Politecnico di Bari

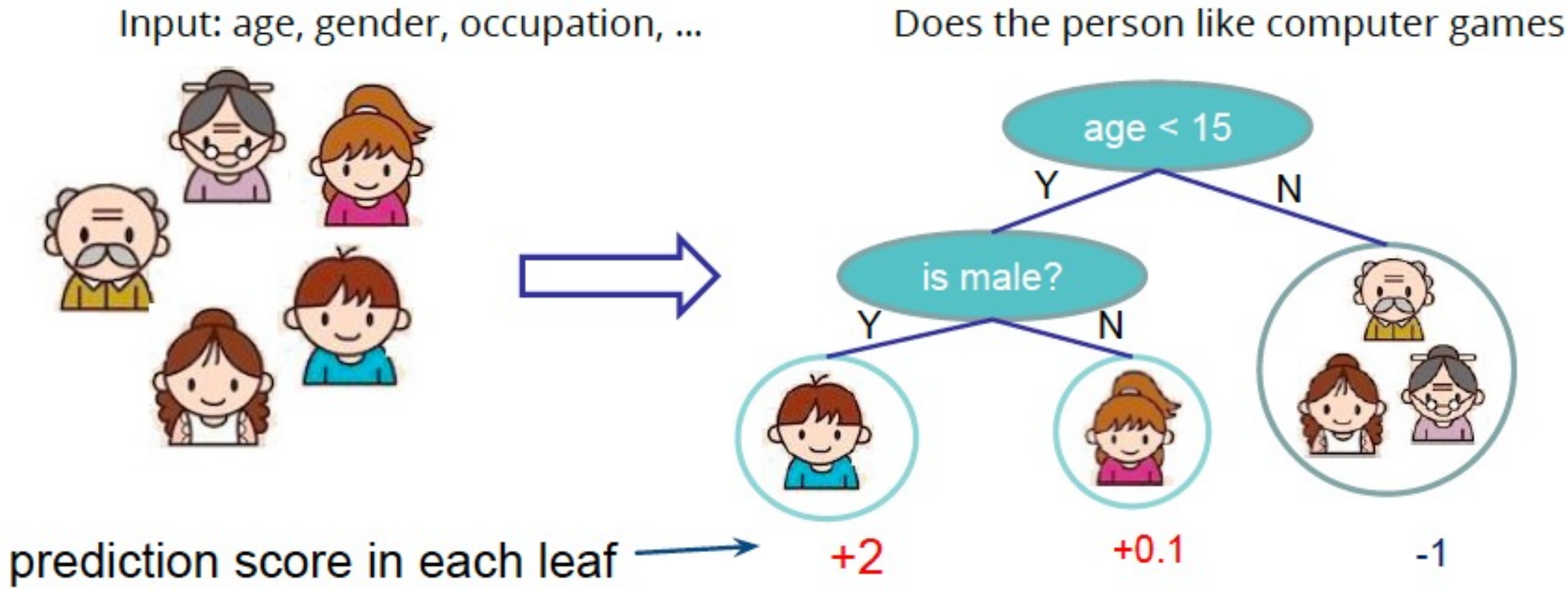
DIPARTIMENTO INTERATENEO DI FISICA

Dottorato in Fisica – XXXIX ciclo - 2024

Machine Learning techniques for particle physics

Federica Maria Simone - federica.simone@poliba.it

Decision Trees



(image credit: xgboost)

Decision Tree - formalism

Input space:

- Examples $\mathbf{X} \subset \mathbb{R}^{\boldsymbol{n}}$, each being $(x_1, x_2, x_3 \ldots x_n)$
- Labels $\mathbf{Y} \subset \mathbb{R}$, one per example

Let's say we have ℓ examples, our input space can be written as $\mathbf{X}^{\ell} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{\ell}$

A decision tree is a **partitioning of the input space** into k domains $R_1, R_2 \ldots R_k \subset \mathbb{R}^{\boldsymbol{n}}$, such as $R_1 \cup R_2 \cup \cdots \cup R_k = \mathbf{X}$.

To each partition, it associates a **prediction** $w_1, w_2 \ldots w_k$.

Therefore, a decision tree can be seen as a mapping function:

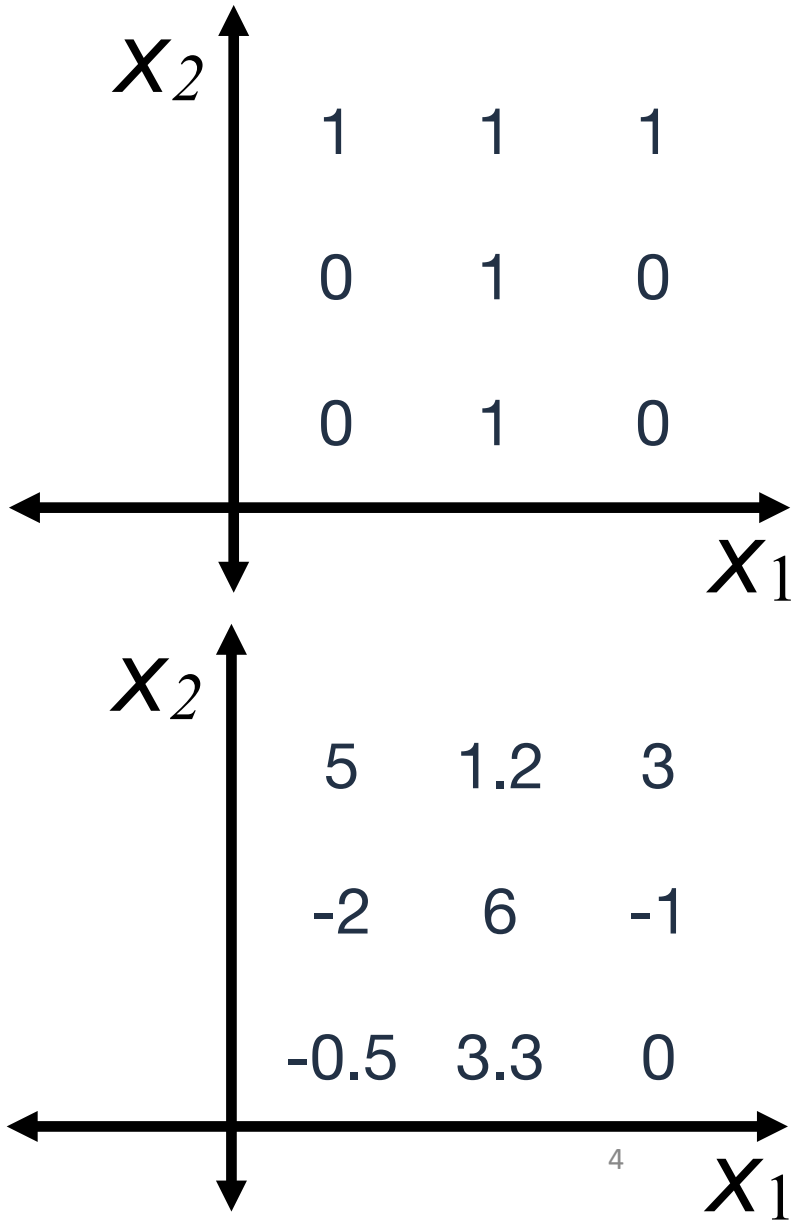
$$f(x) = \sum_{j=1}^{k} w_j \mathbf{1}[x \in R_j]$$

Decision Tree - formalism

How do we calculate the predictions $w_1, w_2 \ldots w_k$?

Suppose we have $n = 2$ input features, $\ell = 9$ examples

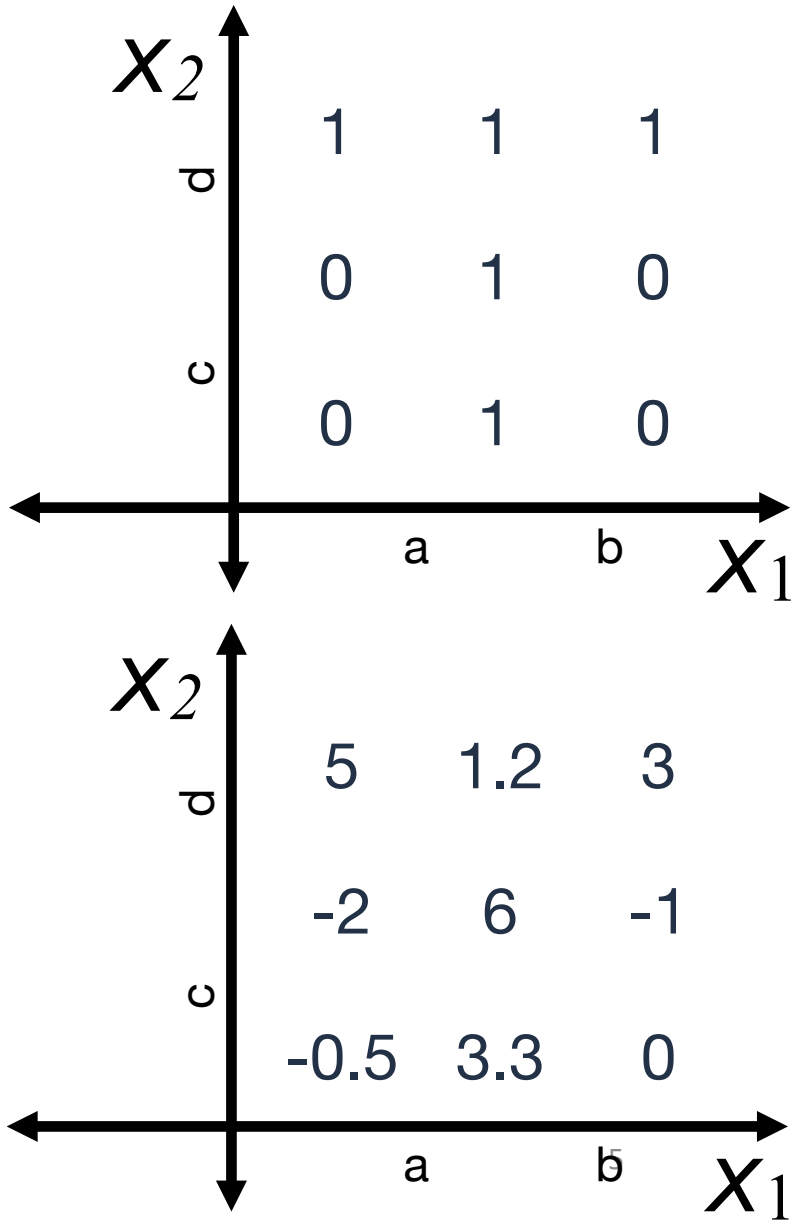
- For a classification problem, the labels can be for example 0 and 1
- For a regression problem, labels will be real numbers



Decision Tree - formalism

How do we calculate the predictions $w_1, w_2 \ldots w_k$?

For **classification**: prediction is the most common class among the training data in a given partition

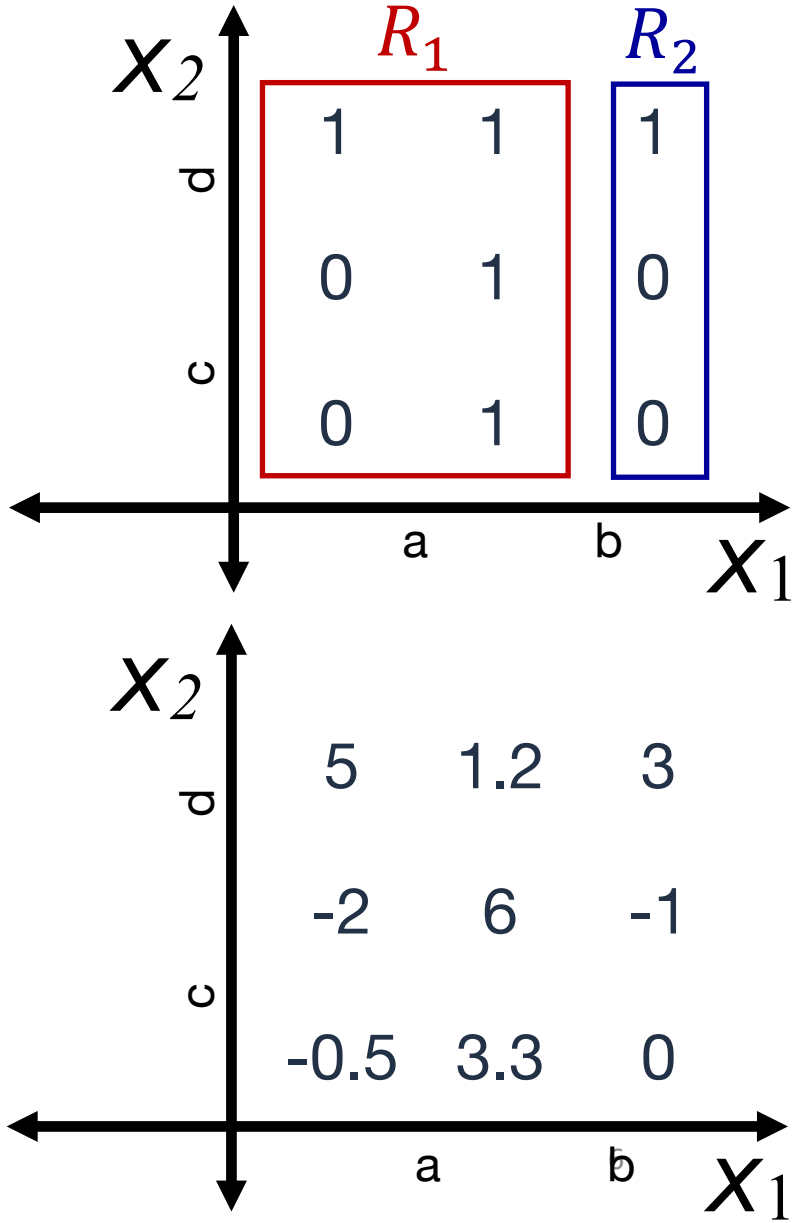


Decision Tree - formalism

How do we calculate the predictions $w_1, w_2 \ldots w_k$?

For **classification**: prediction is the most common class among the training data in a given partition

$$R_1 \rightarrow w_1 = 1, R_2 \rightarrow w_2 = 1$$

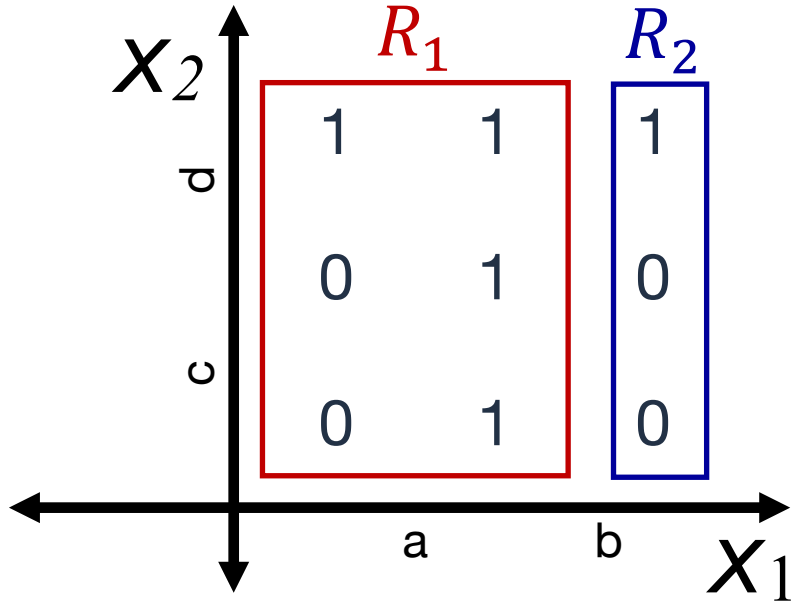


Decision Tree - formalism

How do we calculate the predictions $w_1, w_2 \ldots w_k$?

For **classification**: prediction is the most common class among the training data in a given partition

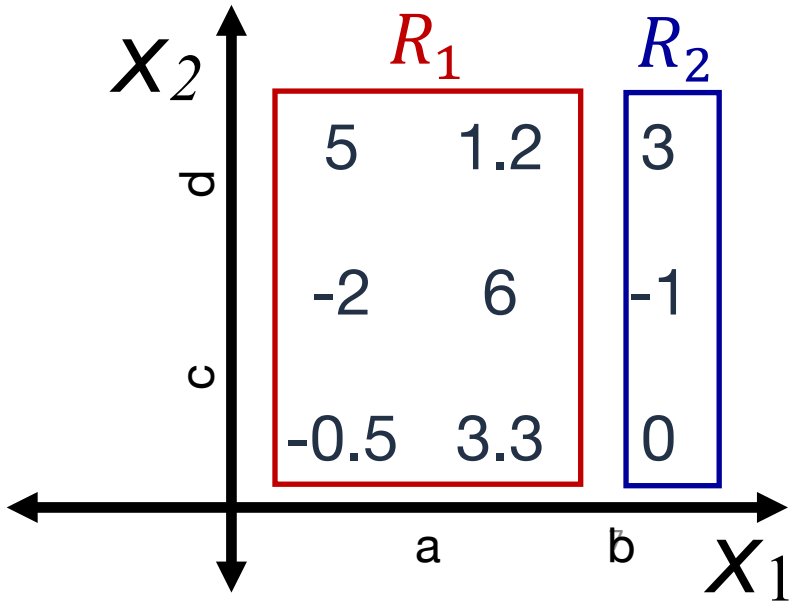
$$R_1 \rightarrow w_1 = 1, R_2 \rightarrow w_2 = 0$$



For **regression**: average value of the labels is a common choice

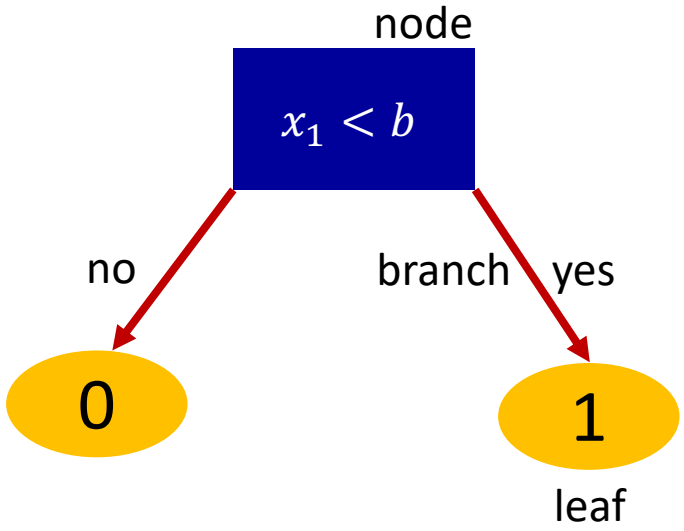
$$R_1 \rightarrow w_1 = 6.5, R_2 \rightarrow w_2 = 0.7$$

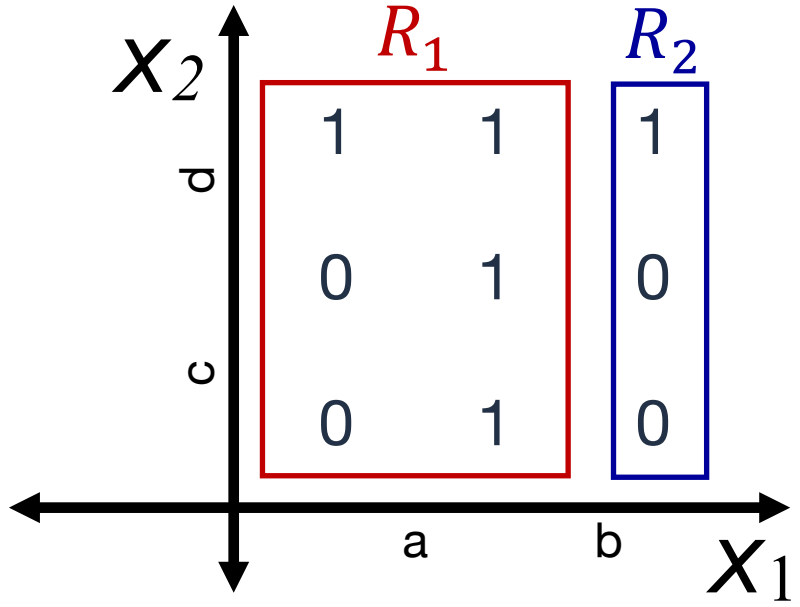
$$w_j = \frac{\sum_{i=1}^{n} y^{(i)} \mathbf{1}[x^{(i)} \in R_j]}{\sum_{i=1}^{n} \mathbf{1}[x^{(i)} \in R_j]}$$



Decision Tree – building process

The partitioning can be conveniently visualised as a decision tree.



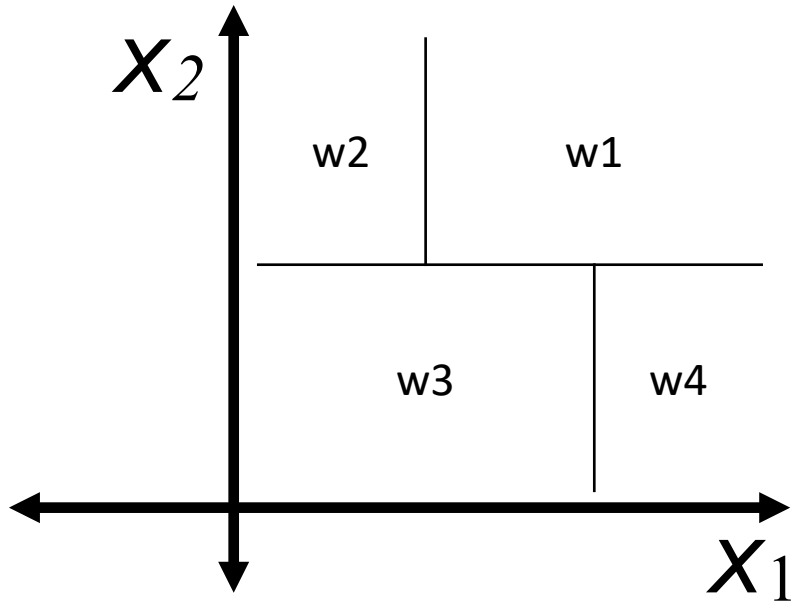


Decision Tree – building process

Recall: familiar pattern

1. Choose how to predict label (given features & parameters)
2. Choose a loss (between predicted & actual label)
3. Choose parameters by trying to minimize the training loss

$$\mathcal{L}(f) = \sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right) = \sum_{j=1}^{k} \sum_{x^{(i)} \in R_j} \ell(y^{(i)}, w_j).$$

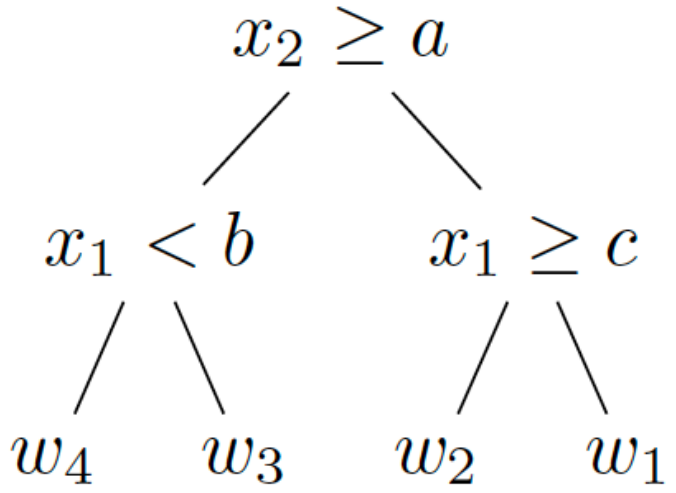


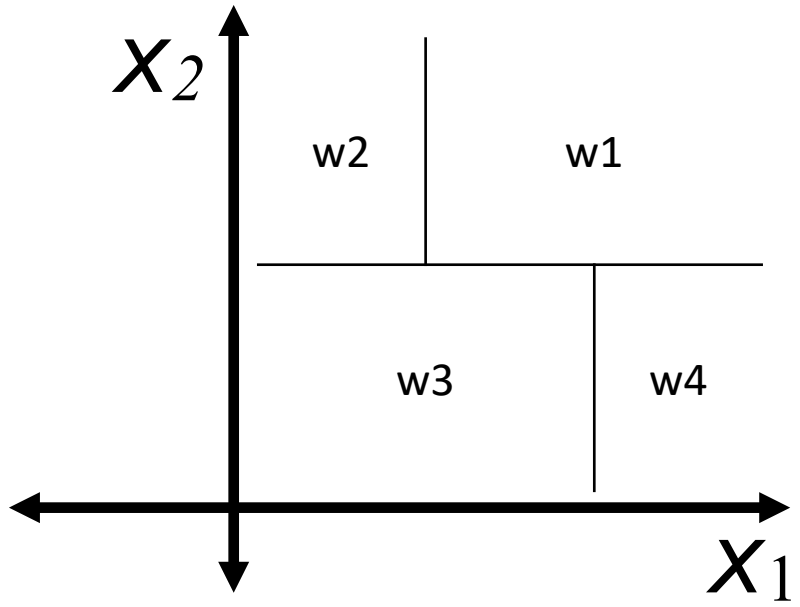
Decision Tree – building process

How many parameters?

For each internal node:
- Split dimension
- Split value
- Child nodes

For each leaf node:
- label



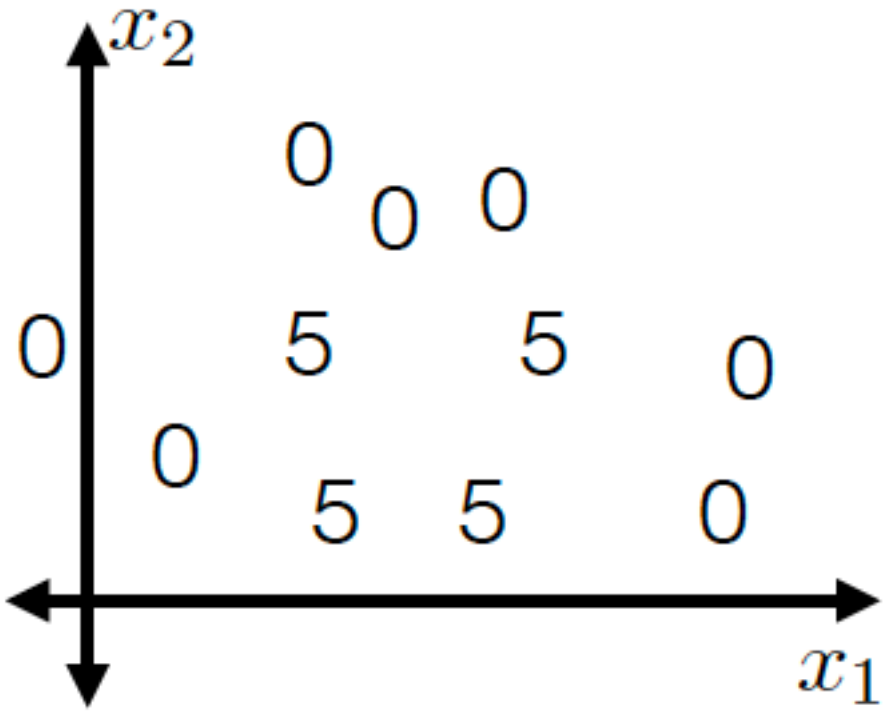


- Parameter space has no fixed dimension
- Can't find a smooth loss function due to the discrete structure of the decision tree

Can't use GD/SGD!

Decision Tree – building process

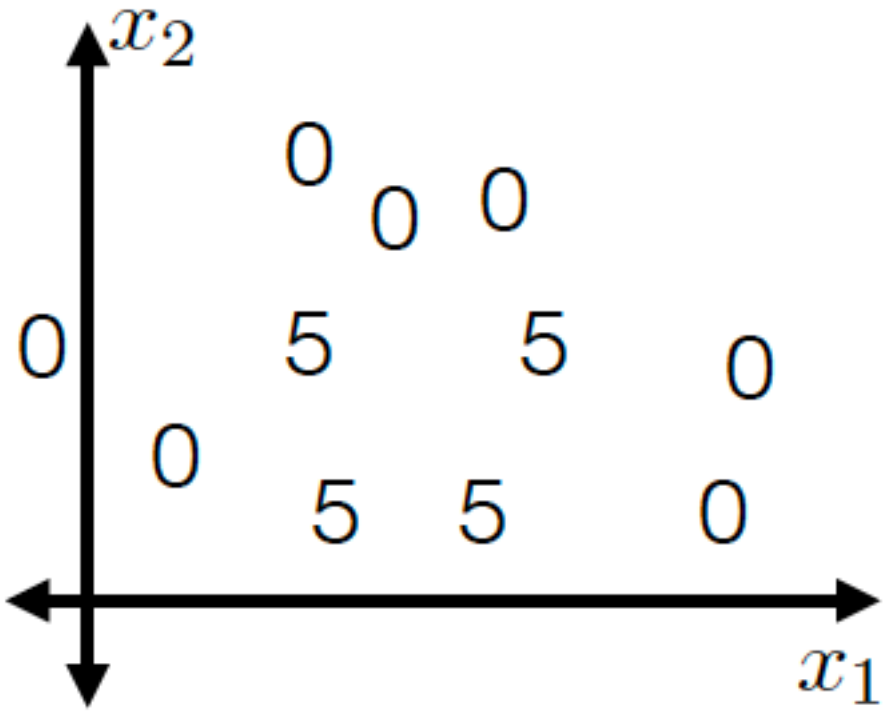
***Heuristic* solution: greedy, top-down, iterative algorithm**



Decision Tree – building process

***Heuristic* solution: greedy, top-down, iterative algorithm**

1. First node → split input space in two partitions, according to some condition $x_j < \theta$ (j ∈ $\{1 \dots n\}$)

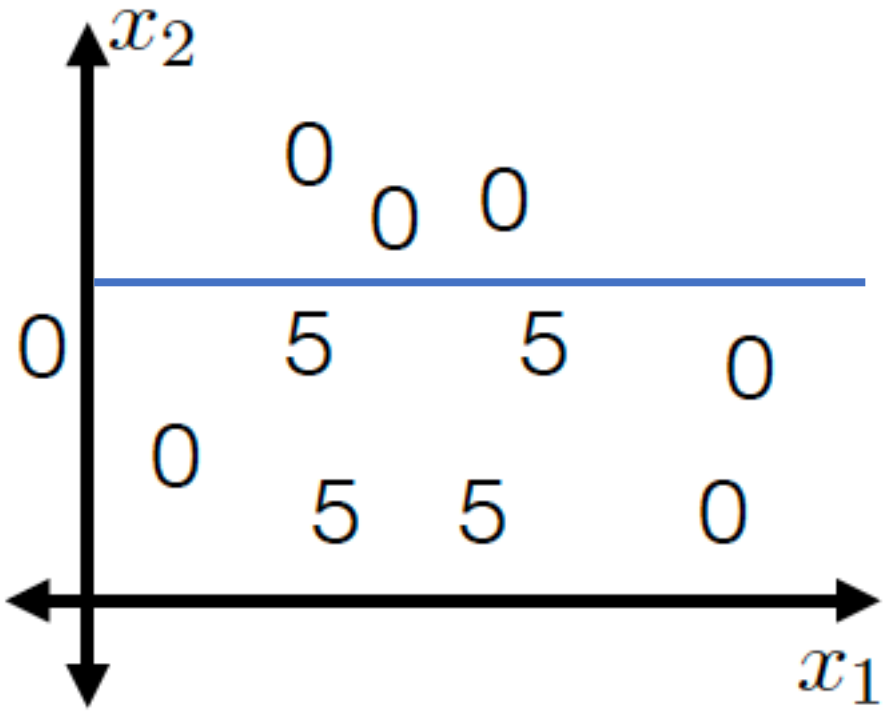


$x_j < \theta$

Decision Tree – building process

***Heuristic* solution: greedy, top-down, iterative algorithm**

1. First node → split input space in two partitions, according to some condition $x_j < \theta$ (j $\in \{1 \dots n\}$)
2. Find the feature j and split value θ that maximise some gain function $\mathrm{G}(\mathrm{j}, \theta)$

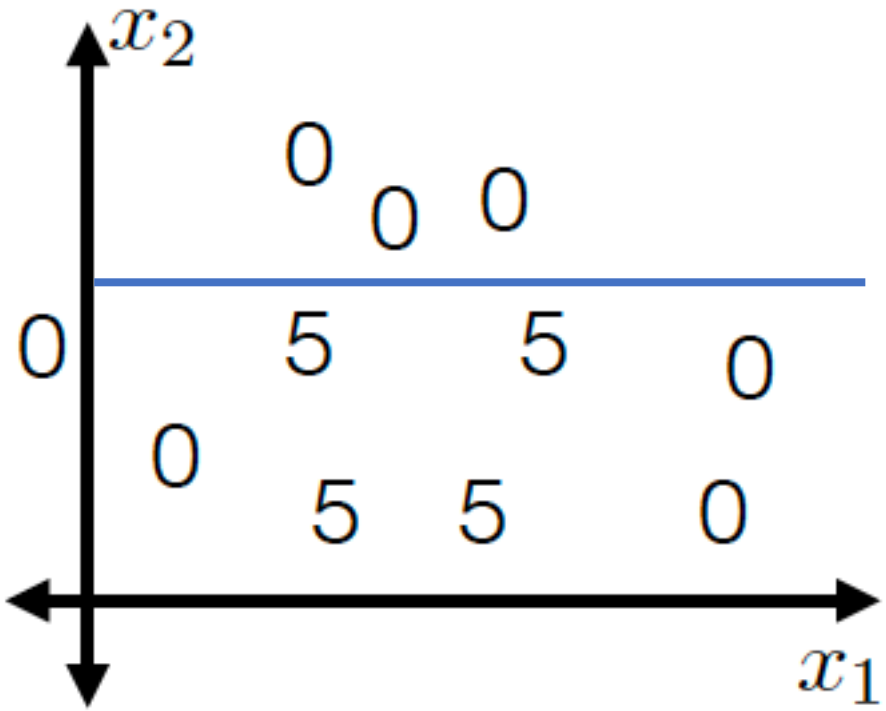


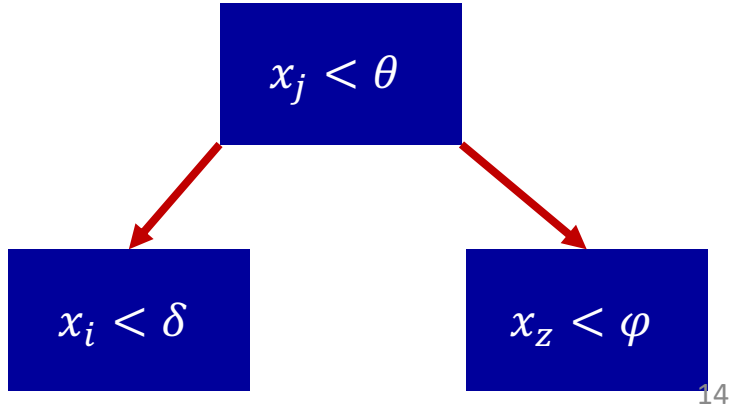
$x_j < \theta$

Decision Tree – building process

***Heuristic* solution: greedy, top-down, iterative algorithm**

1. First node → split input space in two partitions, according to some condition $x_j < \theta$ (j $\in \{1 \dots n\}$)
2. Find the feature j and split value θ that maximise some gain function $\mathrm{G}(\mathrm{j}, \theta)$
3. Decide whether to further split
   - If yes: repeat steps 1..3

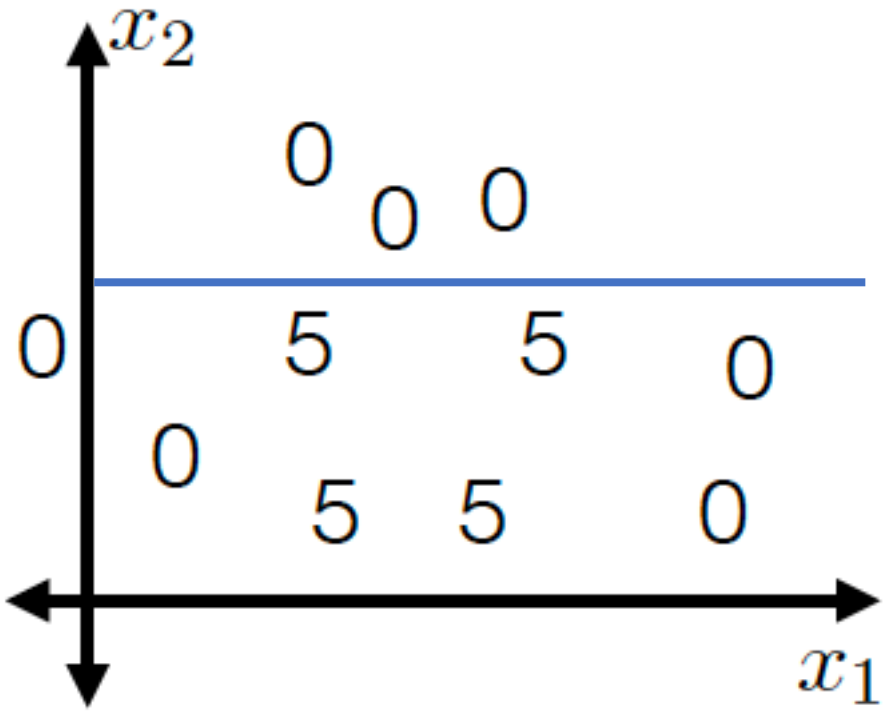


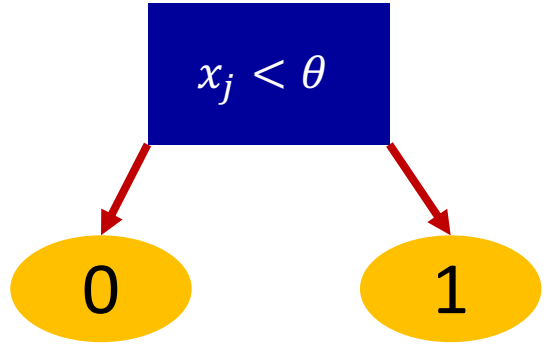


Decision Tree – building process

***Heuristic* solution: greedy, top-down, iterative algorithm**

1. First node → split input space in two partitions, according to some condition $x_j < \theta$ (j $\in \{1 \dots n\}$)
2. Find the feature j and split value θ that maximise some gain function $\mathrm{G}(\mathrm{j}, \theta)$
3. Decide whether to further split
   - If yes: repeat steps 1..3
   - If not: assign values to leaves

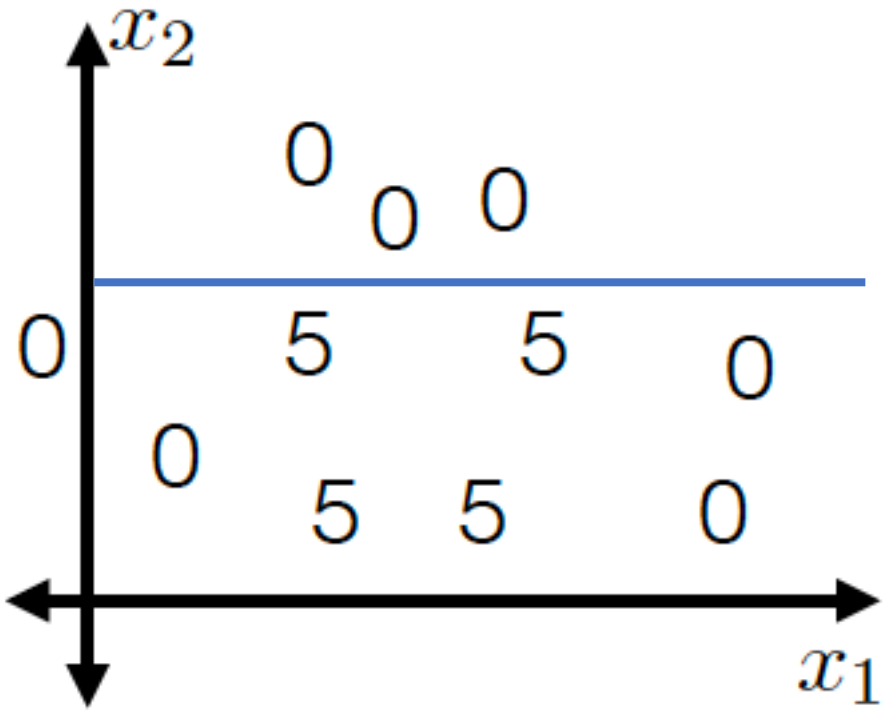




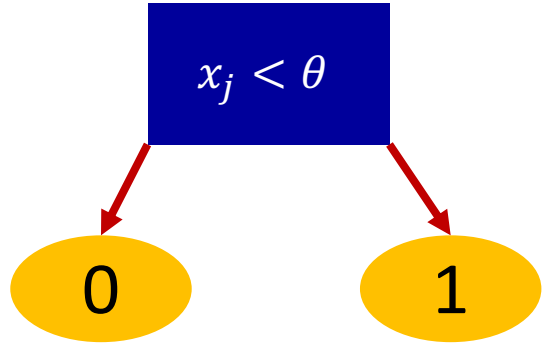
Decision Tree – building process

***Heuristic* solution: greedy, top-down, iterative algorithm**

1. First node → split input space in two partitions, according to some condition $x_j < \theta$ (j $\in \{1 \dots n\}$)
2. Find the feature j and split value θ that maximise some gain function $\mathrm{G}(\mathrm{j}, \theta)$
3. Decide whether to further split
   - If yes: repeat steps 1..3
   - If not: assign values to leaves



Missing items:
- Gain function
- Stop rule



Decision Tree – maximizing gain by minimizing cost

Here **R** and **L** indicate the datapoints associated with the left and right children of the original node after splitting according to $x_j < \theta$

S is the input set of datapoints to which the decision $x_j < \theta$ applies

C is some cost function

$$G(j,\theta) = \underbrace{C(S)}_{\text{Cost before splitting}} - \underbrace{\left[\frac{|L|}{|S|}C(L) + \frac{|R|}{|S|}C(R)\right]}_{\text{Cost after splitting}}$$

Common choices for the cost function:

- Classification → entropy (where C is the set of classes and pc is the fraction of datapoints in S of class C)

$$\sum_{c \in C} -p_c \log p_c$$

- Regression → MSE

$$\frac{1}{|S|}\sum_{x^{(i)} \in R} (y^{(i)} - w)^2$$

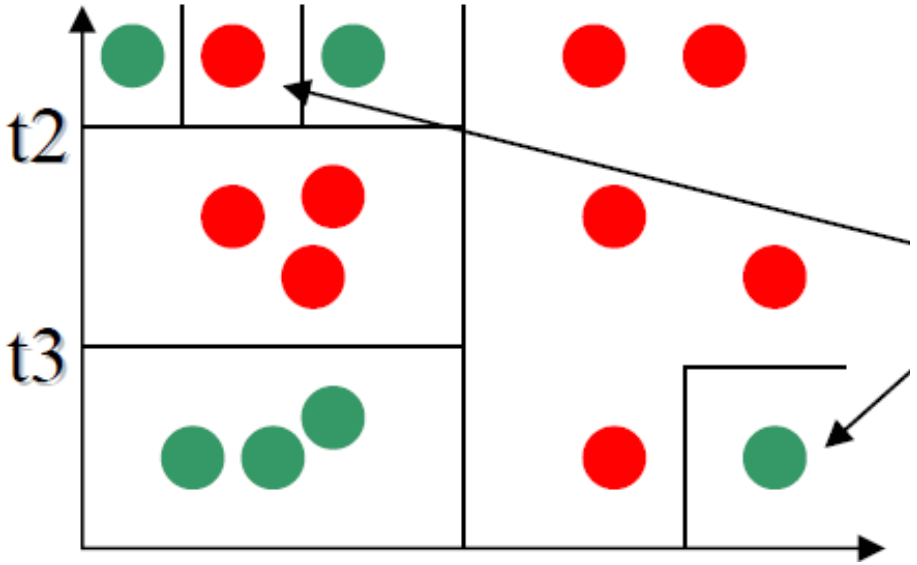
Decision Tree – stopping rule: regularization

A tree of sufficient **depth** can perfectly fit on any training test:

- Overfitting
- Very large model

Multiple choices available:

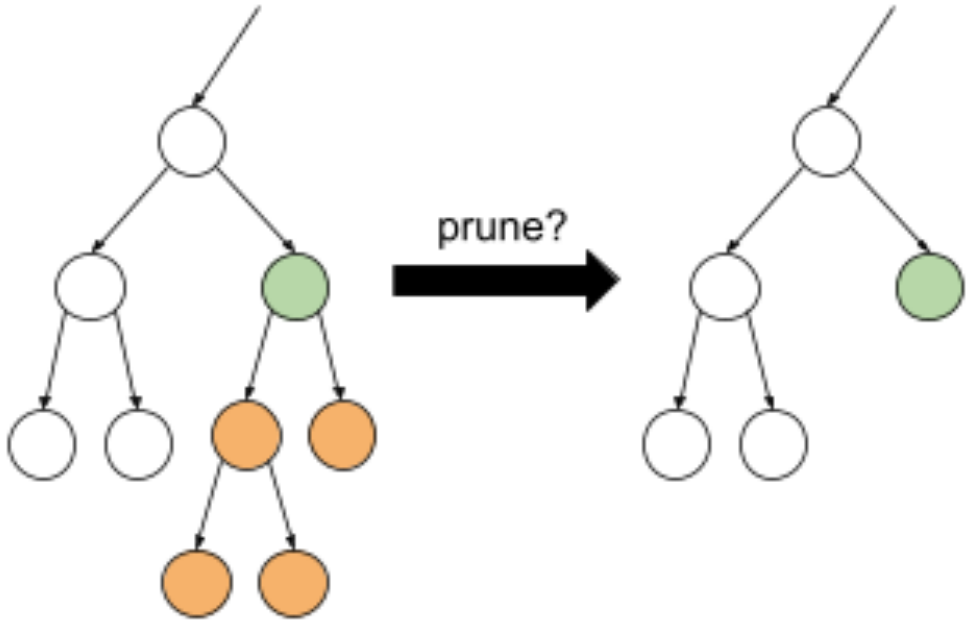
- Maximum tree depth
- Minimum number of objects in leaf
- Maximum number of leafs in tree
- Stop if all objects fall into same leaf
- Stop when improvement gains drop below x%



Decision Tree – pruning

Pruning:

- Let the tree overfit, i.e. reach a 0-loss fit
- Evaluating whether the increased performance associated with a node (or set of nodes) is worth the extra model size and reduced generalization
  - For example, detect over-fitting with k-fold cross-validation
- Remove least important nodes



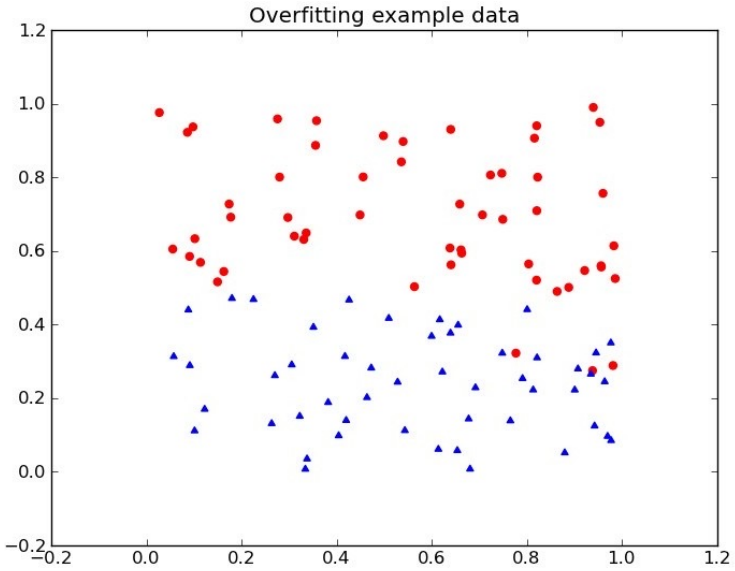
4-fold validation (k=4)

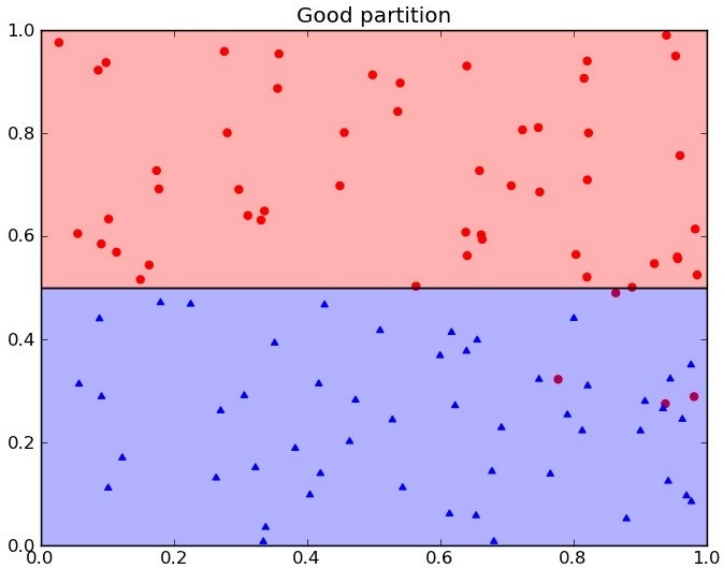
Image credits: https://it.mathworks.com

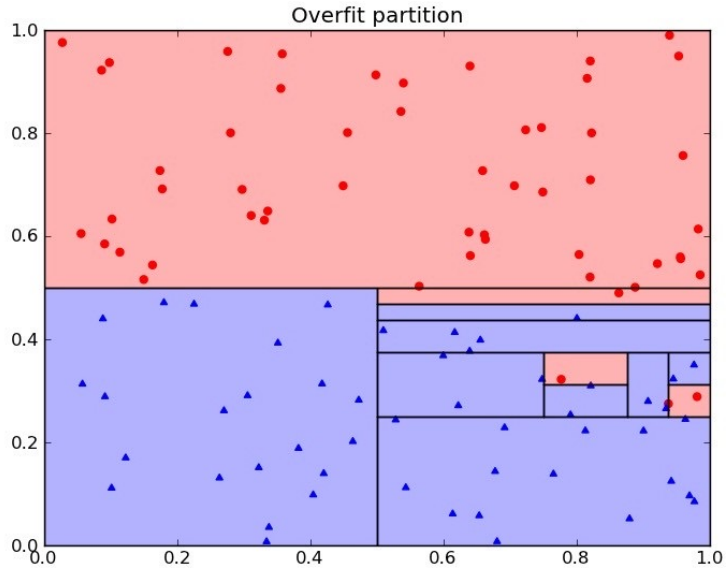
Decision Trees – pro and cons

- ✓ simple to understand and interpret
- ✓ can handle both numerical and categorical variables
- ✓ robust to outliers
- ✓ can handle non-linear relationships between the input and target variables

- ○ prone to overfitting
- ○ unstable around boundaries

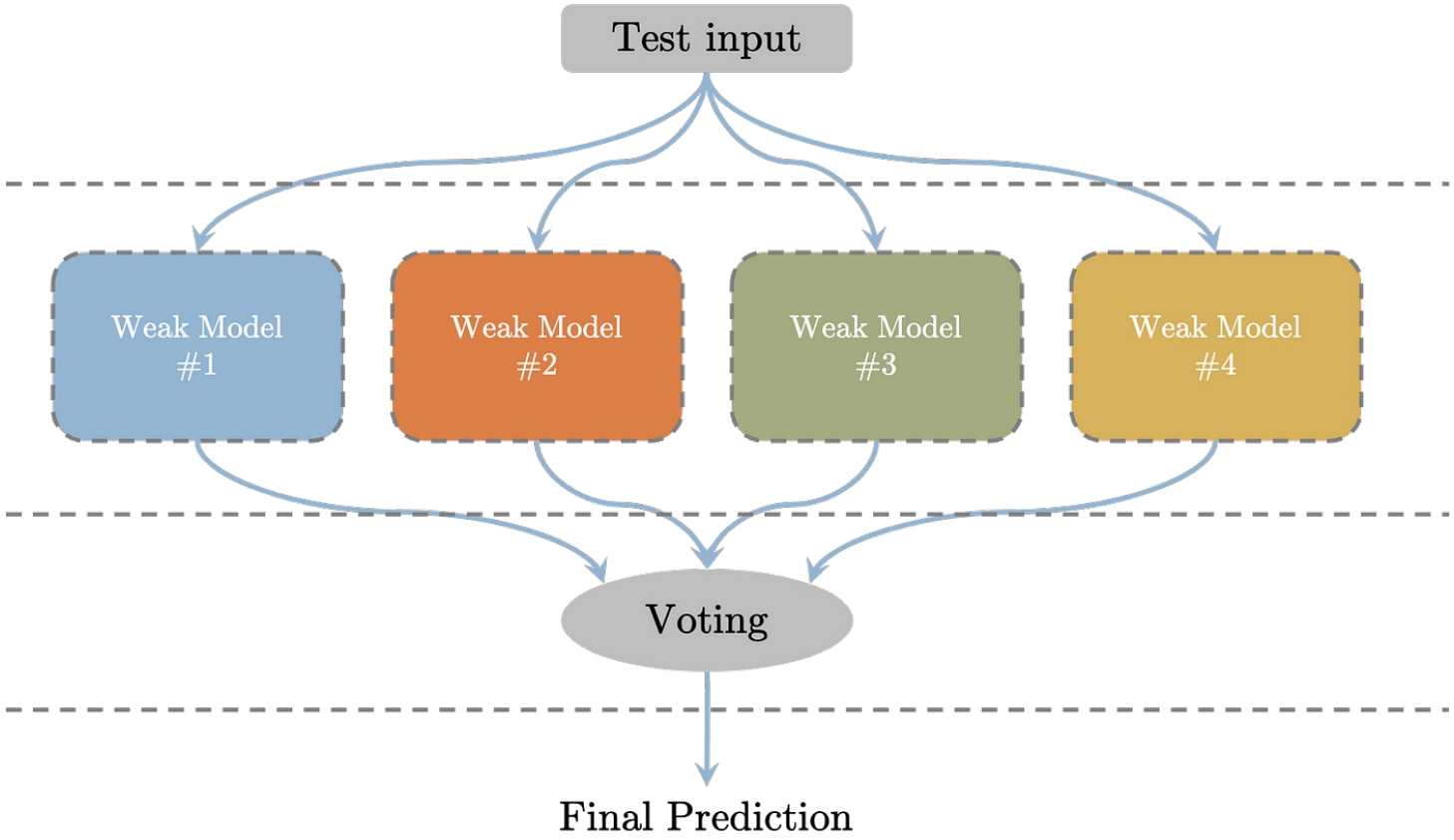






Ensemble Learning

Reduce model instability by averaging predictions over multiple instances of the same (similar) model



Weights

$$f(x) = \sum_{i=1}^{m} \beta_i F_i(x)$$

Ensemble

Individual models

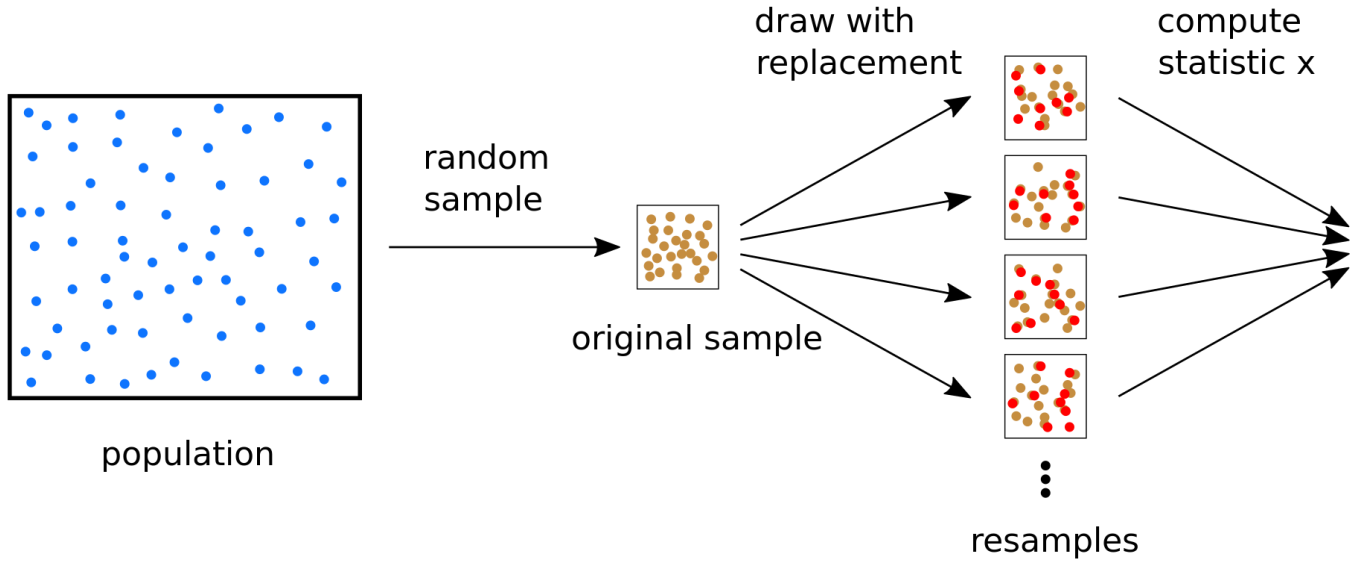
Image credits: https://towardsdatascience.com

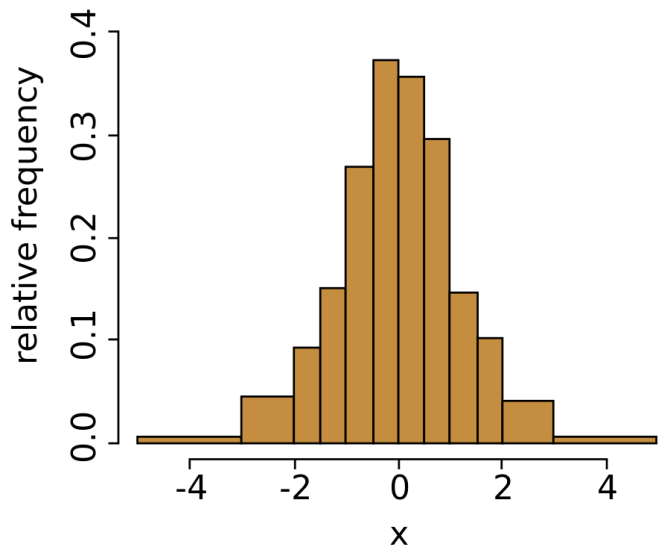
Most common ensembling methos for decision trees

- Bagging: bootstrap aggregating
- Random Forest
- Boosting

Bagging: bootstrap aggregating

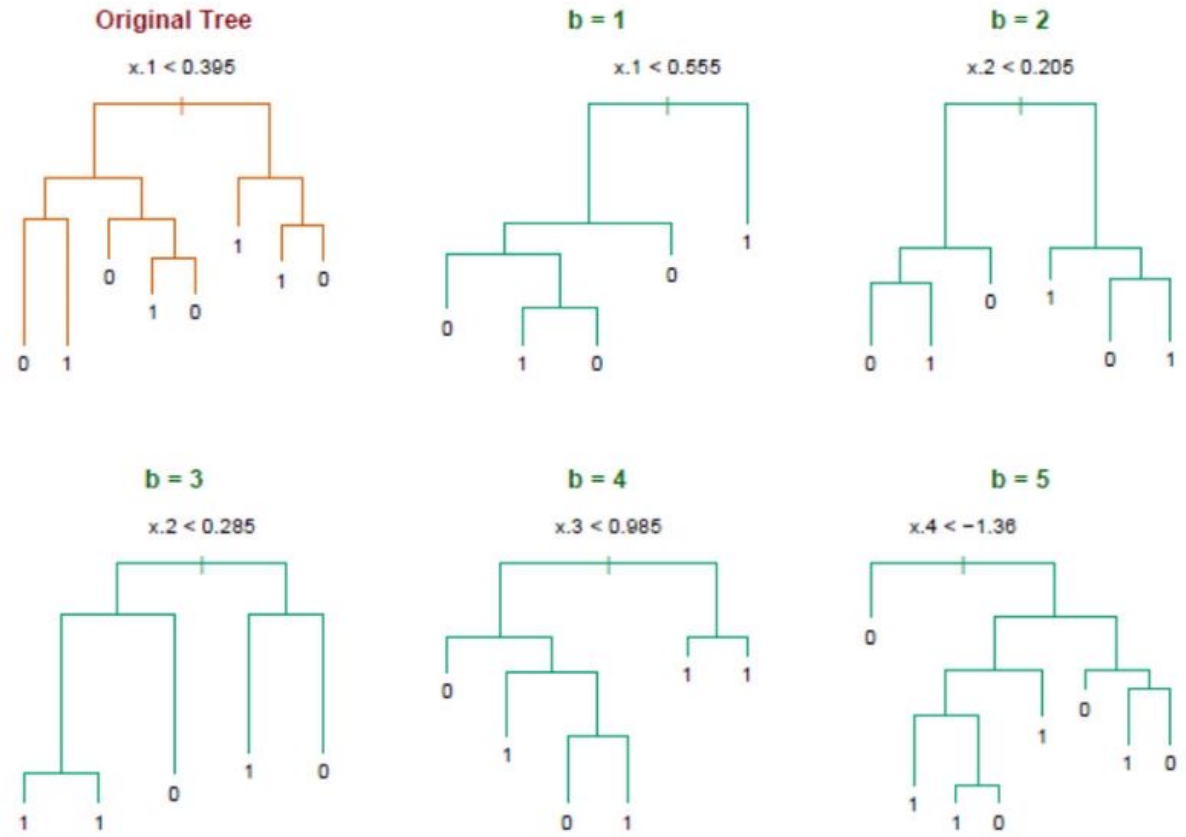
- **Bootstrapping (statistics):** generate a "new" dataset by sampling uniformly and with replacement from our original one.





Bagging: bootstrap aggregating

- **Ensemble learning idea:**
  - Generate N bootstrapped datasets
  - Learn N hypotheses
  - Average

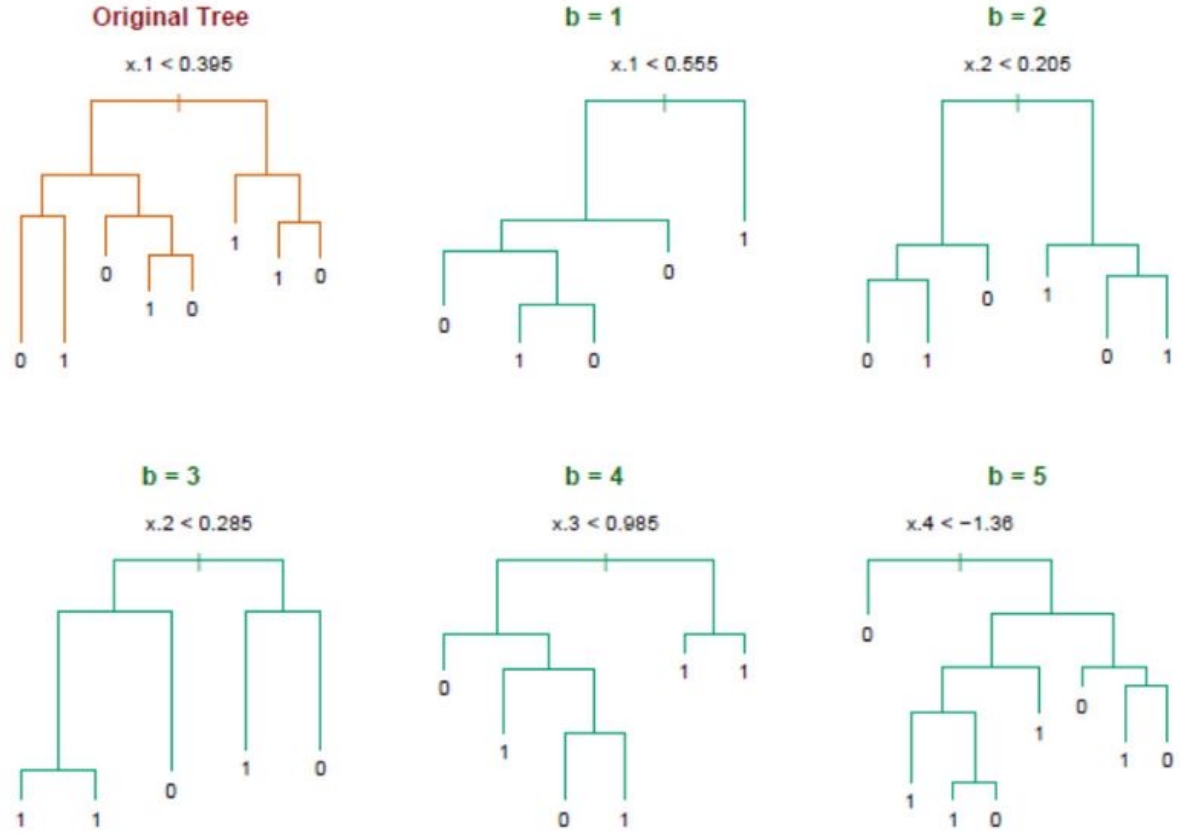


Hastie et al.,"The Elements of Statistical Learning: Data Mining, Inference, and Prediction", Springer (2009)

The Random Forest algorithm

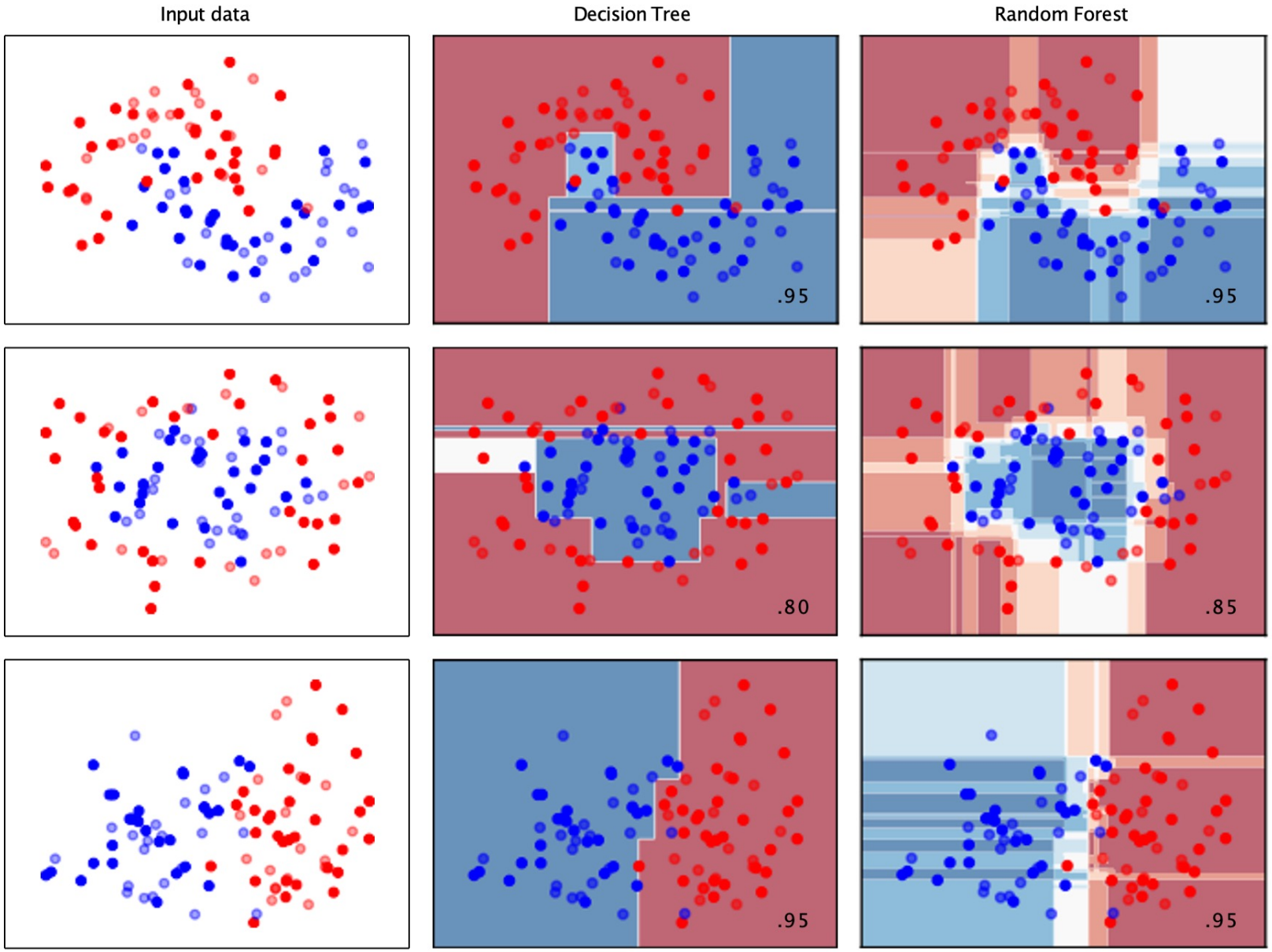
Random Forest is an example of bagging applied to decision trees, **with some extra randomness** to further decrease the correlation between individual models.

- For each decision tree
  - For each node:
    - Consider only a subset of the available features (randomly chosen)
    - Pick the best split dimension and split value among the features in the subset



Hastie et al.,"The Elements of Statistical Learning: Data Mining, Inference, and Prediction", Springer (2009)

The Random Forest algorithm

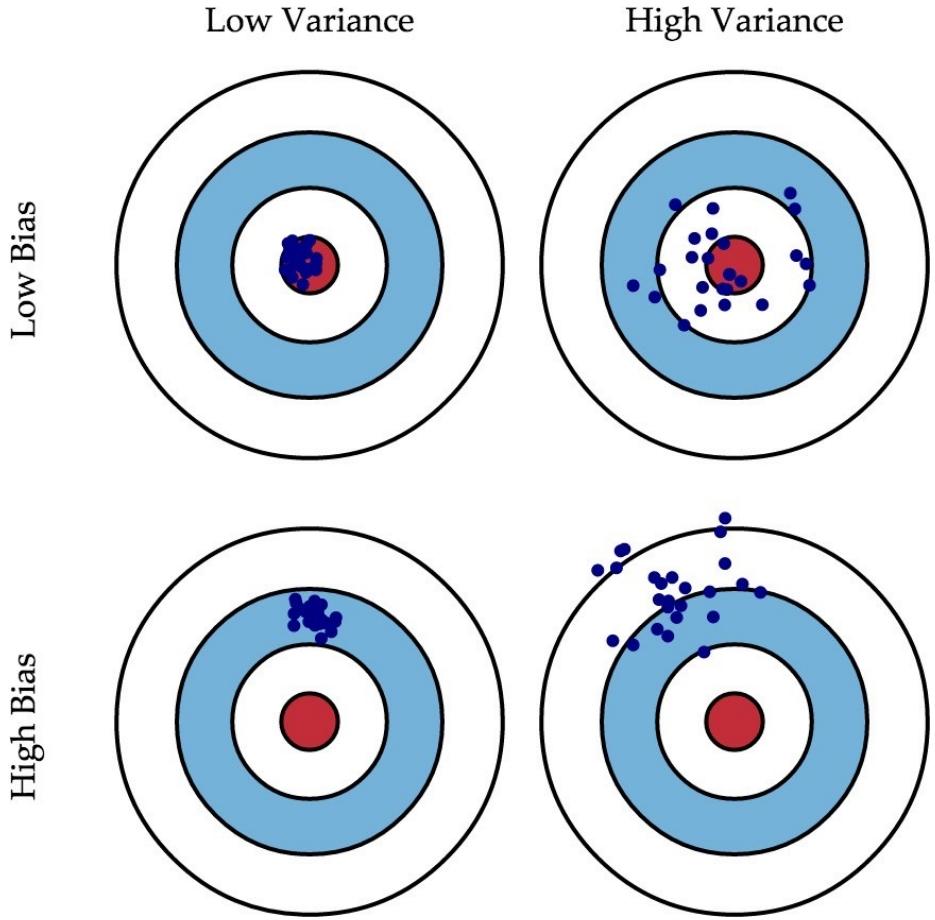


Remember: bias-variance decomposition

$$Q(\mu) = \underbrace{\mathbb{E}_{x,y}\left[\left(y - \mathbb{E}[y \mid x]\right)^2\right]}_{\text{noise}} +$$

$$+ \underbrace{\mathbb{E}_x\left[\left(\mathbb{E}_{X^\ell}\left[\mu(X^\ell)\right] - \mathbb{E}[y \mid x]\right)^2\right]}_{\text{bias}} +$$

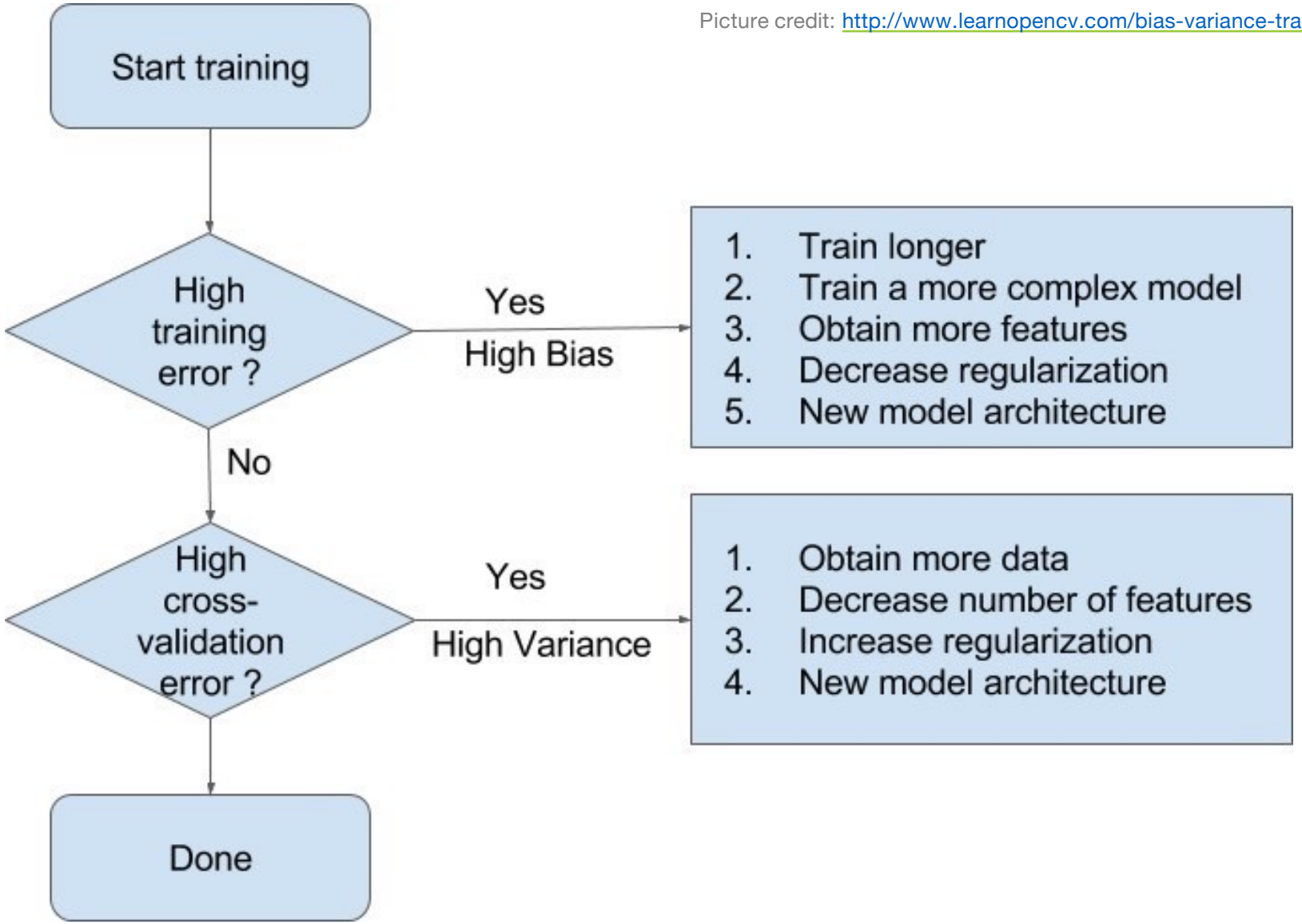
$$+ \underbrace{\mathbb{E}_x\left[\mathbb{E}_{X^\ell}\left[\left(\mu(X^\ell) - \mathbb{E}_{X^\ell}\left[\mu(X^\ell)\right]\right)^2\right]\right]}_{\text{variance}}$$



- Bias: not made any worse by bagging multiple hypotheses
- Variance: N times lower for uncorrelated hypotheses, yet not as much an improvement for highly correlated

Remember: bias-variance decomposition

Picture credit: http://www.learnopencv.com/bias-variance-tradeoff-in-machine-learni3n1g



Check-point

- **Bootstrapping:** a general statistical technique for computing sample functionals (and their variance)
- **Bagging:** meta-learner over arbitrary weak algorithms via bootstrap aggregation
- **The Random Forest algorithm:** bagging over decision trees
- **Bias/Variance trade-off:** important concept that allows to understand and improve model performance

Boosting algorithms

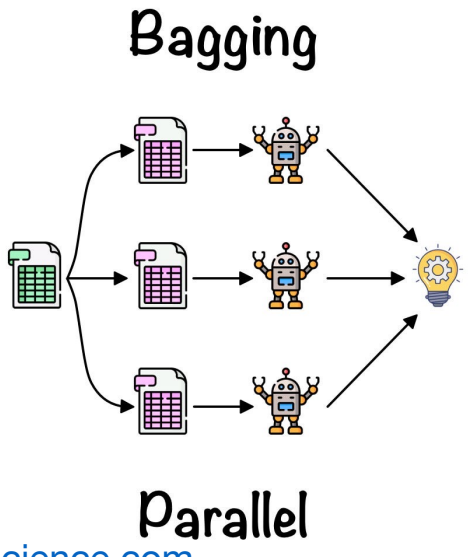
Weights

$$f(x) = \sum_{i=1}^{m} \beta_i F_i(x)$$

Ensemble

Individual models

Idea behind boosting: instead of training individual models in parallel, we train model $F_{i+1}(x)$ based on the performance of the ensemble built so far $f_i(x)$



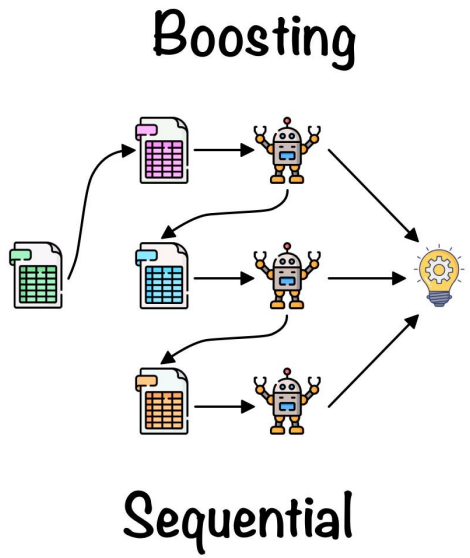


Image credits: https://towardsdatascience.com

Boosting with forward stagewise additive modelling

At iteration i of our ensemble creation, we create the model F_i. To do so, we compute

$$\beta_i, \theta_i = \arg\min_{\beta,\theta} \mathcal{L}(f_{i-1} + \beta F_i) = \arg\min_{\beta,\theta} \sum_{j=1}^{n} \ell\left(y^{(j)}, \left(f_{i-1}(x^{(j)}) + \beta F_i(x^{(j)}; \theta)\right)\right),$$

where θ represents the parameters of the new model F_i. We then set $F_i(x) := F_i(x; \theta_i)$ and afterwards set

$$f_i(x) := f_{i-1}(x) + \beta_i F_i(x)$$

to define our new ensemble model.

http://cs229.stanford.edu/

Gradient boosting

Idea: learn the entire function F_i → optimization in the form of (functional) gradient descent.

$$F_i = \arg\min_F \mathcal{L}(f_{i-1} + F) = \arg\min_F \sum_{j=1}^{n} \ell\left(y^{(j)}, \left(f_{i-1}(x^{(j)}) + F(x^{(j)})\right)\right)$$

At iteration i, our (ensemble) model output on the training datapoints is a vector:

$$f = \left(f_{i-1}(x^{(1)}), \ldots, f_{i-1}(x^{(n)})\right)$$

The gradient of our loss function with respect to f is:

$$g = \left(\frac{\partial \ell(f_{i-1}(x^{(1)}), y^{(1)})}{\partial f_{i-1}(x^{(1)})}, \ldots, \frac{\partial \ell(f_{i-1}(x^{(n)}), y^{(n)})}{\partial f_{i-1}(x^{(n)})}\right)$$

http://cs229.stanford.edu/

Gradient boosting

At each step, fit a weak learner* to our gradient step via a supervised learning task

$$F_i := \arg\min_F \sum_{j=1}^{n} \left(-g_j - F(x^{(j)}) \right)^2,$$

And define an «update rule» for the ensemble model

$$f_i(x) := f_{i-1}(x) + \alpha F_i(x)$$

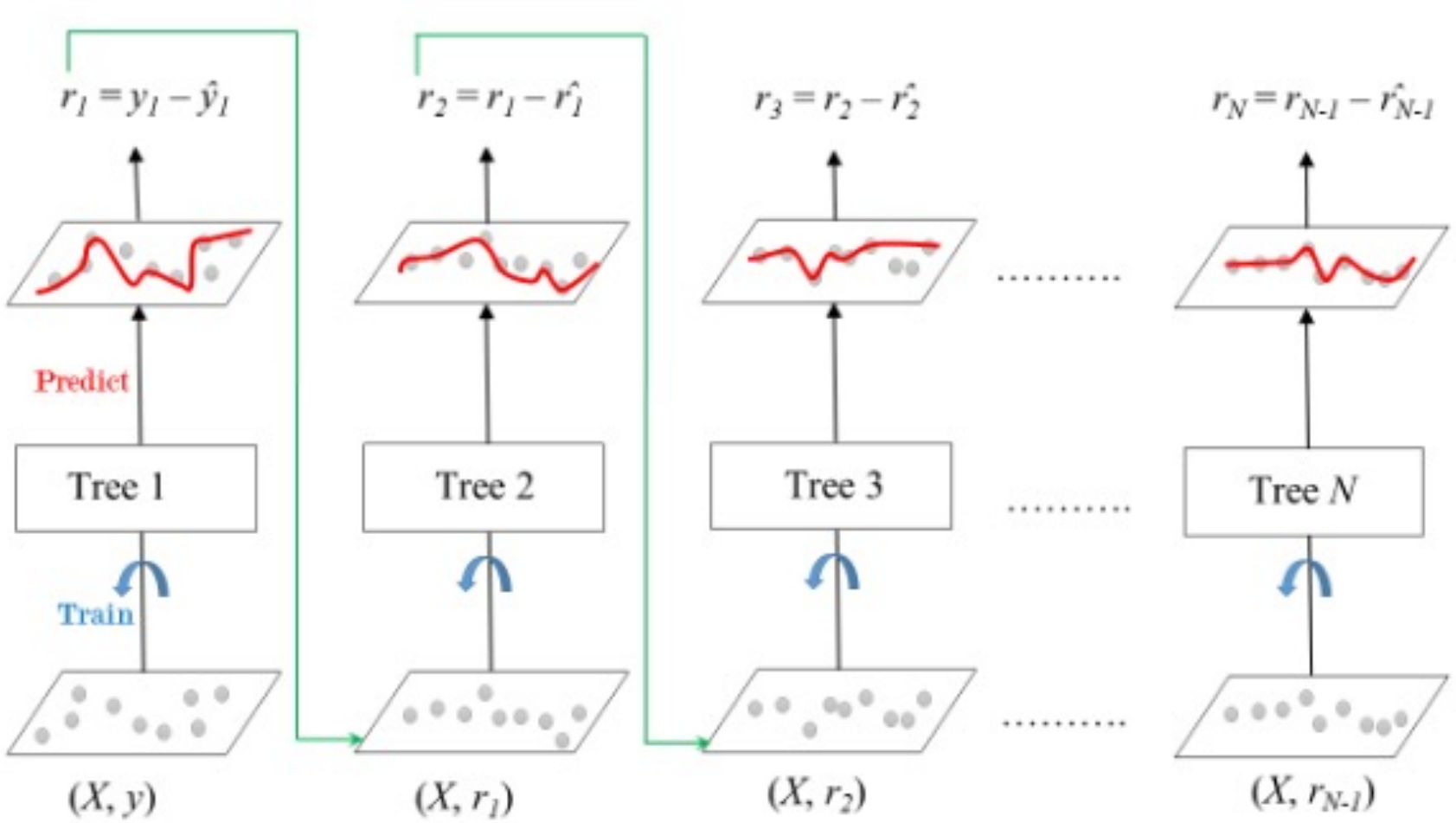
Boosting can be implemented as a function-level gradient descent optimisation problem!

Note: we are fitting a model to the gradient of our loss with respect to our model's current predictions → each successive model is tailored to weaknesses in the current ensemble.

* Models that perform poorly in terms of accurancy, i.e. just above chance.

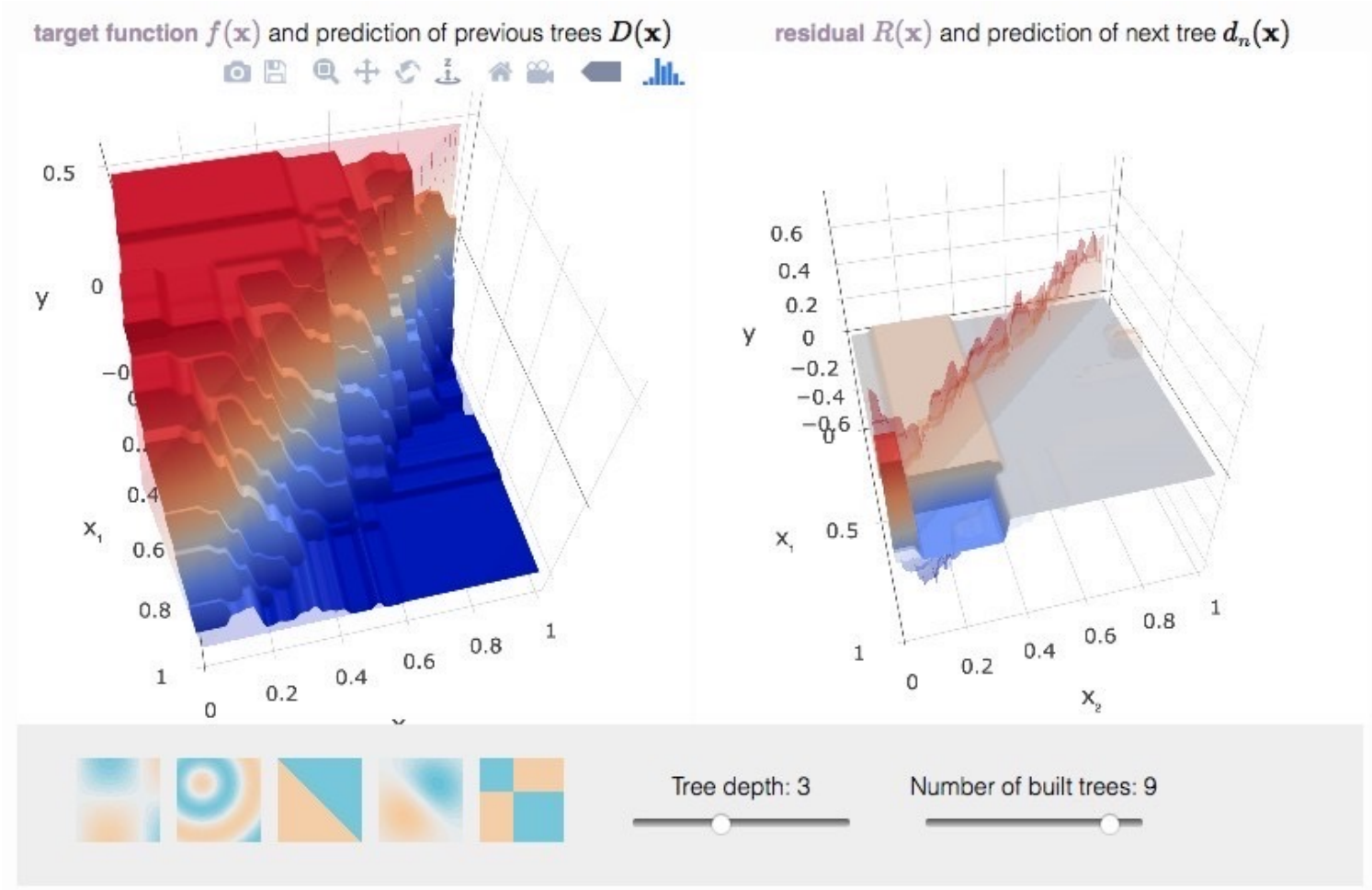
http://cs229.stanford.edu/

Gradient boosting for regression



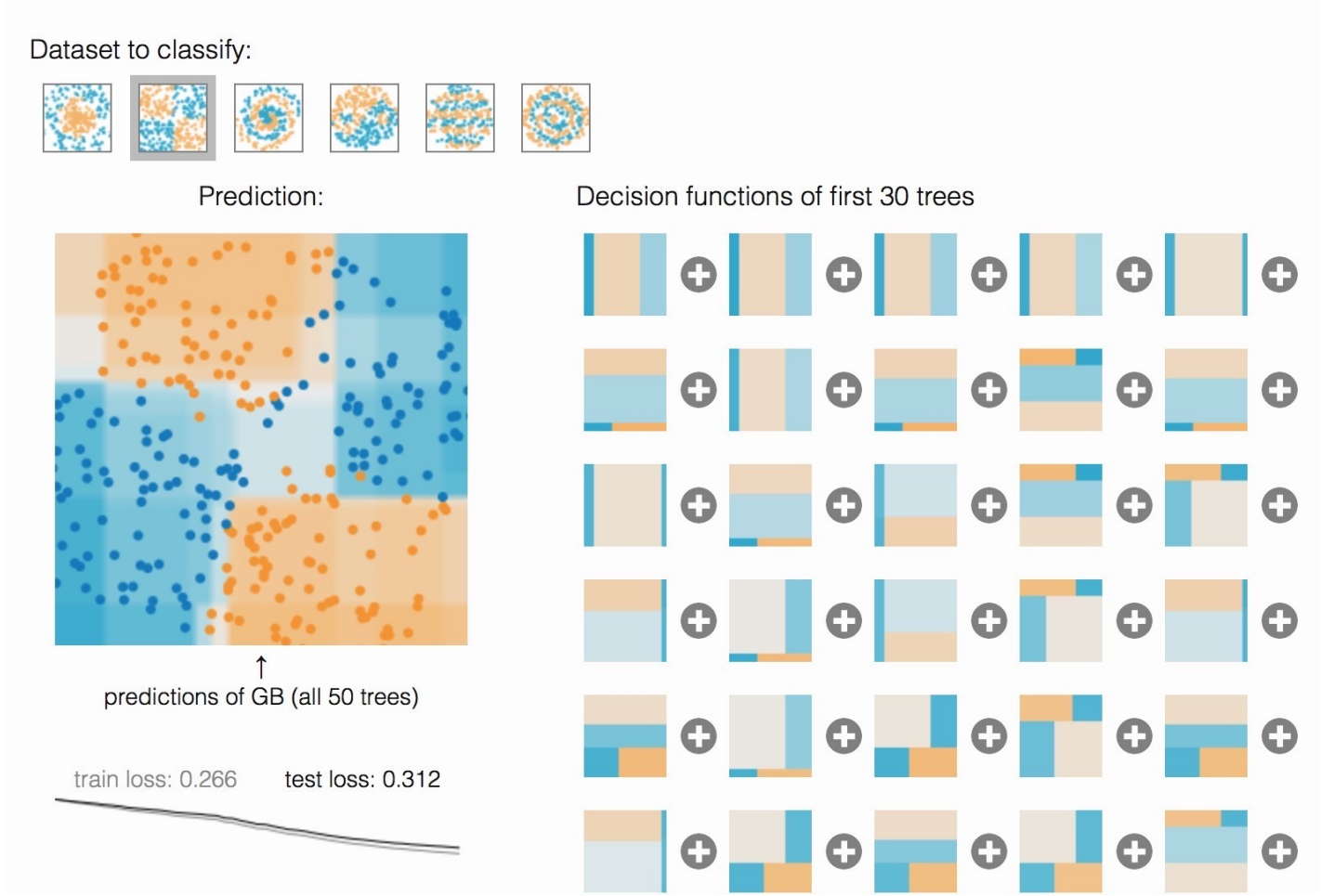
Gradient boosting for regression: interactive demo

http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html



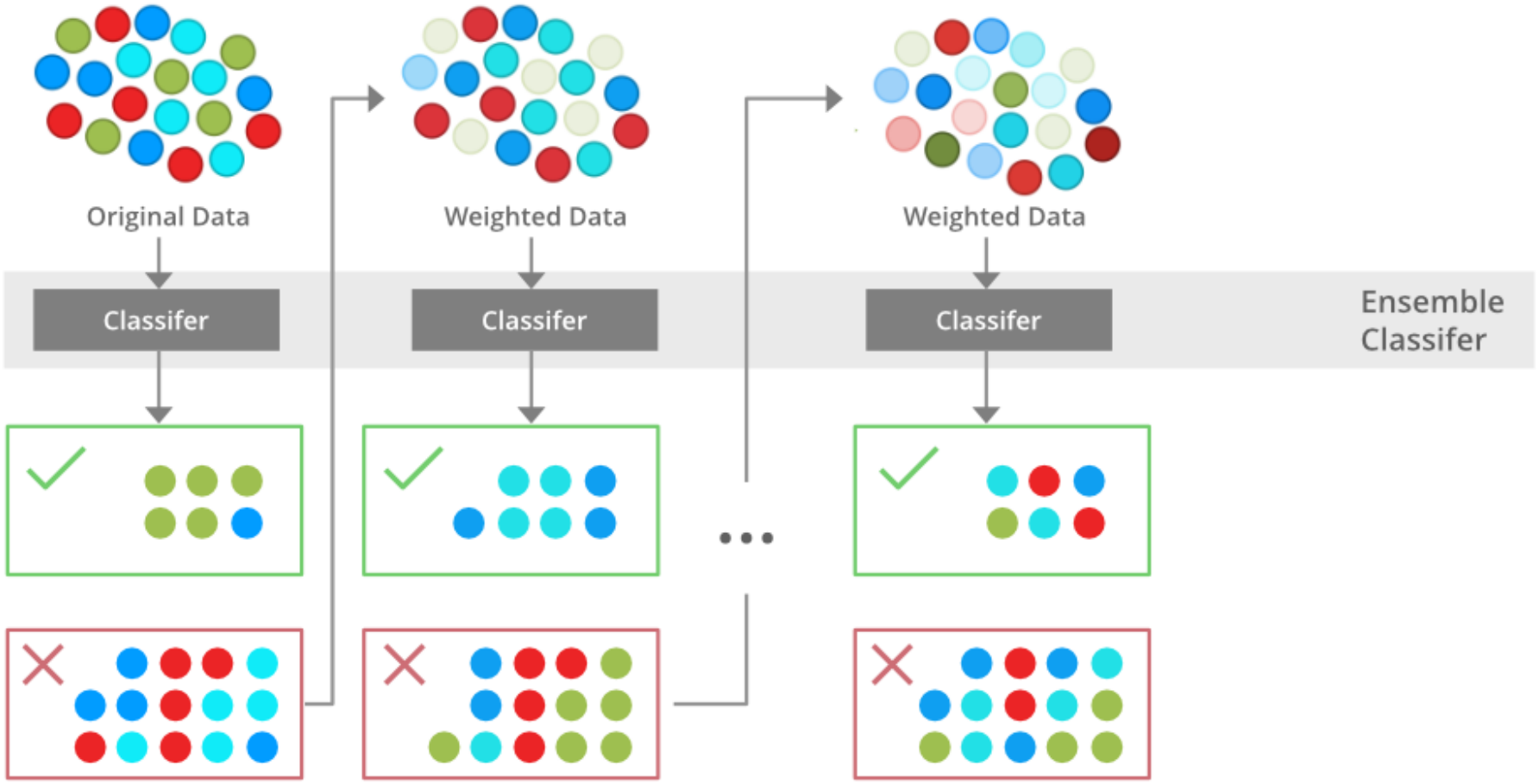
Gradient boosting for regression: interactive demo

http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html



AdaBoost (adaptive boosting) for a classification problem

Idea: perform ensemble training re-weighting the datapoints at each iteration



AdaBoost (adaptive boosting) for a classification problem

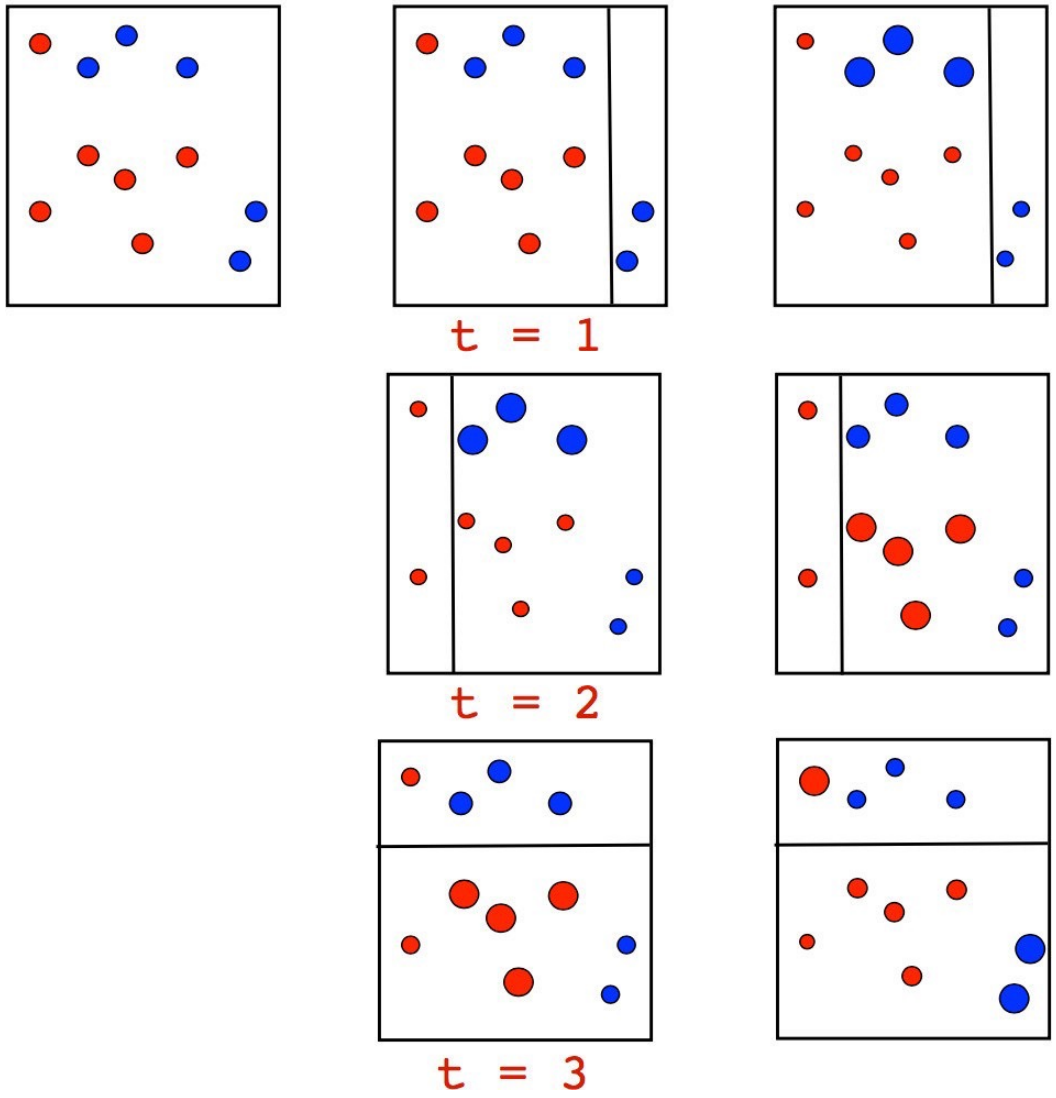
Idea: perform ensemble training re-weighting the datapoints at each iteration

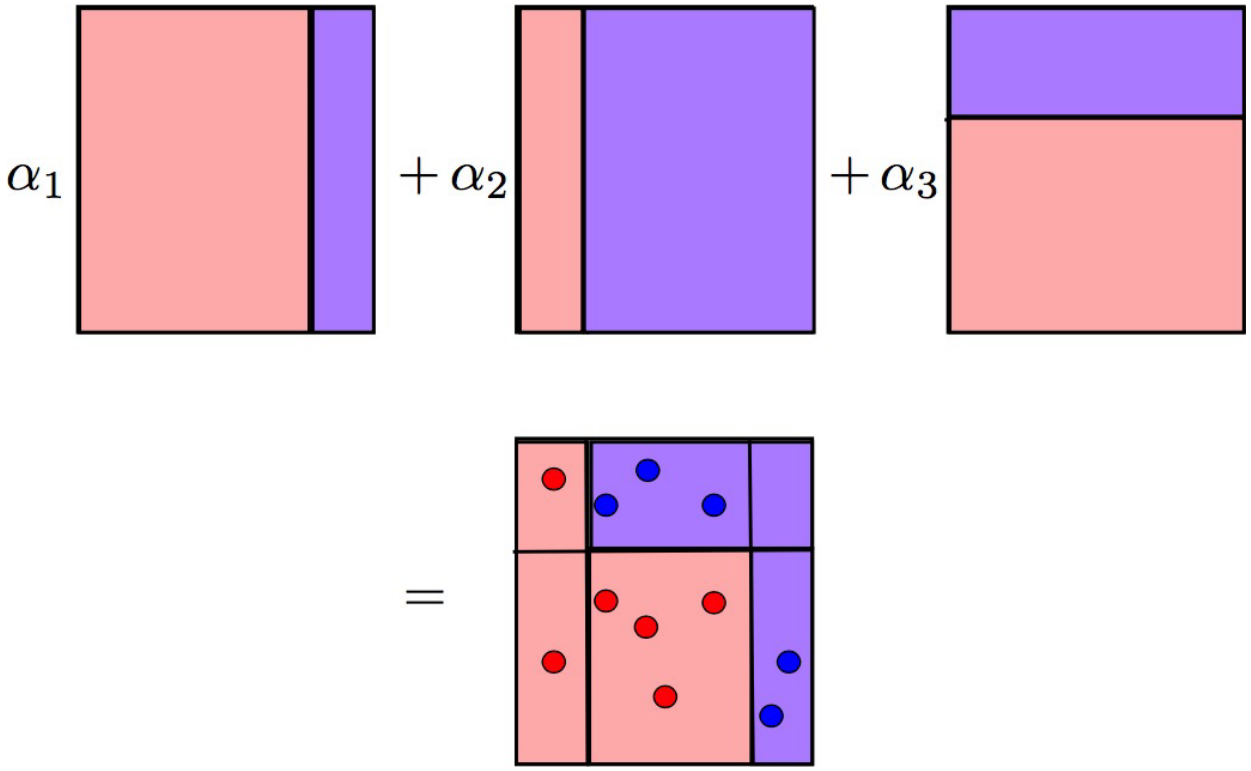
- Input space $\mathbf{X} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, with $y \in \{-1, 1\}$
- Weak learner* $\mathrm{G}(\boldsymbol{x}, y)$ (e.g. decision tree)

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

https://hastie.su.domains/Papers/ESLII.pdf

AdaBoost (adaptive boosting) for a classification problem





Picture credit: https://habrahabr.ru/company/ods/blog/327250

Extreme Gradient Boosting

The main differences between XGBoost and gradient tree boosting are that

- We optimize the loss function with a regularization term included:

$$\mathcal{L}(f) = \left[\sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)\right] + \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} w_j^2.$$

  J is the number of leaves, w_j is the prediction value associated with the jth leaf in the model, and γ, λ are scalar-valued hyperparameters.

- We use a subset of features when splitting tree nodes like in random forests.

- We use a second-order approximation rather than just a gradient, i.e., the Hessian of the loss with respect to our current predictions makes an appearance in our boosting algorithm along with the gradient.

http://cs229.stanford.edu/

Summary

- Boosting: a general meta-algorithm aimed at composing a strong hypothesis from multiple weak hypotheses
- Boosting can be applied for arbitrary losses, arbitrary problems (regression, classification) and over arbitrary weak learners

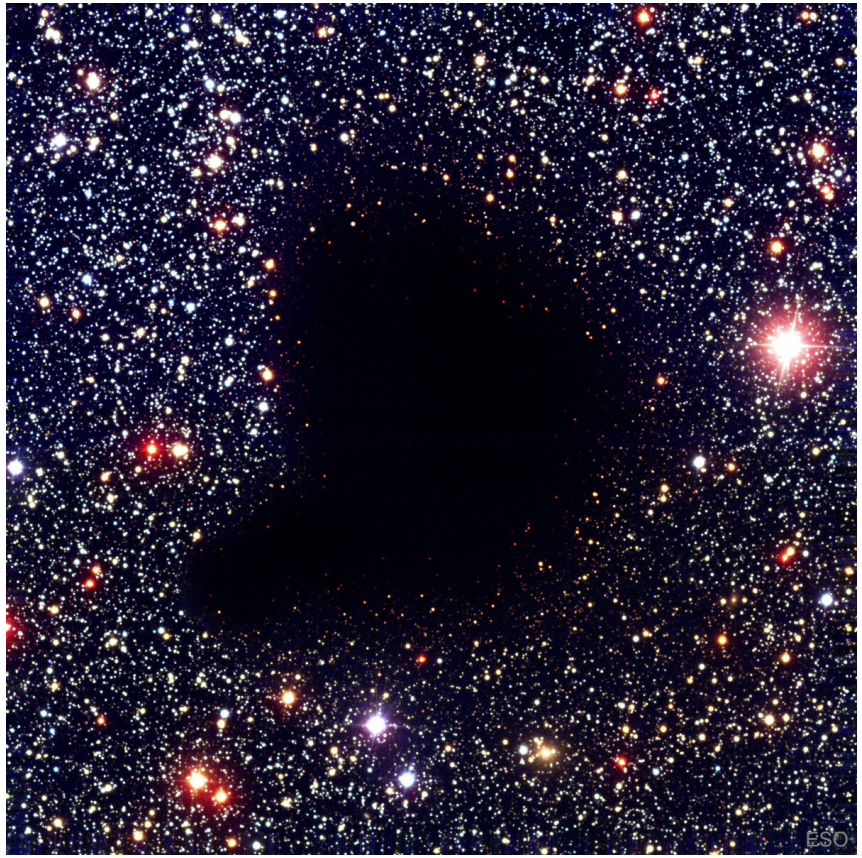
- The Gradient Boosting Machine: a general approach to boosting adding weak learners that approximate gradient of the loss function

- AdaBoost: gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners

- XGBoost: gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion

Hands-on

HTRU2 is **a data set** which describes a sample of pulsar candidates collected during the High Time Resolution Universe Survey (South)



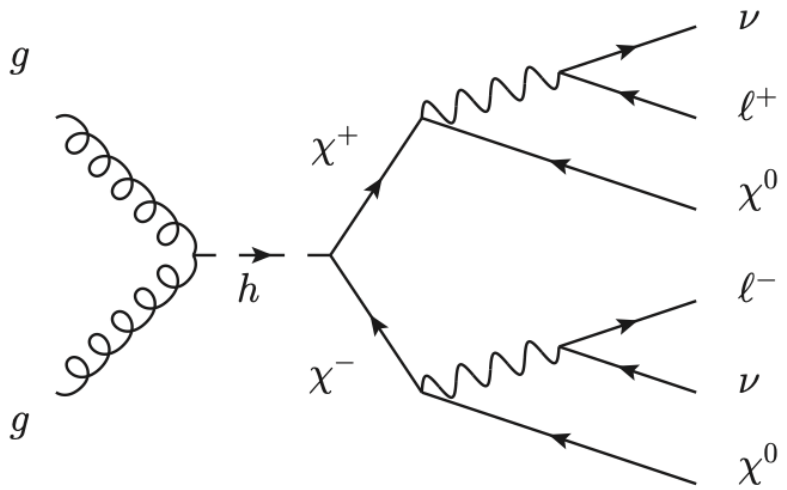
https://archive.ics.uci.edu/dataset/372/htru2

Random Forest for classification using scikit-learn

https://github.com/fsimone91/course_ml4hep/tree/2024/notebooks/2024/4-RandomForest.ipynb

Hands-on

The **Supersymmetry (SUSY)** Dataset supports binary classification of particle physics collision events based on collected features of the collisions.



Baldi et al. Nature Communication 2015 and Arxiv:1402.4735
https://www.kaggle.com/datasets/janus137/supersymmetry-dataset

XGBoost

https://xgboost.readthedocs.io

https://github.com/fsimone91/course_ml4hep/tree/2024/notebooks/2024/4-XGBoost.ipynb