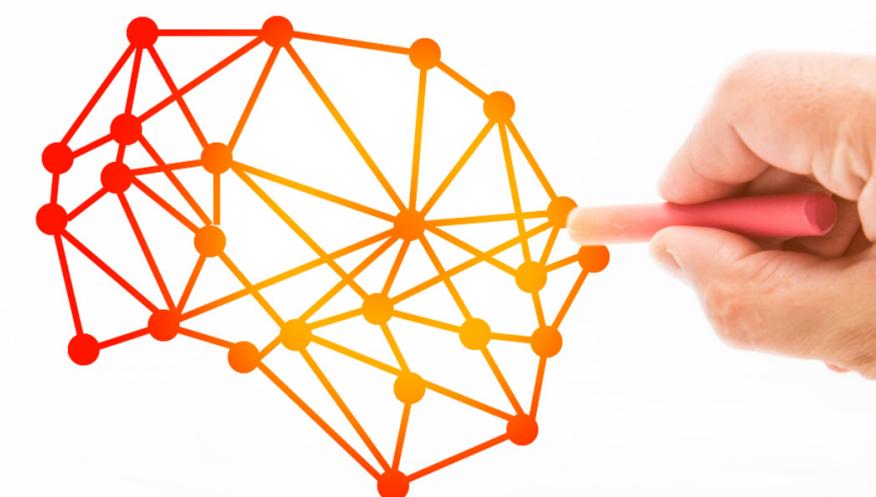


CREWES Data Science

Lecture 10: Tree Based Models for Regression

Marcelo Guarido

www.crewes.org





Today's Agenda

- Part 1:** — Decision Tree
- Part 2:** — Ensemble Models
- Part 3:** — Random Forest
- Part 4:** — Gradient Boosting
- Part 5:** — Live Coding

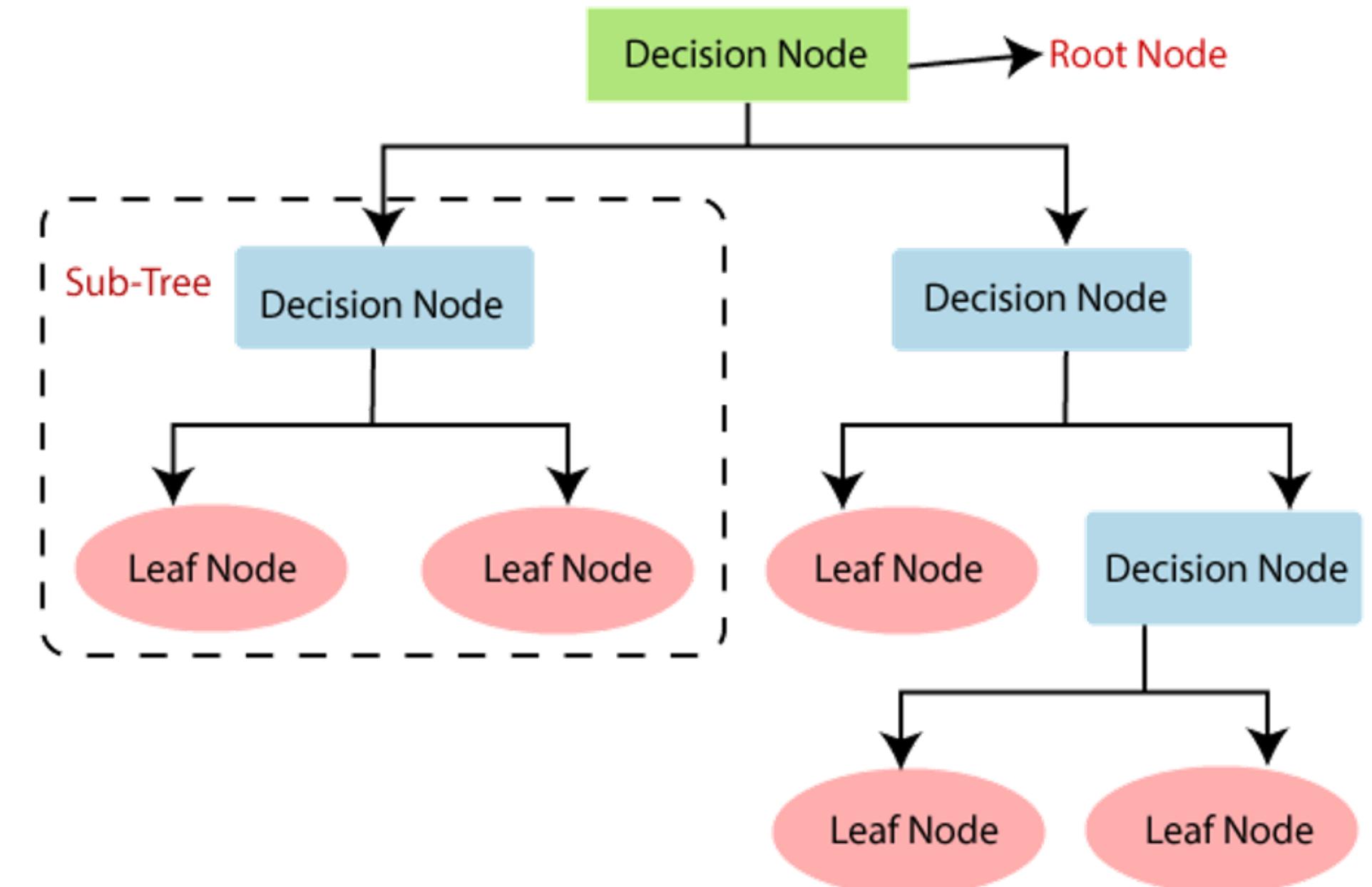
Part 1:

DECISION TREE

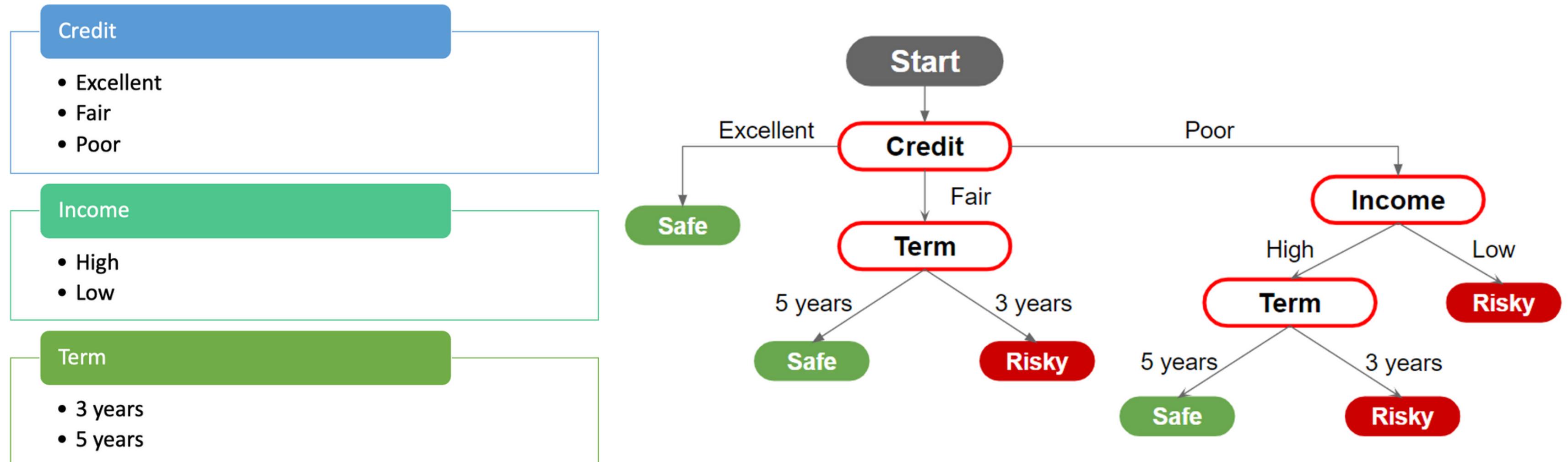


Decision Tree

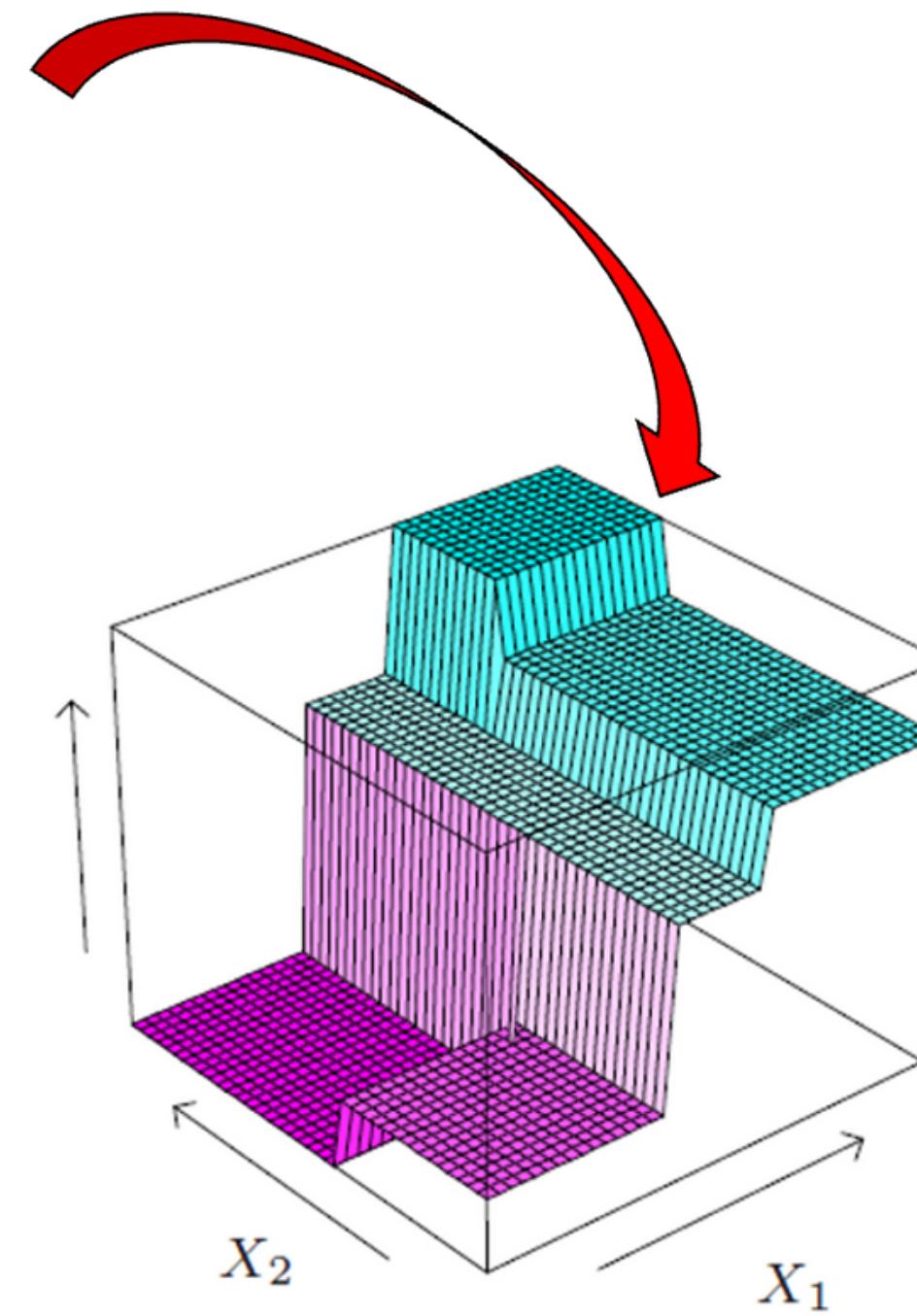
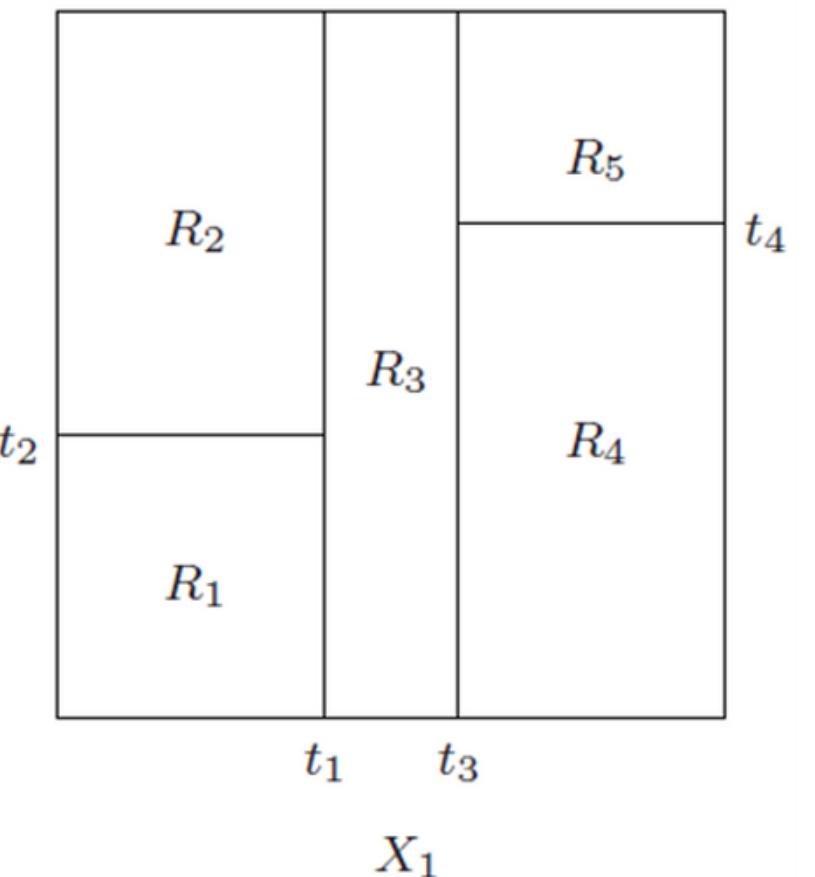
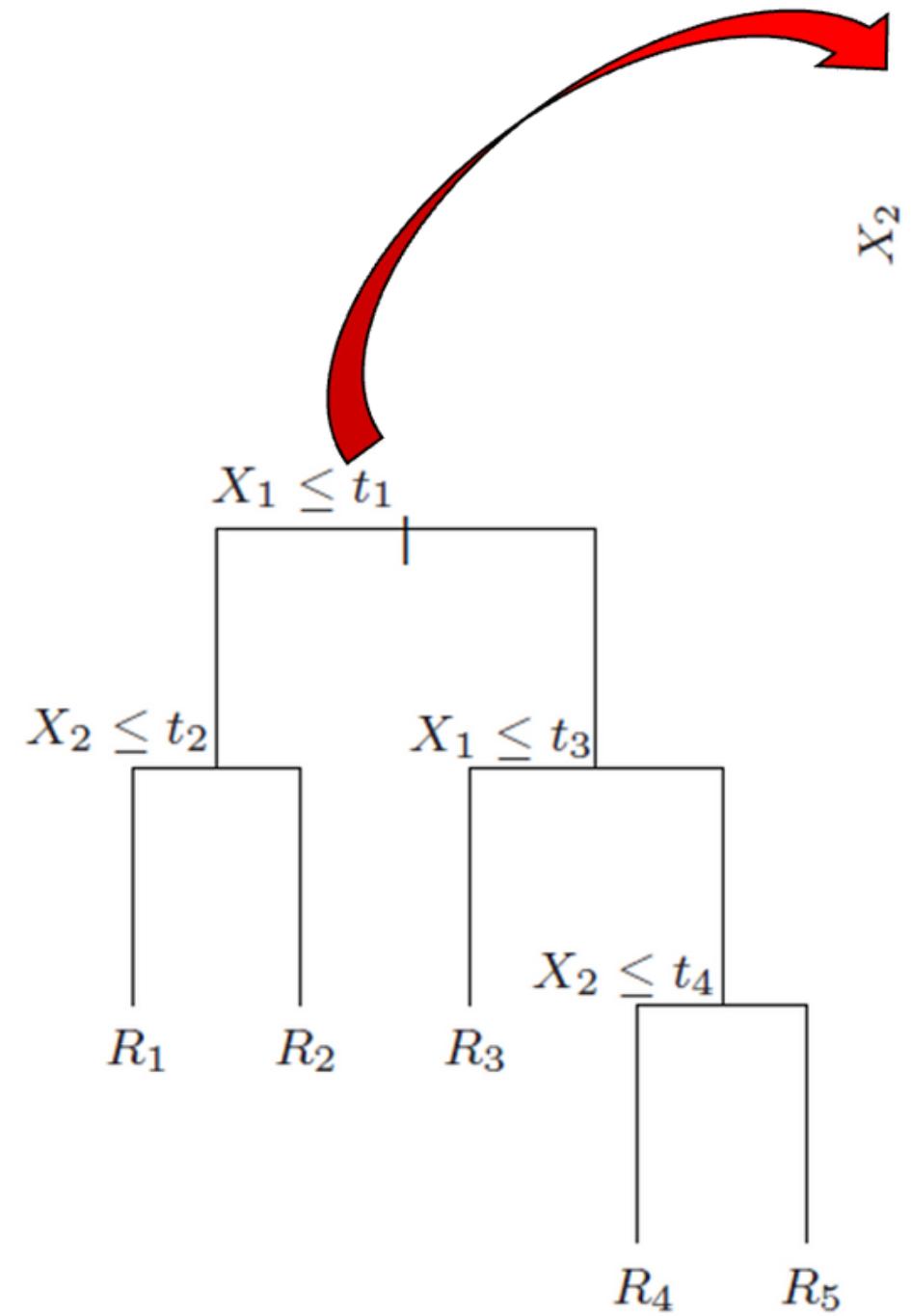
- Used for regression or classification
- Simple algorithm
- Basically, a sequence of “if” conditions
- Select the branch separation that minimizes the error
- Can handle continuous and categorical features
- Usually, considered a weak model



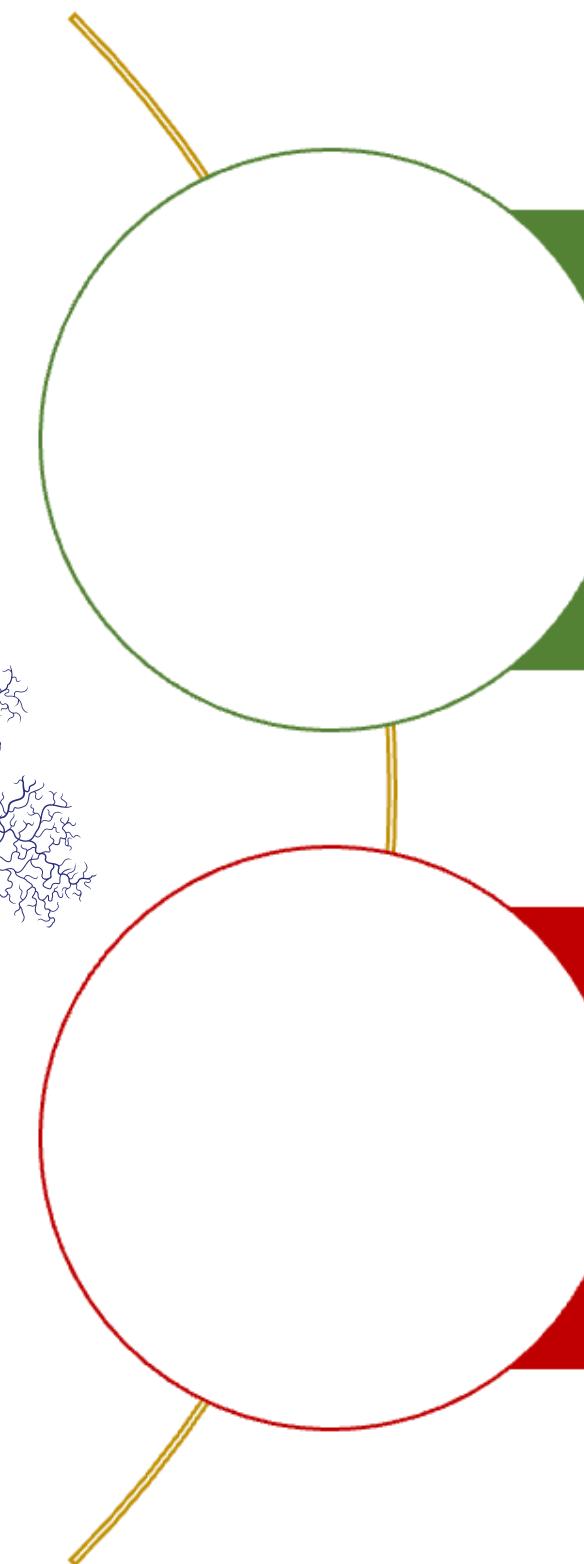
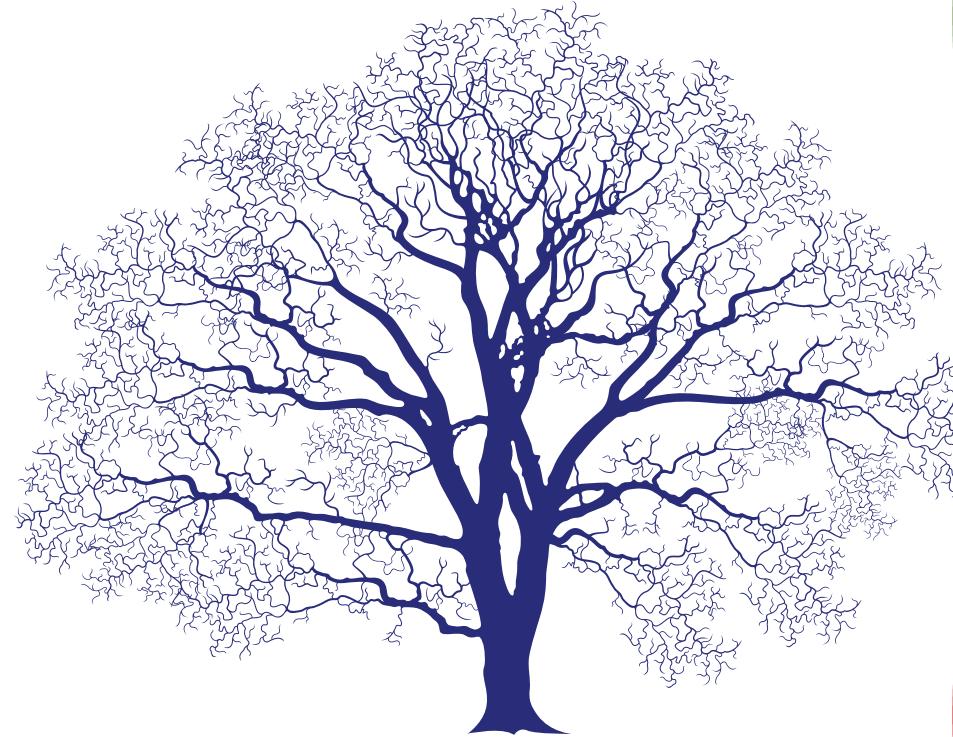
Decision Tree



Decision Tree



Decision Tree



Pros

- Requires less data preparation
- Numerical and categorical data
- No scaling or normalization
- Not affected by missing data
- Easy and intuitive to interpret and explain

Cons

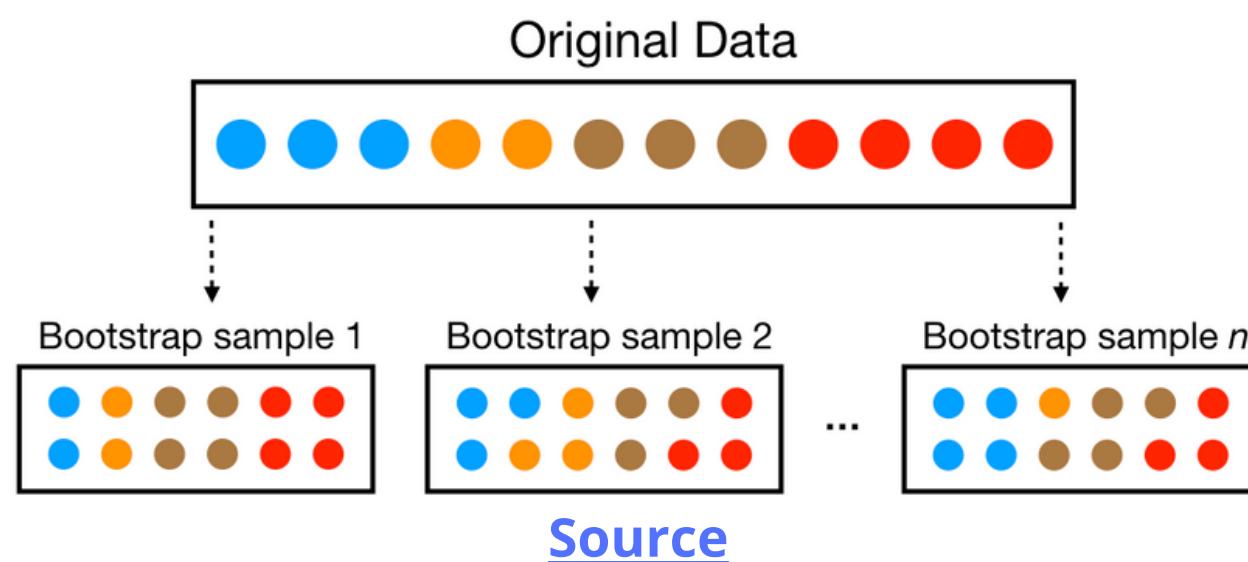
- Unstable
- Overfitting
- Tricky to apply for regression
- Relatively slow
- Limited on complex data

Part 2:

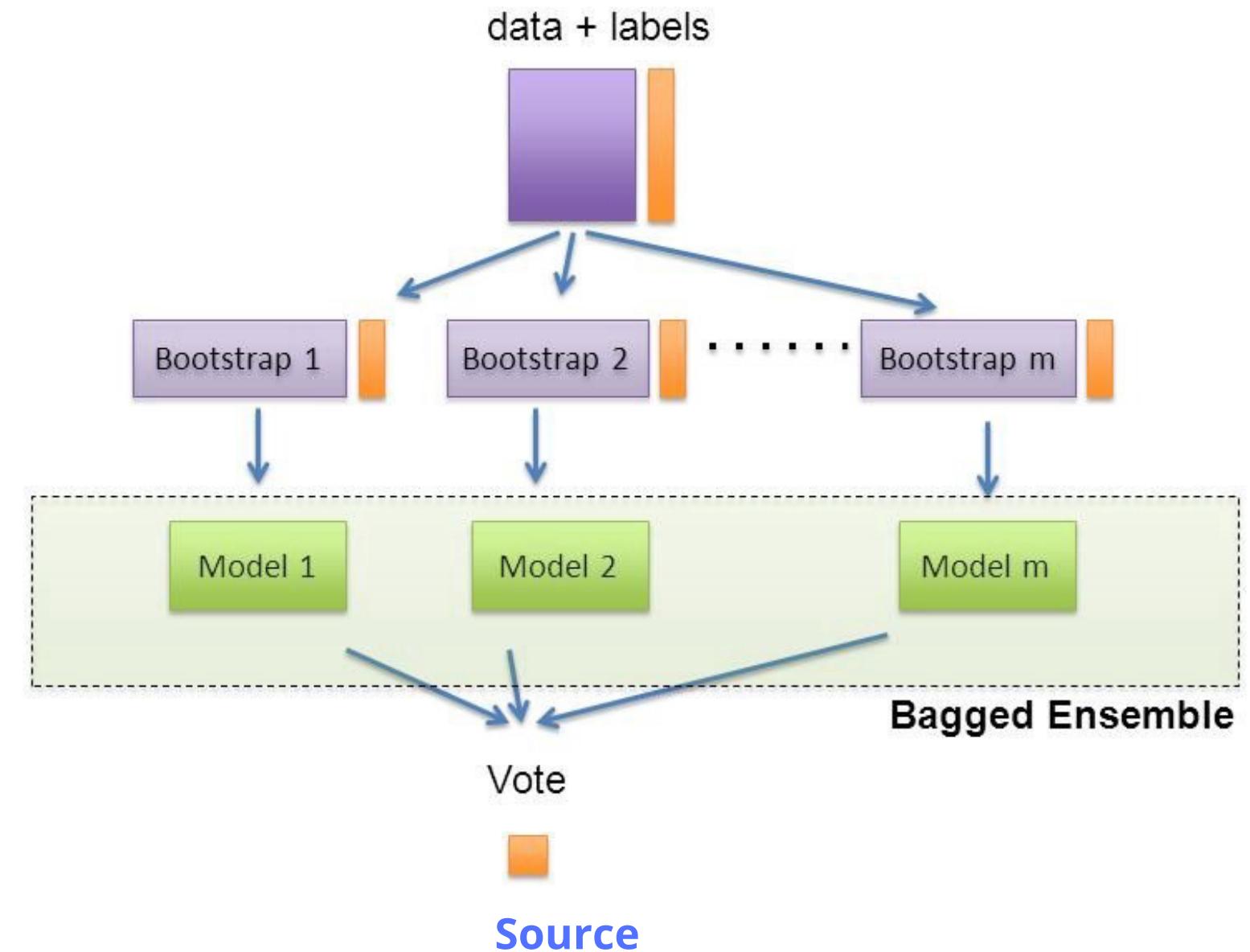
ENSEMBLE MODELS

Ensemble Models

- Combination of "weak" models
- Divide the train data into different subsets (Bootstrap)
- Train on different subsets of the training data (Bagging)
- Vote system (hard or soft)
- Regression instead of vote



"Bagging" : Bootstrap AGGregatING



Part 3:

RANDOM FOREST

Random Forest

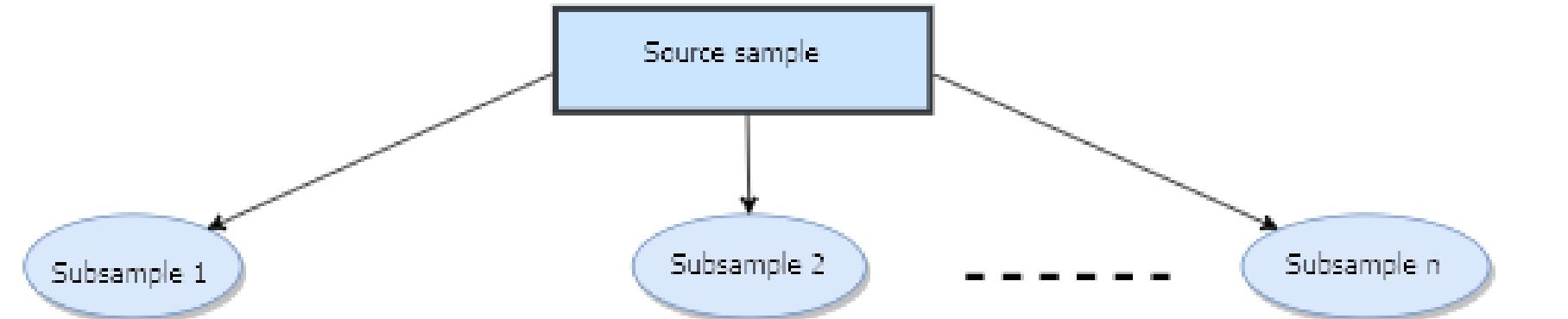
- Ensemble of decision trees
- Each tree is trained over a subset of the features randomly selected
- Can also train over a random selected subsample of the data
- Vote system (classification)
- Average predictions (regression)



Random Forest

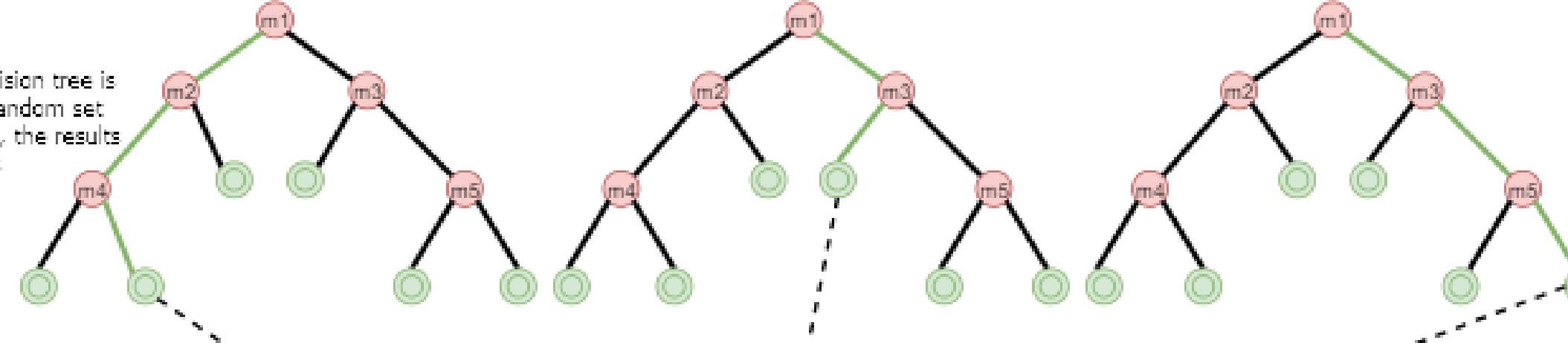
Bootstrap sampling

r (percentage) examples are selected (0.63 in classical implementation) in n random subsamples



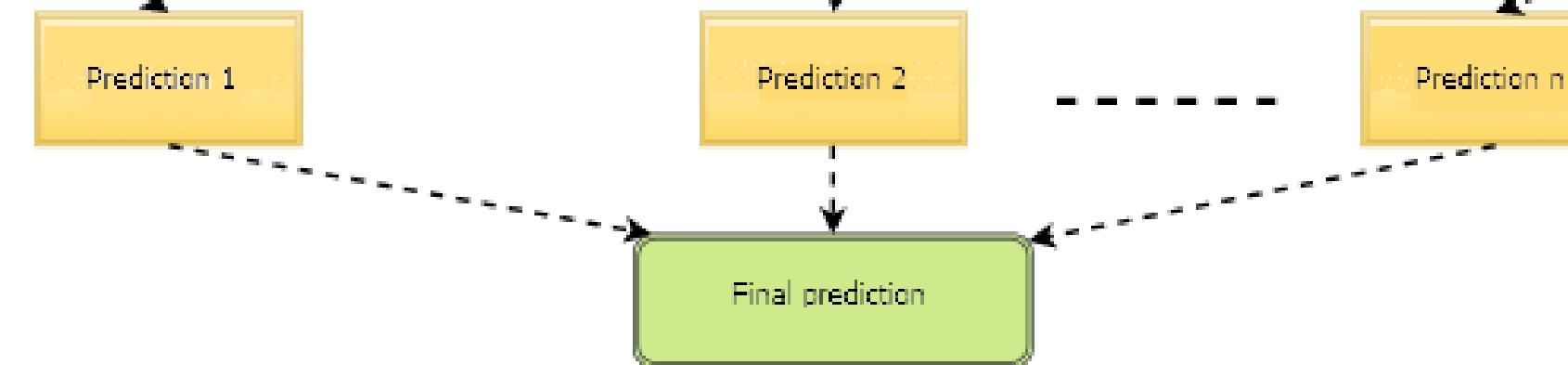
Building the models

for each subsample, a decision tree is constructed based on a random set of m features (covariants), the results fall into leaves



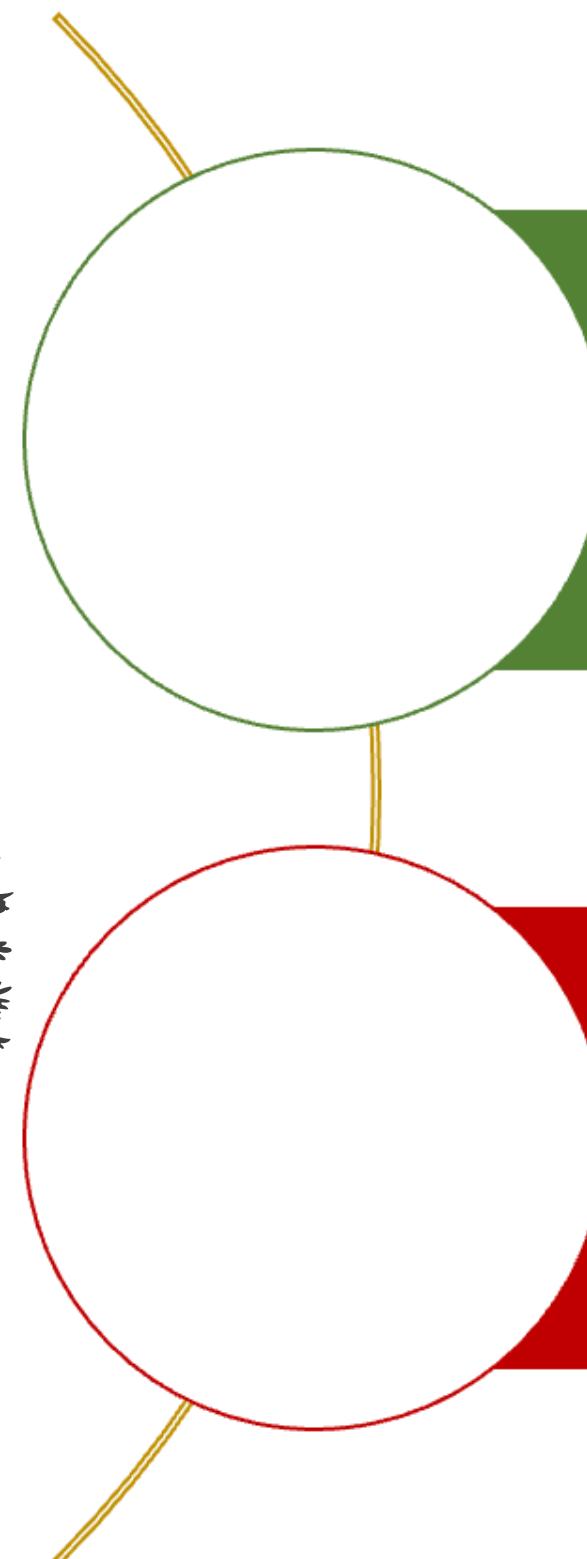
Bootstrap aggregating

results from all constructed trees are gathered and averaged



[Source](#)

Random Forest



Pros

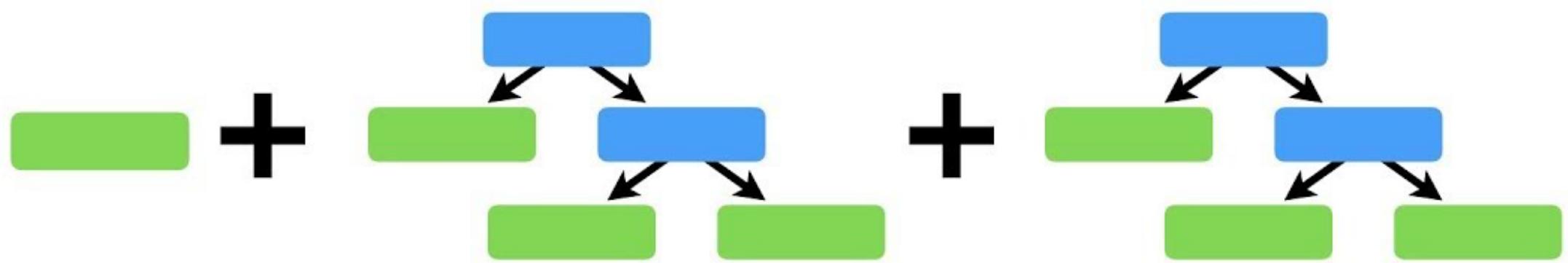
- Requires less data preparation
- Numerical and categorical data
- Handle outliers
- Less overfitting
- Still relatively easy to interpret

Cons

- Biased with categorical features
- Bad with sparse features
- Slow training

Part 4:

GRADIENT BOOSTING

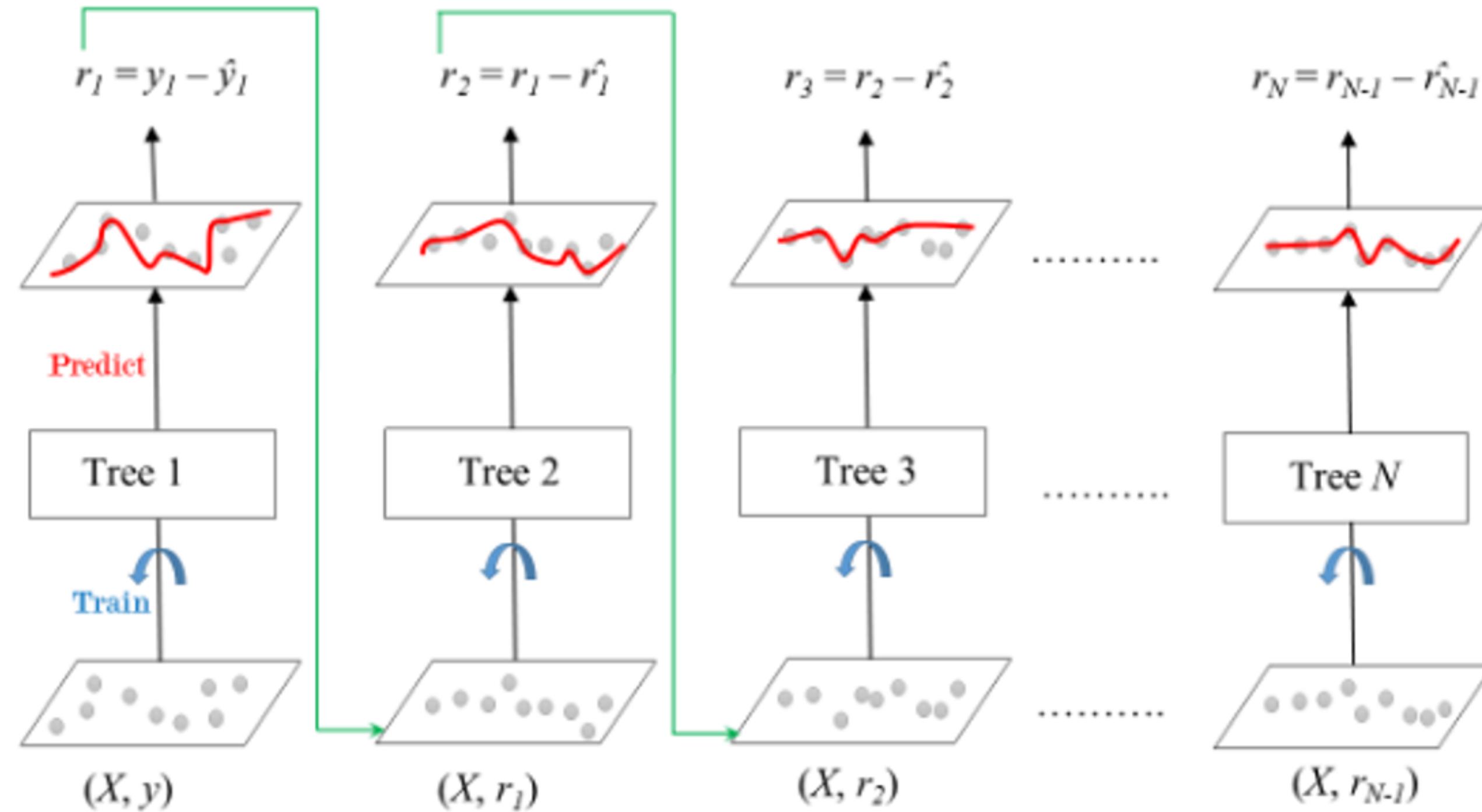


Gradient Boosting

- Ensemble of decision trees
- Sequential trees
- Trained over residuals of previous tree
- Gradient descent

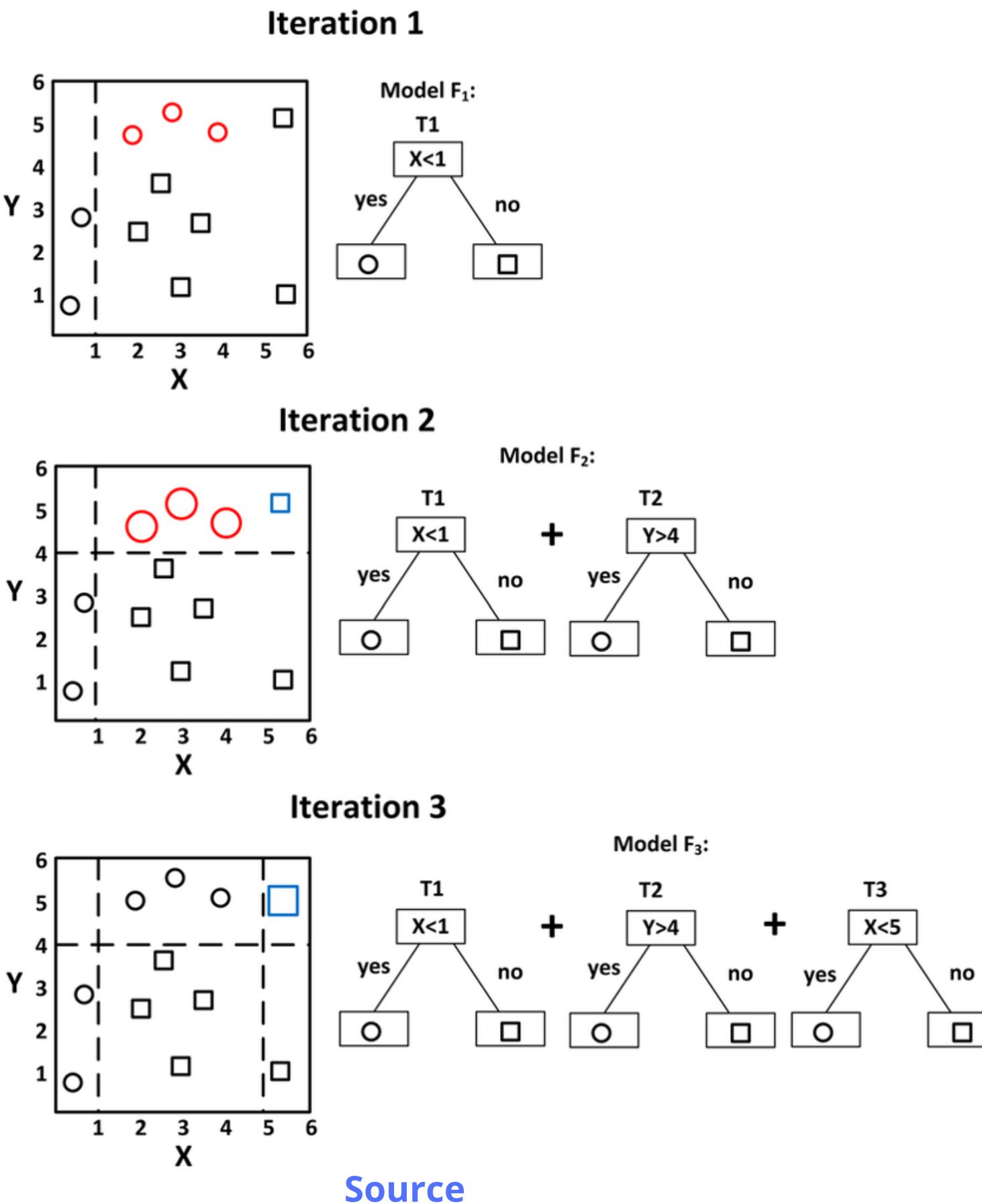


Gradient Boosting



[Source](#)

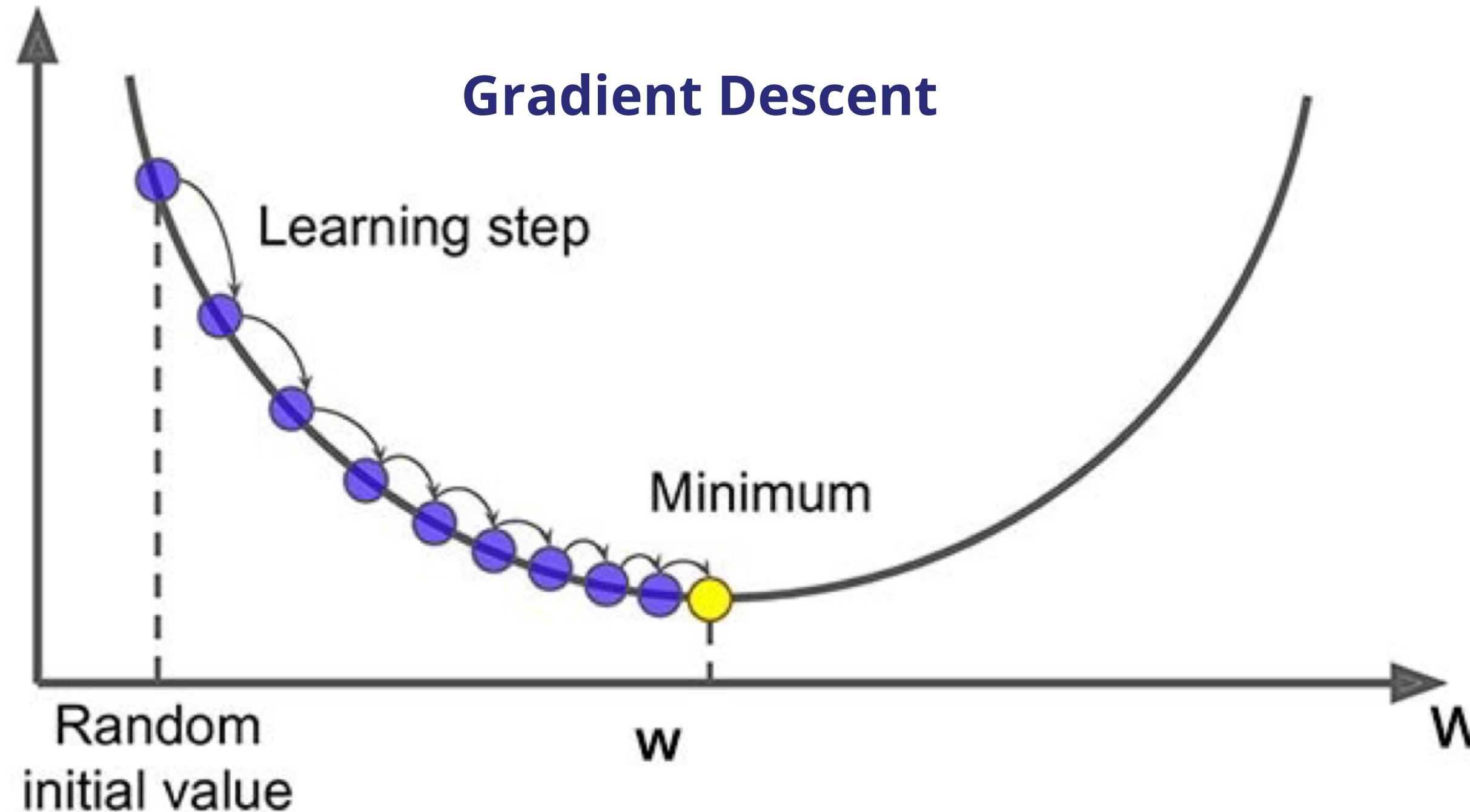
Gradient Boosting



[Source](#)

Gradient Boosting

Cost



[Source](#)

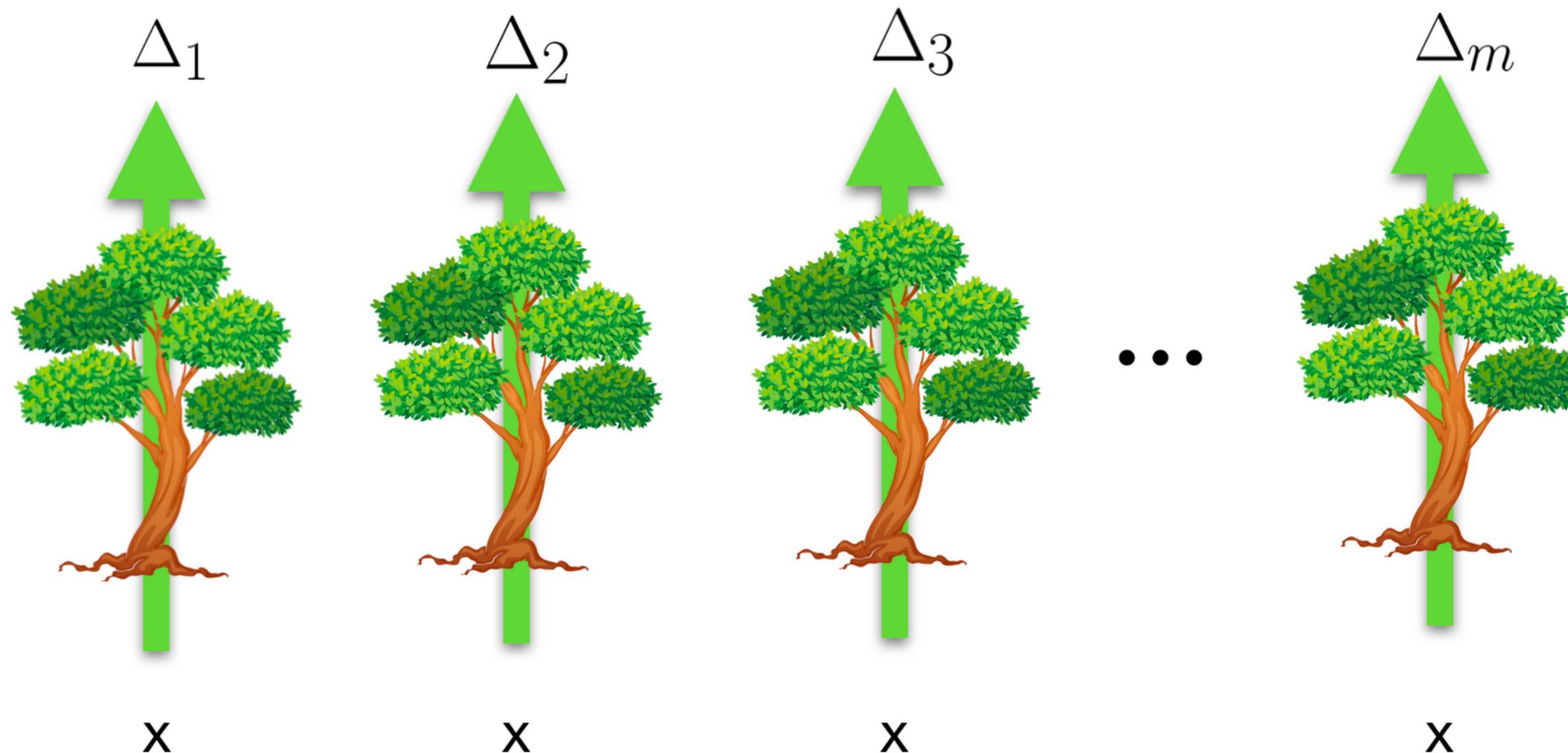
Gradient Boosting

$$\hat{y}_1 = \Delta_1$$

$$\hat{y}_2 = \hat{y}_1 + \eta \Delta_2$$

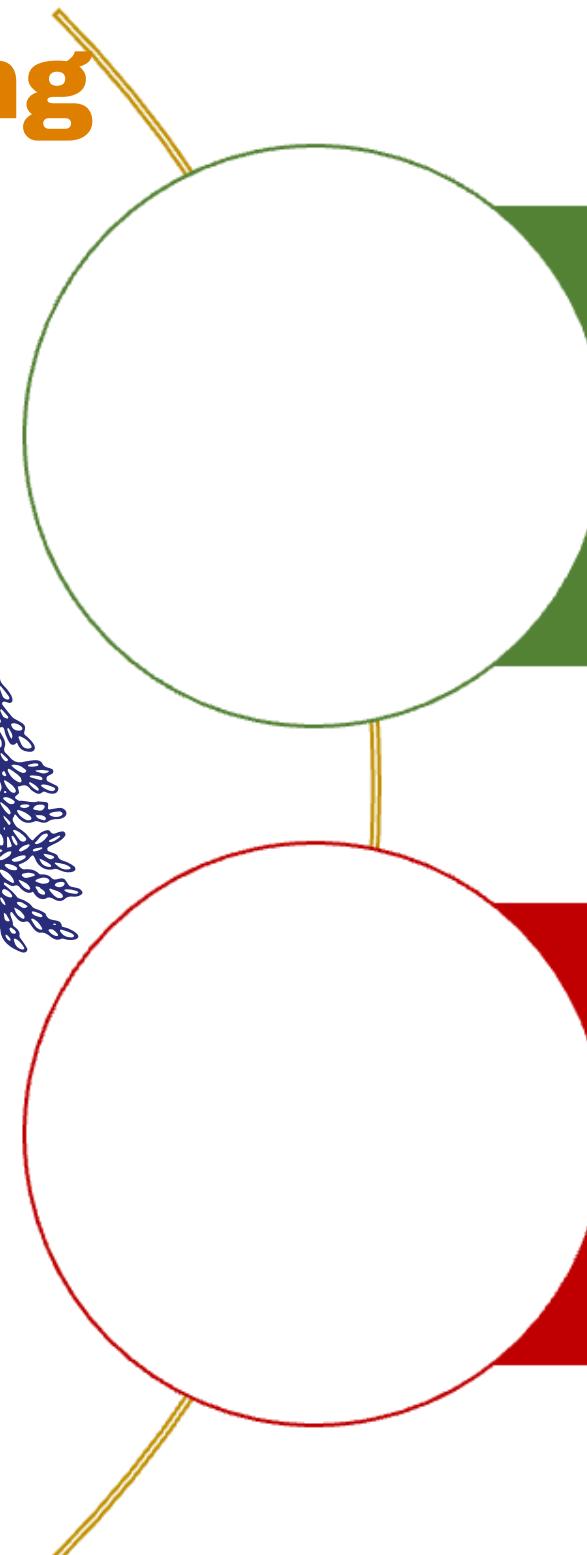
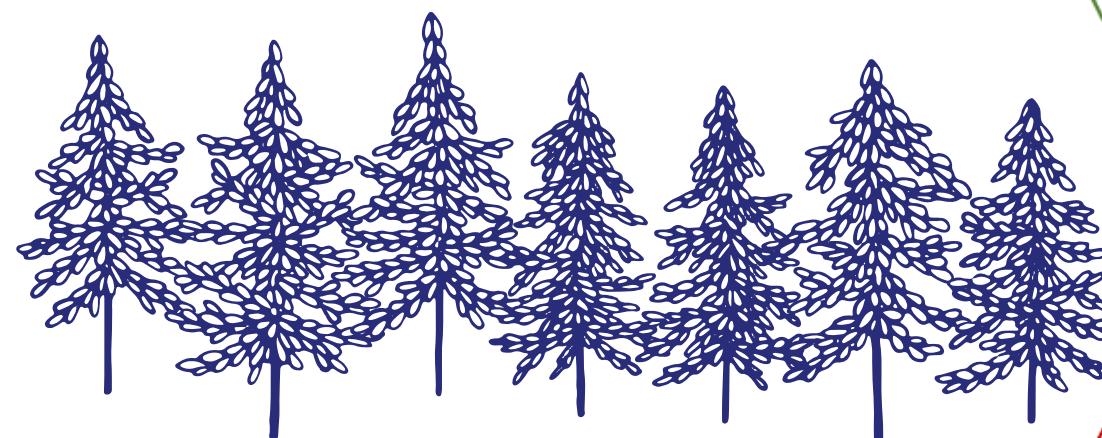
$$\hat{y}_3 = \hat{y}_2 + \eta \Delta_3$$

$$\hat{y}_m = \hat{y}_{m-1} + \eta \Delta_m$$



[Source](#)

Gradient Boosting



Pros

- Requires less data preparation
- Numerical and categorical data
- No scaling or normalization
- Less overfitting
- Still relatively easy to interpret

Cons

- Sensitive to outliers
- Hard for parallel processes
- Can get quite slow

Part 5:

LIVE CODING (PYTHON)

```
    mirror_mod.mirror_object = selected_object
    mirror_mod.mirror_axis = "MIRROR_X"
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
    if operation == "MIRROR_Y":
        mirror_mod.mirror_axis = "MIRROR_Y"
        mirror_mod.use_x = False
        mirror_mod.use_y = True
        mirror_mod.use_z = False
    if operation == "MIRROR_Z":
        mirror_mod.mirror_axis = "MIRROR_Z"
        mirror_mod.use_x = False
        mirror_mod.use_y = False
        mirror_mod.use_z = True

    #selection at the end -add to selection
    mirror_ob.select= 1
    mirror_mod.select=1
    bpy.context.scene.objects.active = mirror_mod
    ("Selected" + str(modifier_index))
    mirror_ob.select = 0
    bpy.context.selected_objects.append(mirror_mod)
    data.objects[one.name].select = 1
    print("please select exactly one object to mirror")

--- OPERATOR CLASSES ----

@operator
class MIRROR_OT_Mirror(bpy.types.Operator):
    bl_idname = "object.mirror"
    bl_label = "X mirror to the selected object.mirror_mirr... X"
    bl_options = {'REGISTER', 'UNDO'}
```