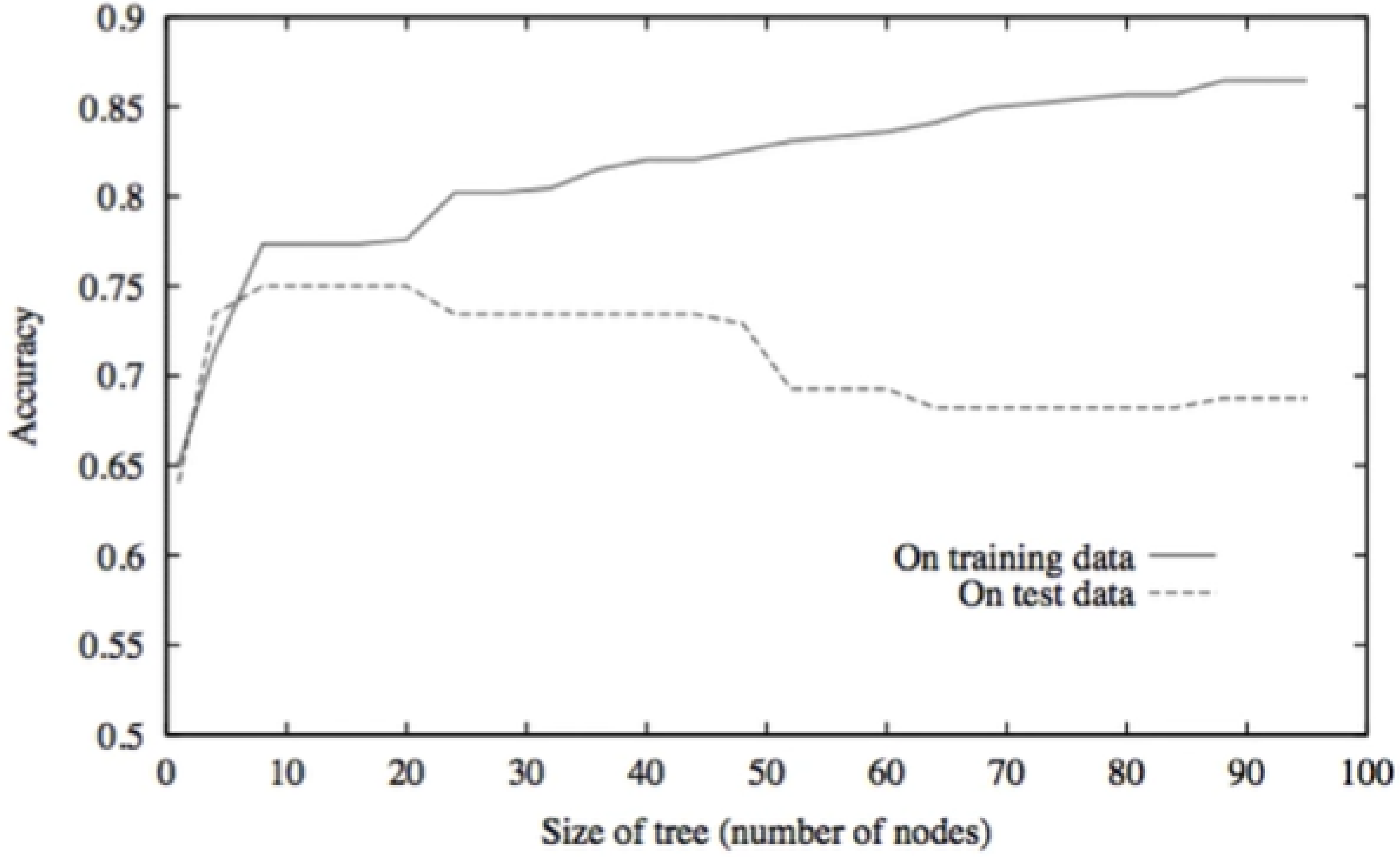
- Decision trees are **non-parametric models** with a structure that is determined by the data.

- As a result, they are flexible and can easily fit the training set, with high risk of **overfitting.**

- Standard techniques to improve generalization apply also to decision trees (early stopping, regularization, data augmentation, complexity reduction, ensembling).

- A technique to reduce complexity a posteriori is called **pruning.**

# Decision trees and overfitting

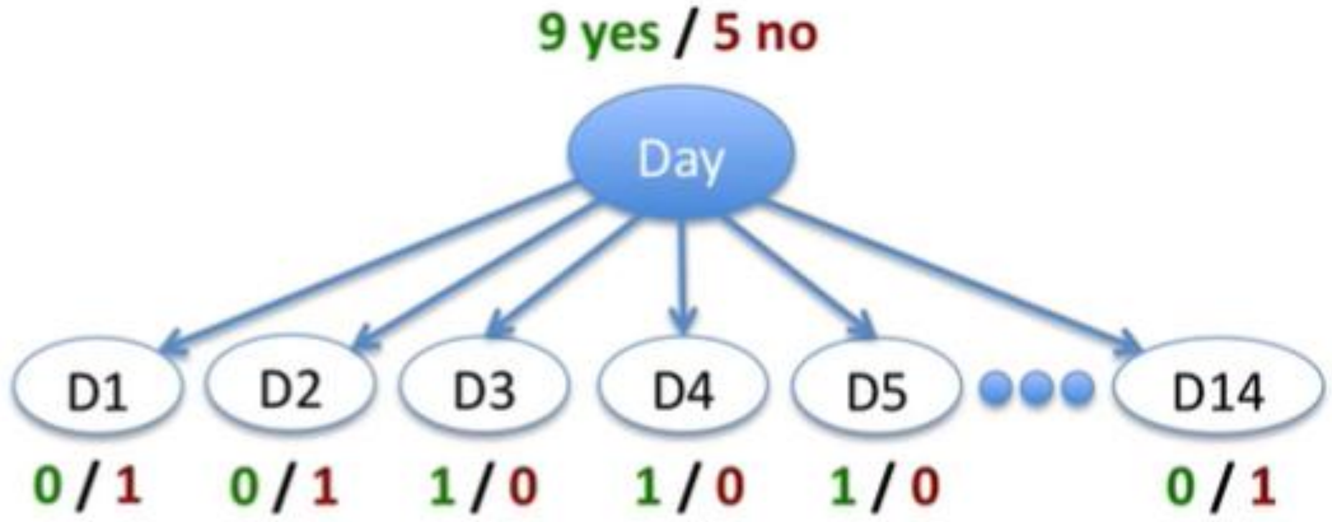



- Early stopping!
- Pruning!

Figure credit: Tom Mitchell, 1997

# Information Gain-- problems

| Day | Outlook | Humidity | Wind | Play |
|---|---|---|---|---|
| D1 | Sunny | High | Weak | No |
| D2 | Sunny | High | Strong | No |
| D3 | Overcast | High | Weak | Yes |
| D4 | Rain | High | Weak | Yes |
| D5 | Rain | Normal | Weak | Yes |
| D6 | Rain | Normal | Strong | No |
| D7 | Overcast | Normal | Strong | Yes |
| D8 | Sunny | High | Weak | No |
| D9 | Sunny | Normal | Weak | Yes |
| D10 | Rain | Normal | Weak | Yes |
| D11 | Sunny | Normal | Strong | Yes |
| D12 | Overcast | High | Strong | Yes |
| D13 | Overcast | Normal | Weak | Yes |
| D14 | Rain | High | Strong | No |




All subsets perfectly pure ➔optimal split

According to definition of *Information Gain*, "Day" is a perfect attribute. But….

**Generalize poor** in the testing data

Problem!!!!!

***Won't work for a new data!***

"D15 Rain High Weak"

# Information Gain- Cons.

According to definition of *Information Gain*, "Day" is a perfect attribute. But….

**Generalize poor** in the testing data

How to handle this? One possibility if using **GainRatio**

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

$$SplitEntropy(S,A) = -\sum_{V \in Values(A)} \frac{|S_V|}{|S|} \log \frac{|S_V|}{|S|}$$

A: candidate attribute
V: possible values of A
S: set of examples {X}
Sv: subset where $X_A$= V

**Penalize attributes with many values**

# Information Gain– pros.

- Decision trees are interpretable.

- It is possible to read the rules of the tree. There is concise description of what makes an item positive/negative.

- No "black box"
  - Important for users!

Rule: (Outlook = Overcast) ∨
(Outlook = Rain ∧ Wind = Weak) ∨
(Outlook = Sunny ∧ Humidity = Normal)

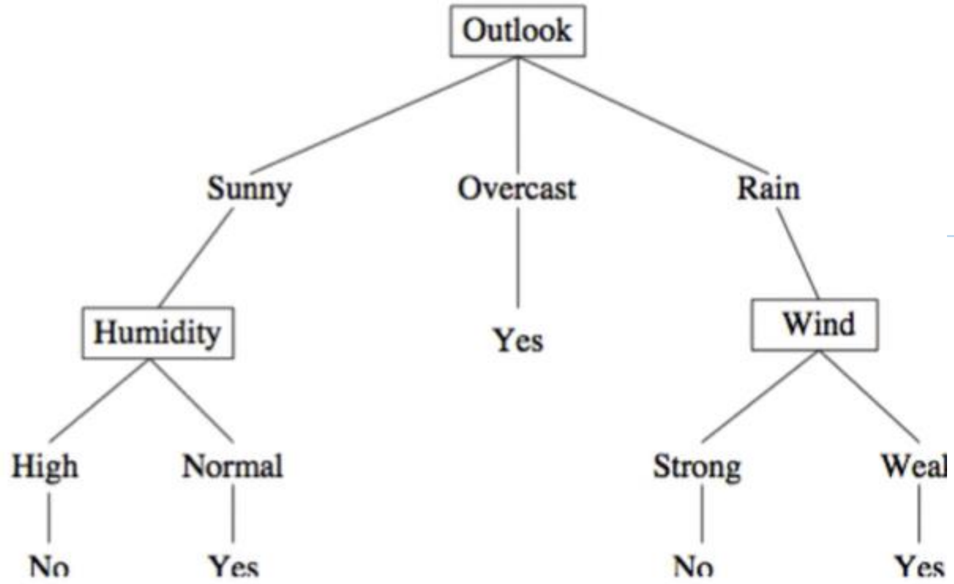



Figure credit: Tom Mitchell, 1997

# Decisin trees- continous attributes

- So far, we have investigated *categorical values*, but you can also use decision trees for *continuous attributes.*
- Pick a threshold to create a split! (e.g., temperature > 77.8)= true

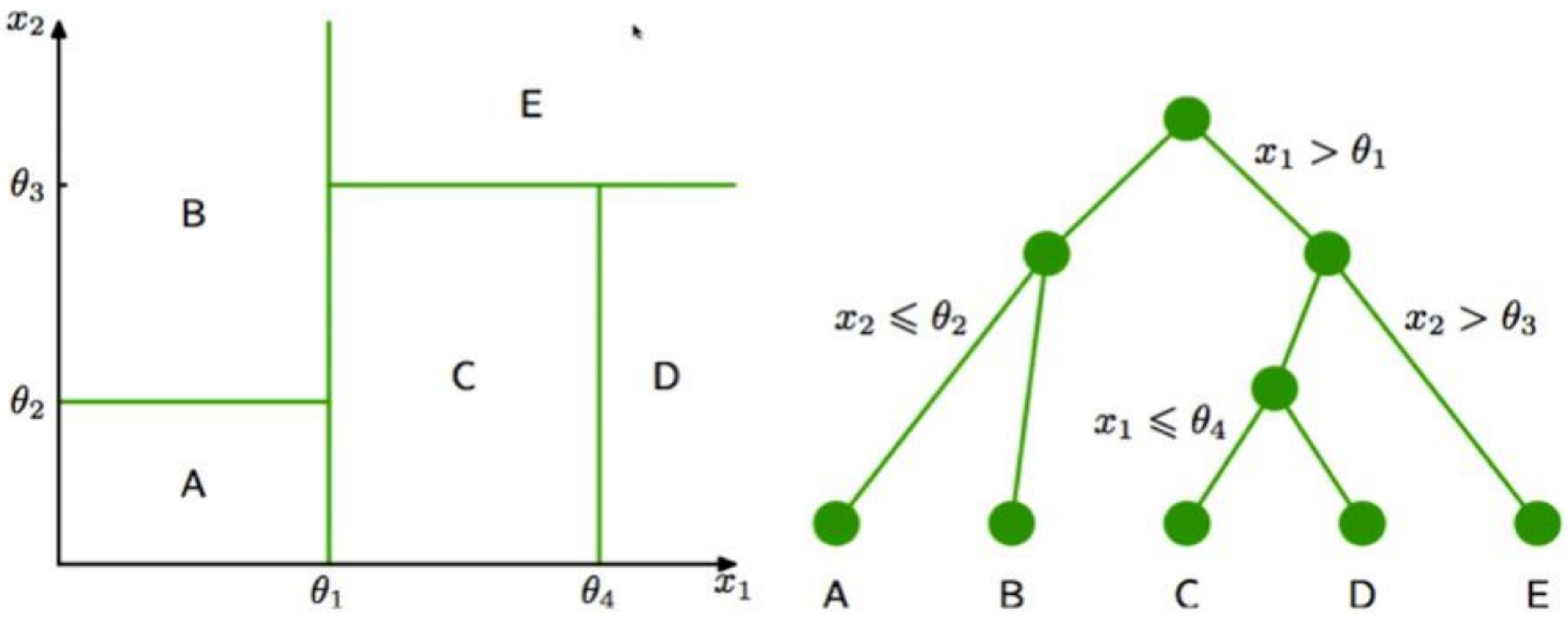



Figure credit: Chris Bishop, PRML

# Decisin trees- Multi-class classification

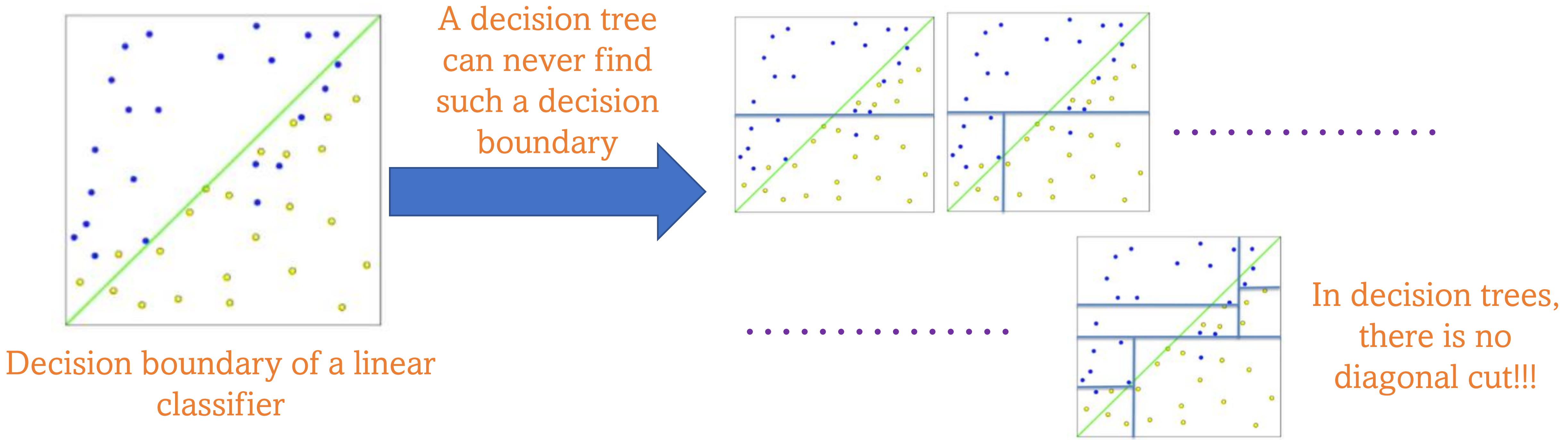
- Up until this point, we observed binary trees.
- Multi-class classification (i.e., *k* different classes)
  - Predict most frequent class in the subset
  - Entropy: $H(S) = - \Sigma_c \, p_{(c)} \log_2 p_{(c)}$
  - $p_{(c)}$= the proportion of the examples of class *c* in subset *S*

# Decisin trees- Prod & Cons

- Pros:
  - Interpretable: humans can understand the reason of decision
  - Easily handles irrelevant data (Gain=0)
  - Can handle missing data
  - Very compact: *#nodes* << *#training data after pruning*
  - Very fast at testing time: O(depth), *depth* << *#training data*

# Decisin trees- Prod & Cons

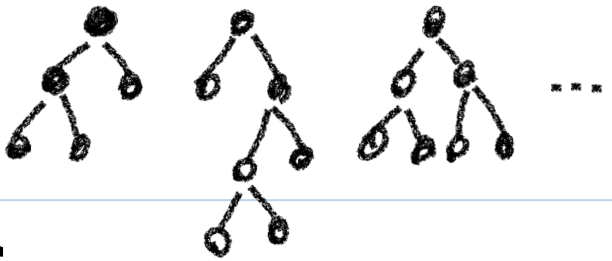
- Cons.
  - Greedy (may not find best tree)– not globally optimal!
    - Exponentially many possible trees– NP hard!
  - Only axis-aligned splits of data (especially in continuous data)



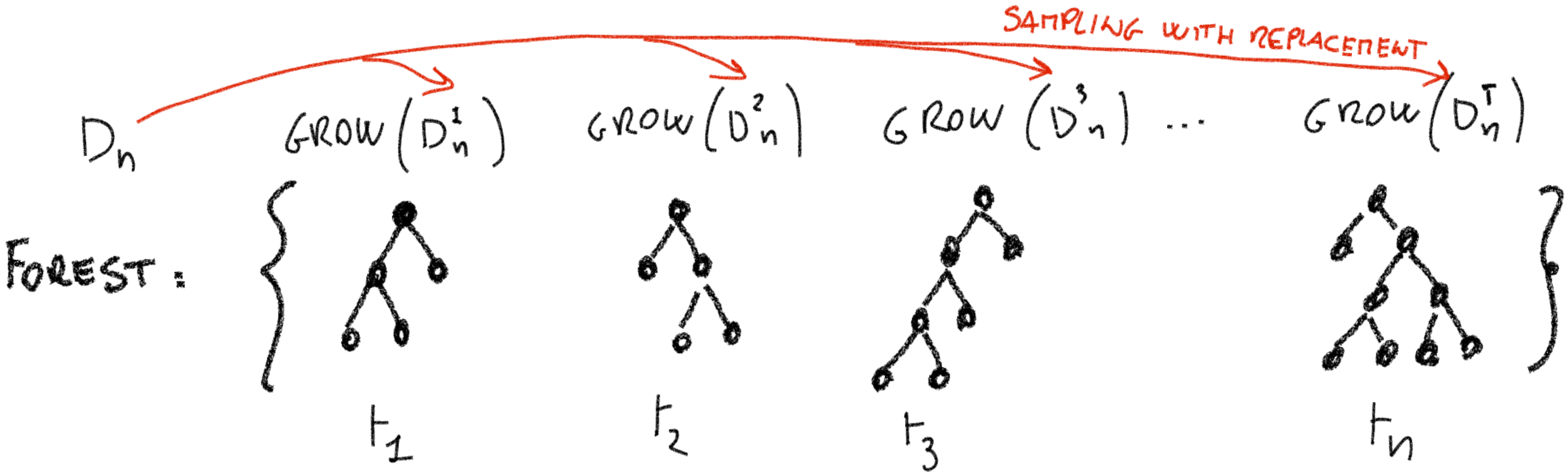
# Decision trees-- summary

- ID3: grows decision tree from the root to down
  - Greedily selects next best attribute using INF. GAIN
  - Entropy: How uncertain we are in terms of Yes/No in a set
  - Inf. Gain: reduction in uncertainty following a split
- Searches a complete hypothesis space
- Prefers smaller trees, high gain at the root
- Overfitting addressed by post-pruning
  - Prune nodes, while accuracy increases on validation set
- Fast, compact, interpretable

# RANDOM FOREST



- Random forests are ensembles of decision trees.
- Each tree is typically trained on a bootstrapped version of the training set (sampled with replacement).



# Random forest

- Split functions are optimized on randomly sampled features or are sampled completely at random (extremely randomized trees).
  - This helps obtaining decorrelated decision trees
- The final prediction of the forest is obtained by averaging the prediction of each tree in the ensemble $\mathcal{Q} = \{t_1, \ldots, t_T\}$

$$f_{\mathcal{Q}}(x) = \frac{1}{T}\sum_{j=1}^{T} f_t(x)$$

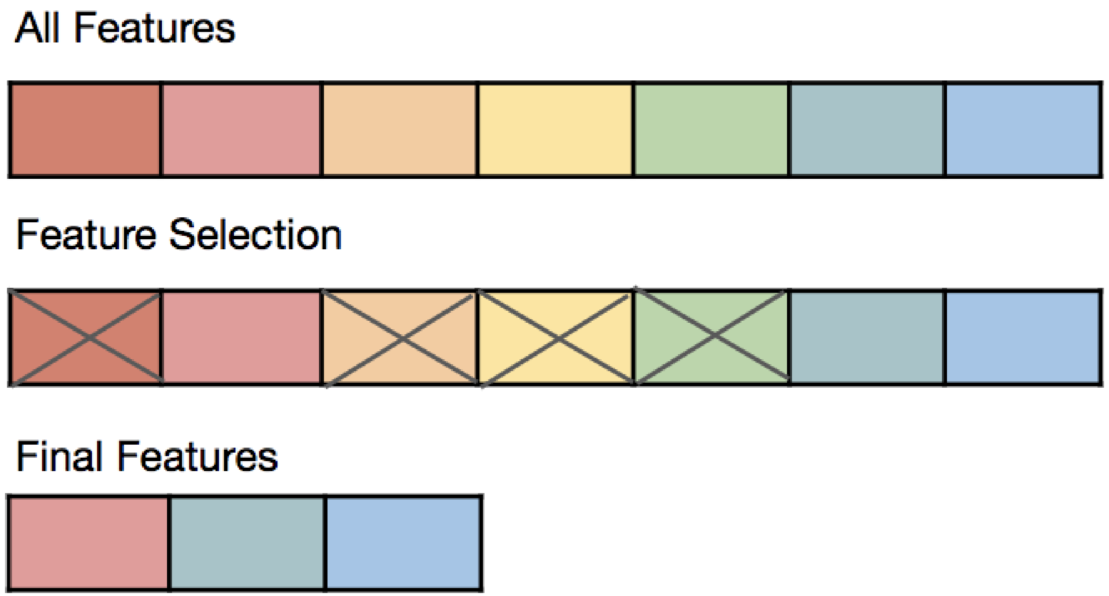
Average of T number of decision trees

# Coding tutorial

- Decision Trees: https://scikit-learn.org/stable/modules/tree.html
- Random Forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- Random Forest: https://www.datacamp.com/tutorial/random-forests-classifier-python

# Feature Selection

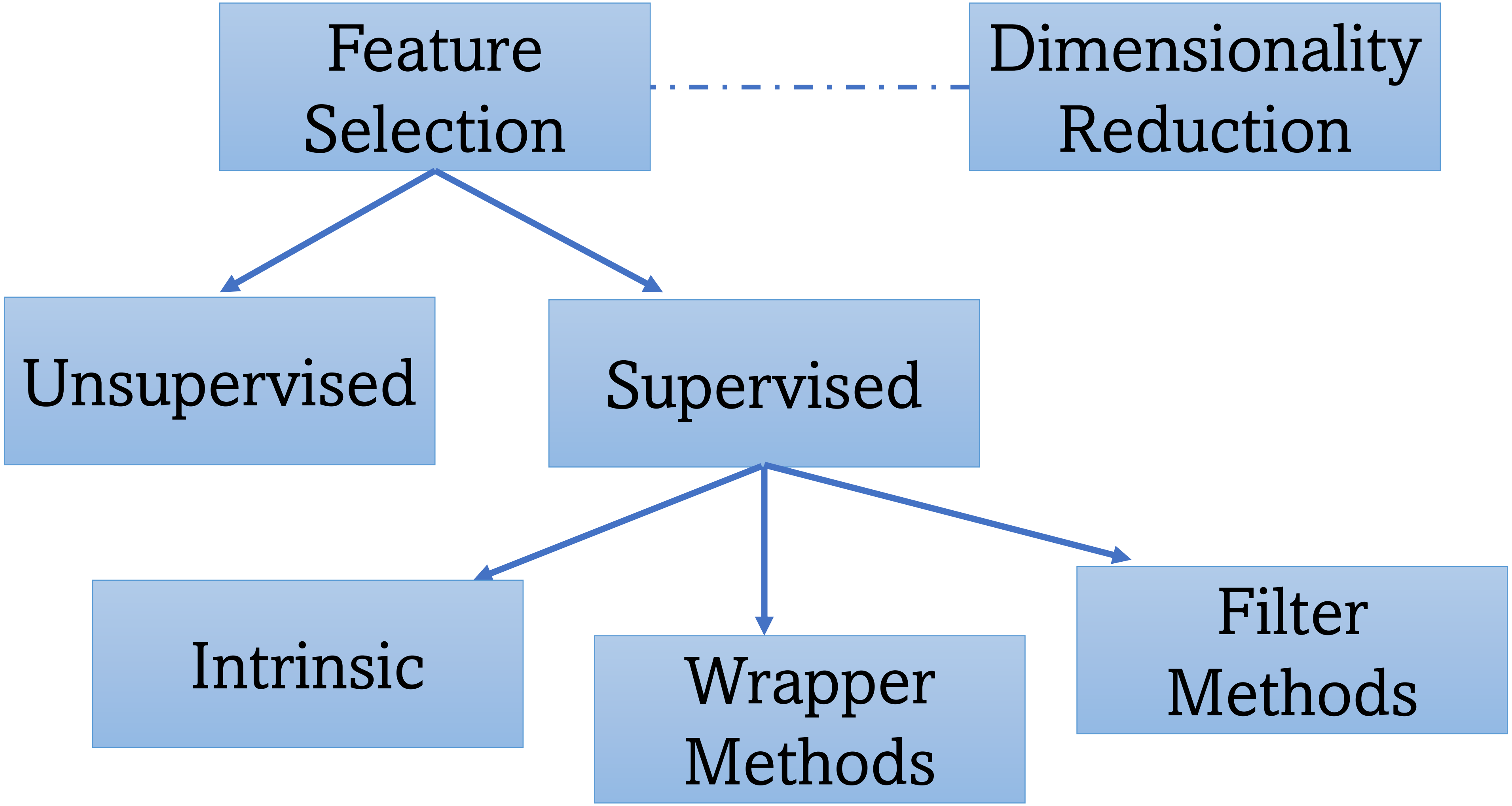
- The process of selecting the input variable to your model by using only relevant data and getting rid of “noise” in data.
- Because, the noisy (irrelevant) attributes can mislead your model, thus decrease its performance.



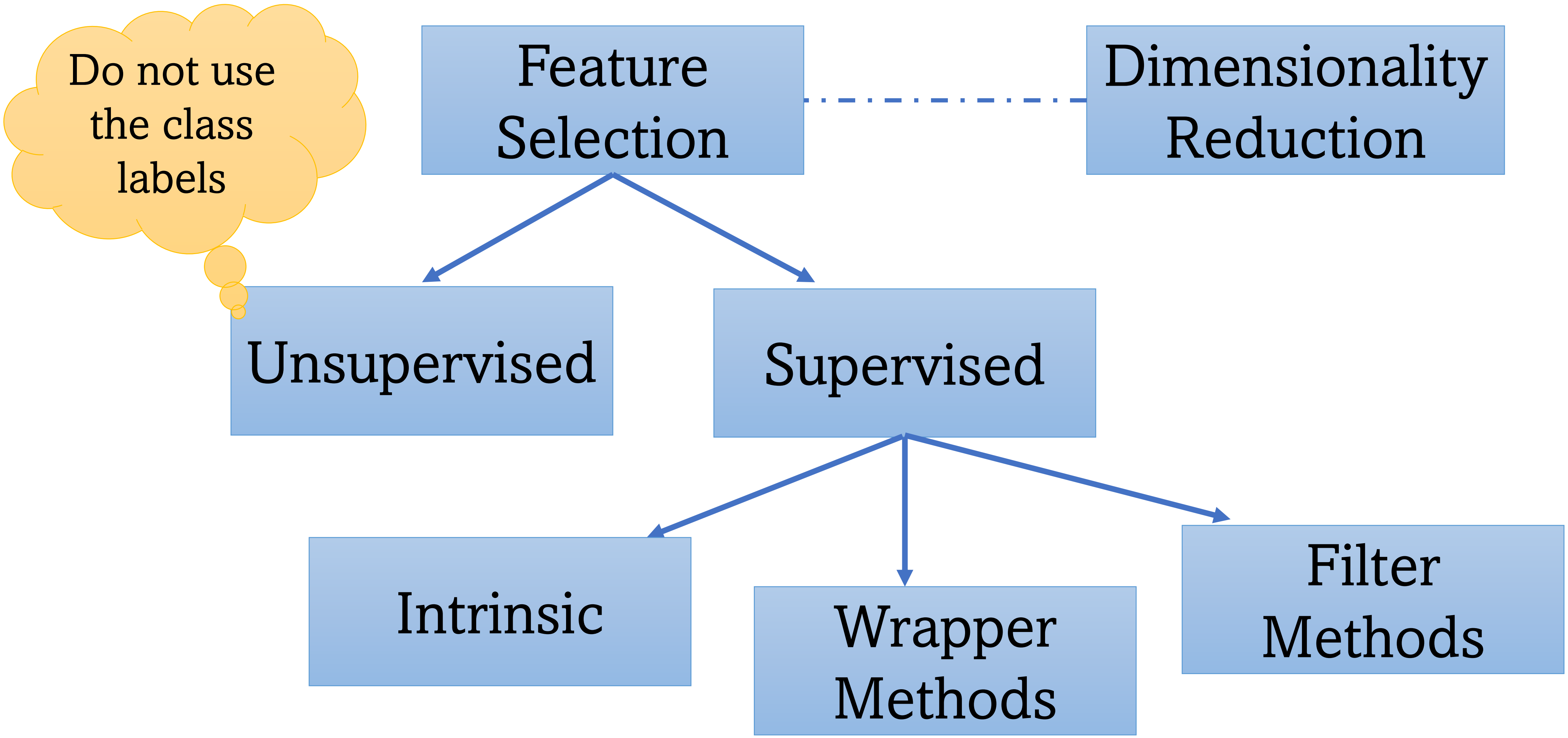
# Feature selection

- High dimensional data suffers from: *Curse of Dimensionality*
- Observations in a high-dimensional space are more sparse and less representative than those in a low-dimensional space.
- Using feature selection, we can optimize our model in several ways:
  - Prevent learning from noise (overfitting)
  - Improved performance, e.g., accuracy
  - Reduce training time (more features, typically means more training time)

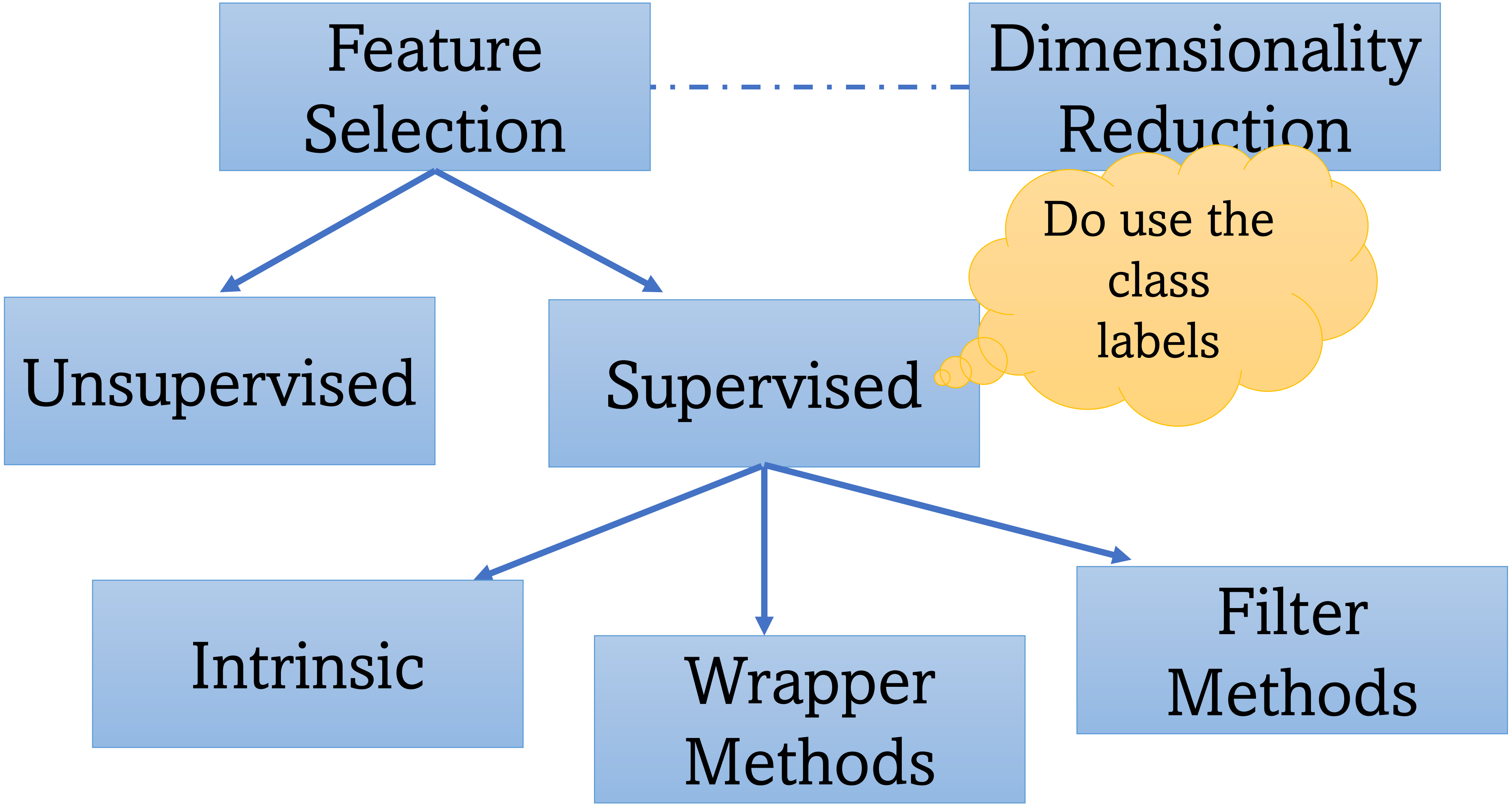
# Feature selection methods



# Feature selection methods



# Feature selection methods



# Feature selection methods

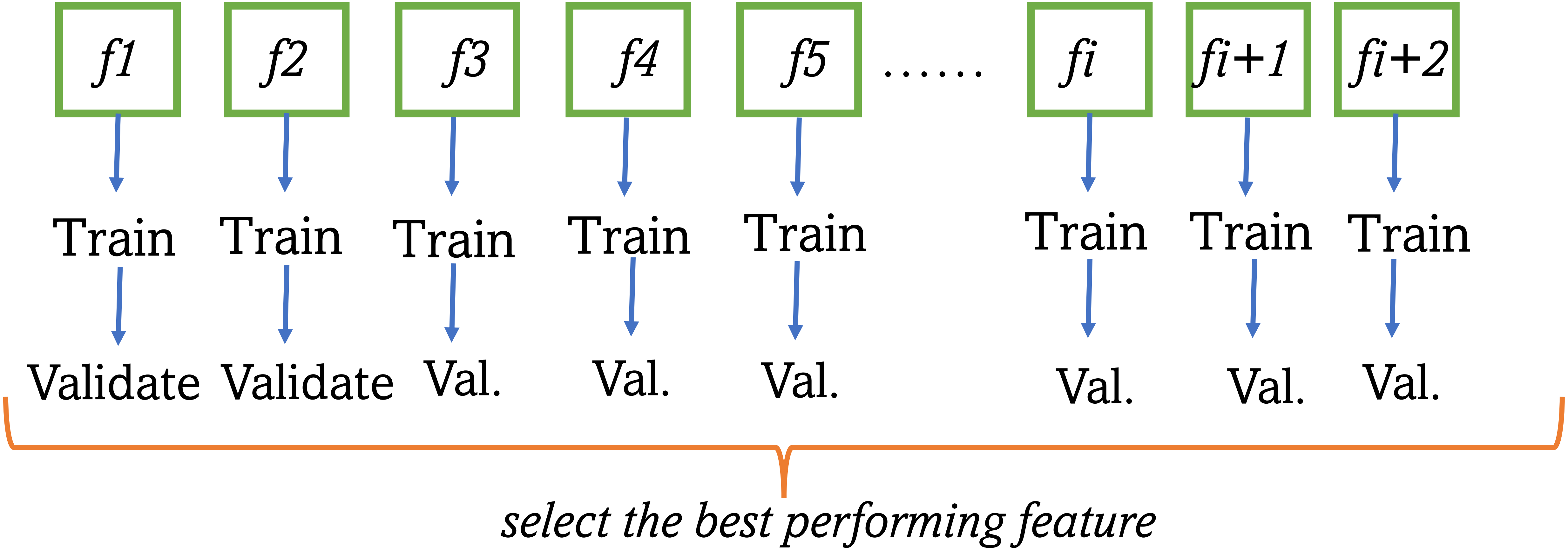
Supervised

- Intrinsic
  - Lasso regularization
  - Decision trees
- Wrapper Methods
  - Recursive & Iterative Methods
  - Genetic Algorithms
- Filter Methods
  - Pearson’s Coefficient
  - Chi squared
  - ANOVA Coefficient

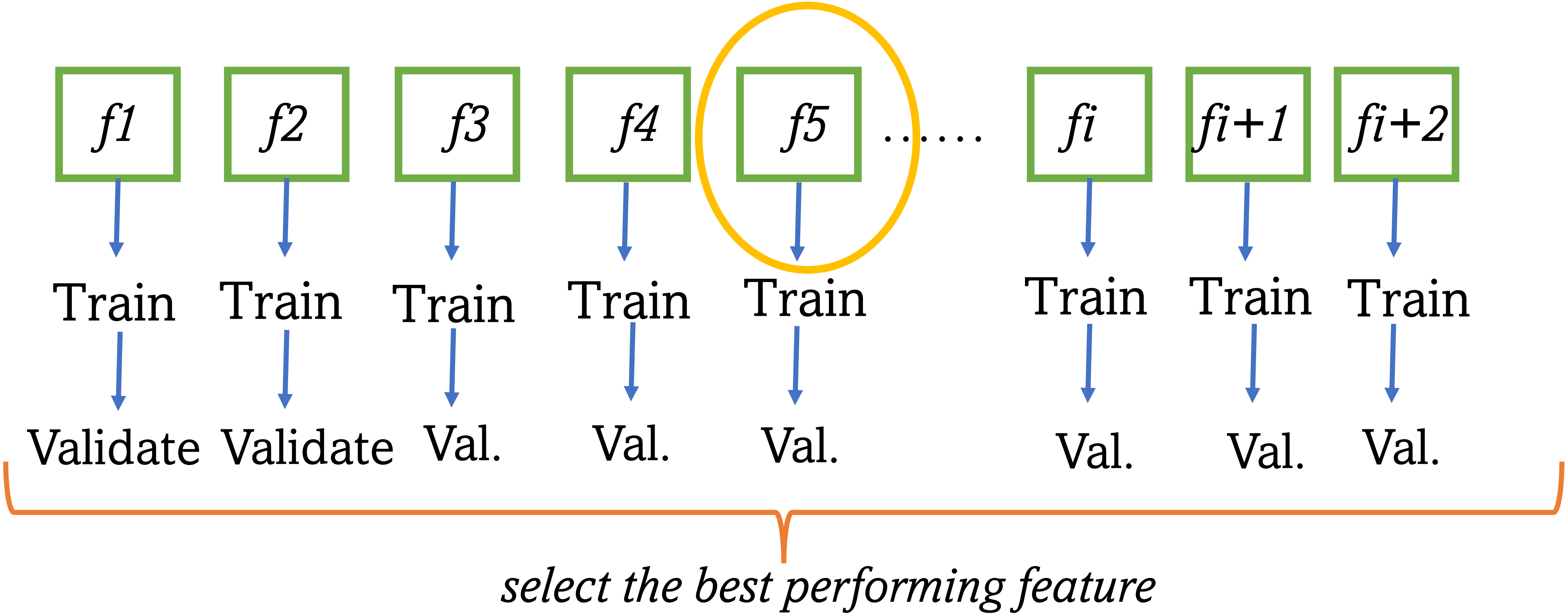
# Forward feature selection

- An iterative method in which we start with having a single feature in the model.
- In each iteration, we keep adding the feature which improves our model the most, till an addition of a new variable does not improve the performance of the model.

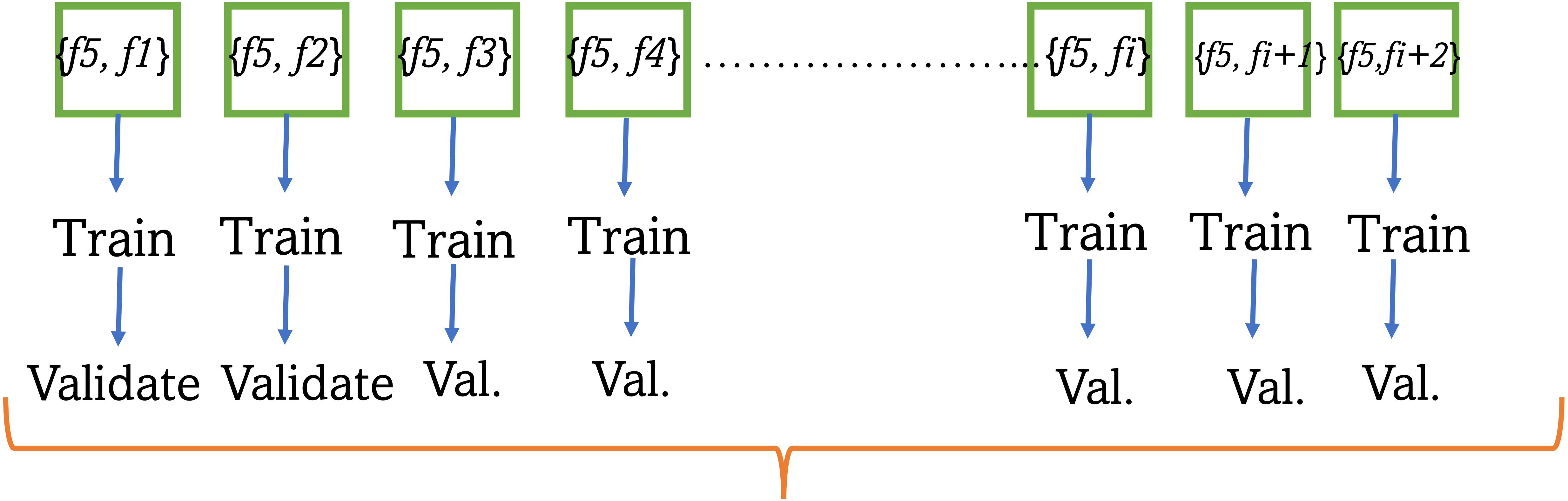
# FORWARD FEATURE SELECTION – ITERATION 1



# Forward Feature Selection – Iteration 1

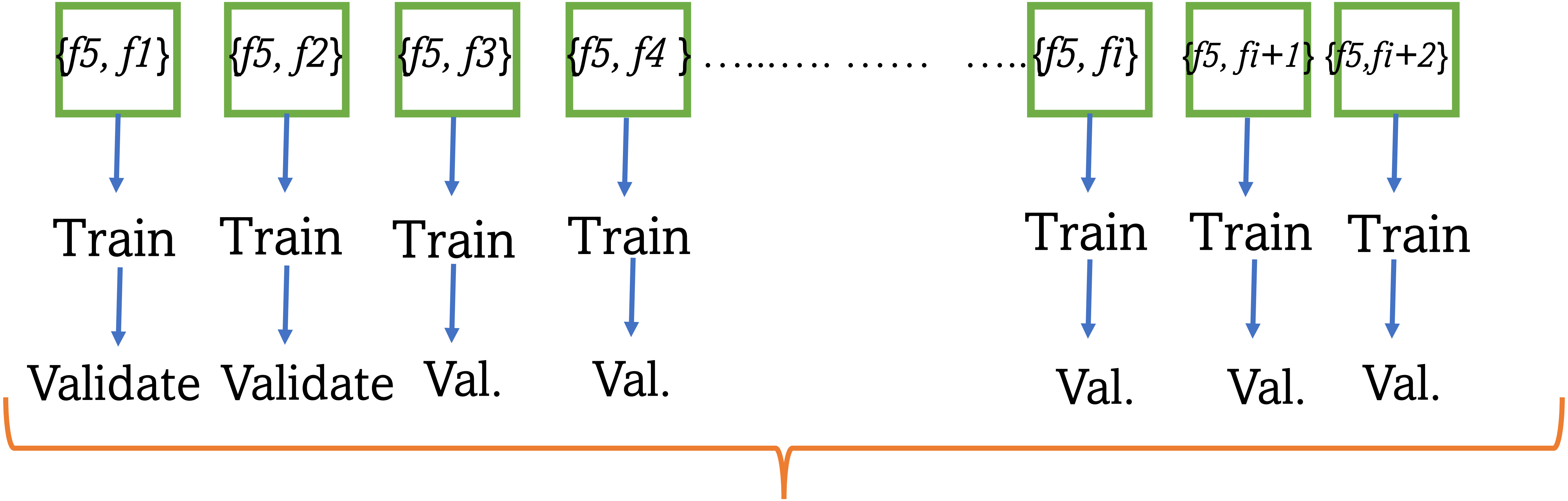


# Forward feature selection-Iteration 2




*select the best performing feature*

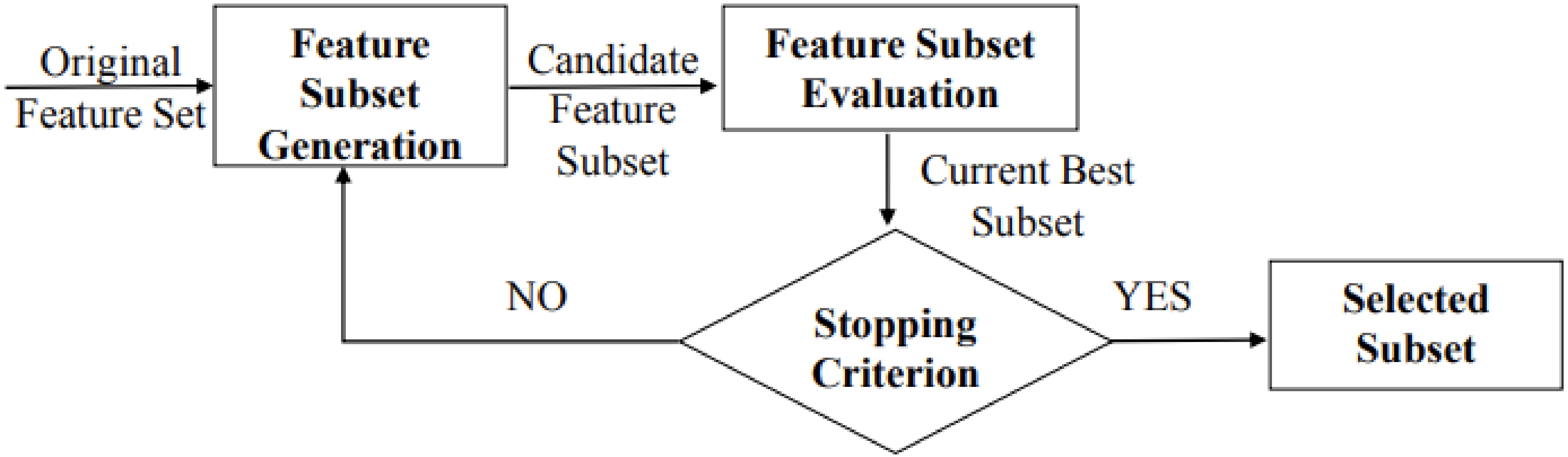
# FORWARD FEATURE SELECTION-ITERATION 2




*select the best performing feature*

*if: performance(f5) << performance(f5,new) continue iteration, otherwise stop!*

# Forward feature selection



# BACKWARD FEATURE ELIMINATION

- An iterative method in which we start with all features, and we remove the least significant feature at each iteration such that removing it increases (rarely not changes) the performance of the model. We repeat this until no improvement is observed on removal of features.

# Coding tutorial

- Feature Selection: https://scikit-learn.org/stable/modules/feature_selection.html

1.13.5. Sequential Feature Selection