Supervised Learning

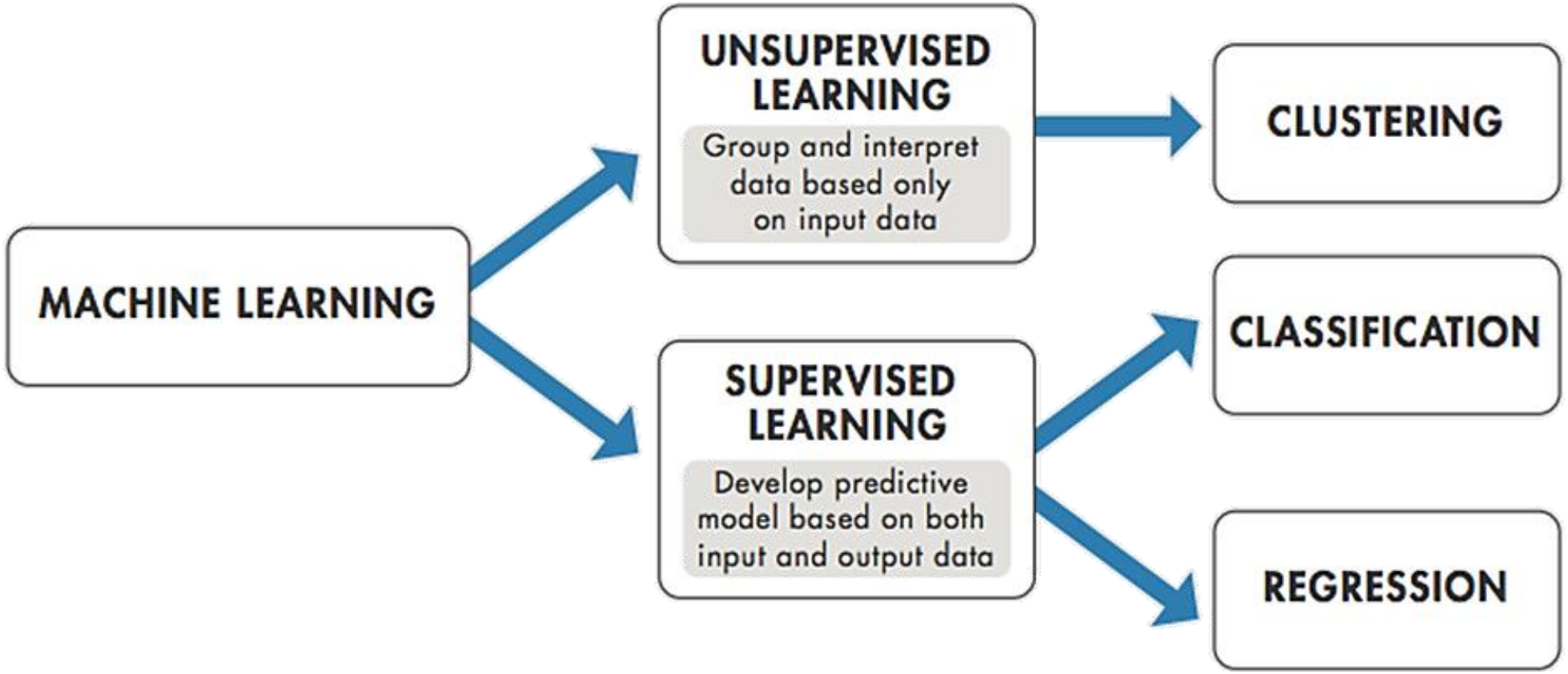
KASHTANOVA VICTORIYA

INRIA, EPIONE

UNIVERSITÉ CÔTE D'AZUR

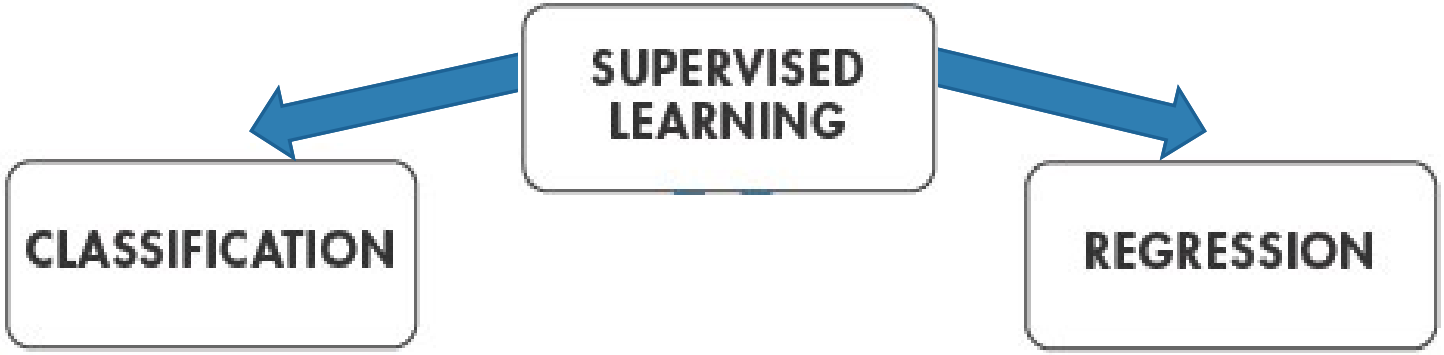
Inria INVENTEURS DU MONDE NUMÉRIQUE

Machine learning methods

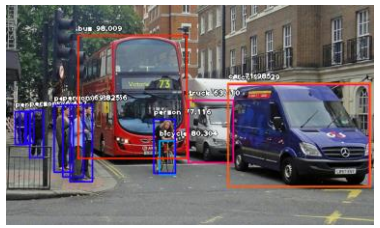


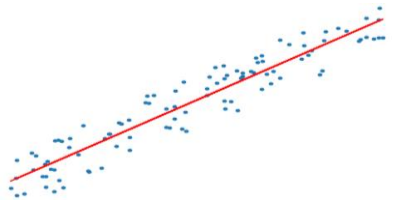
Supervised methods

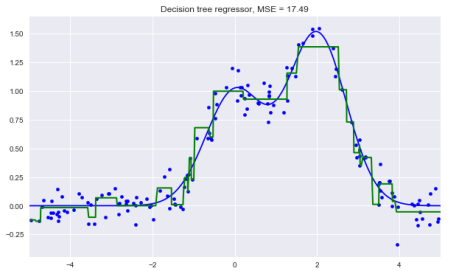
Supervised methods



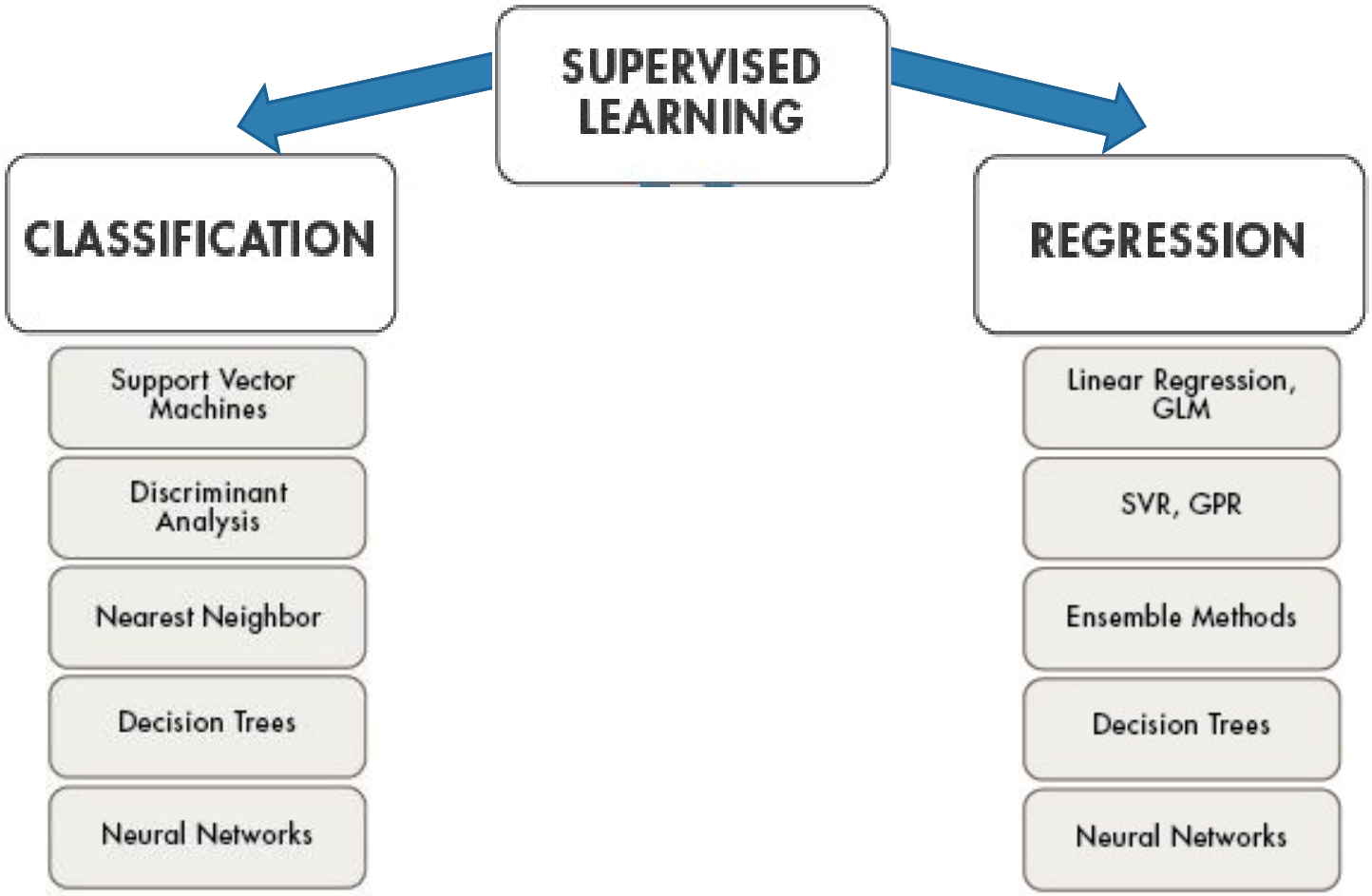
Supervised methods



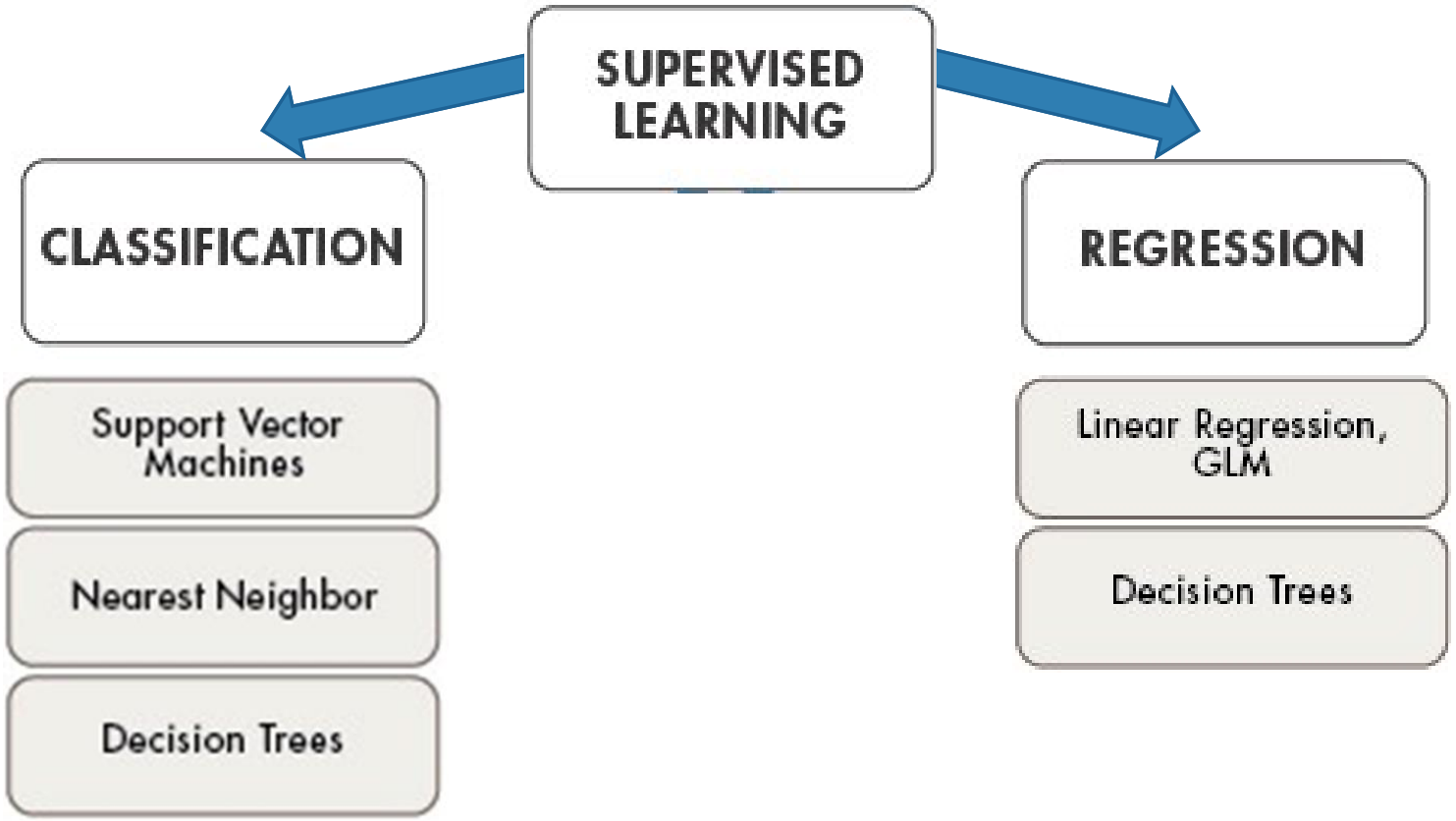




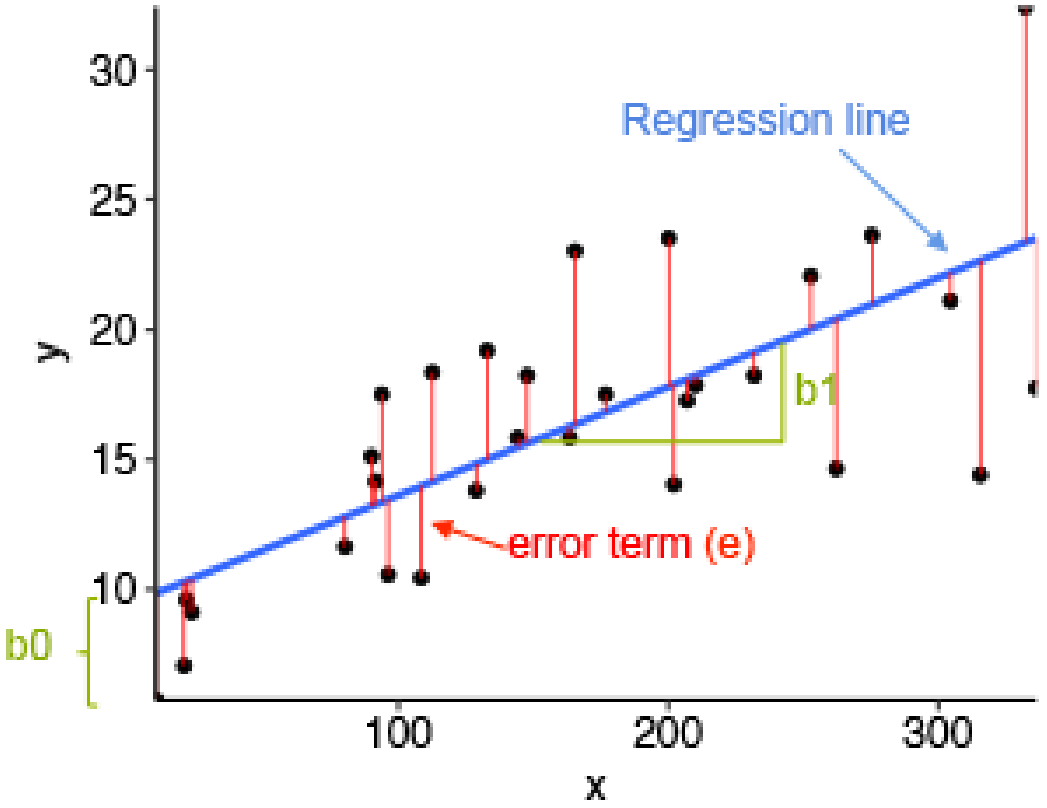
Supervised methods



Supervised methods



Linear Regression

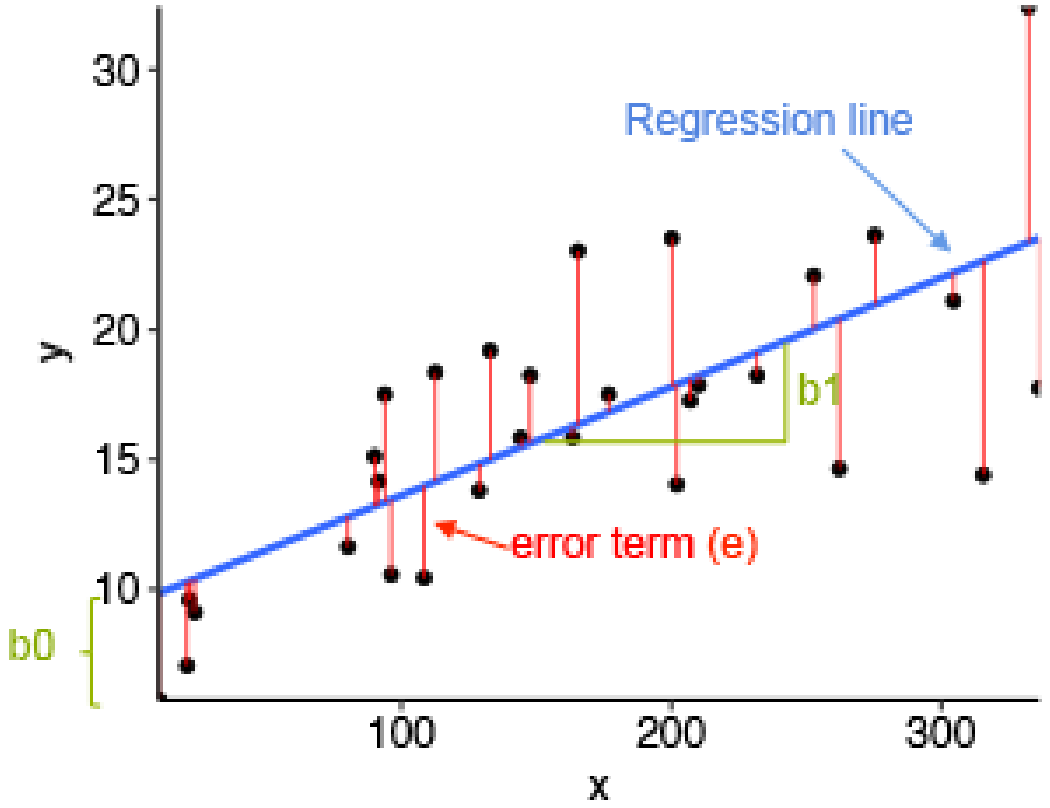


Problem : For set of data $\{(x_i, y_i)\}_1^N$ find an affine function

$$y = f(x) = b_0 + b_1 x + e$$

which define the line closest to the data points.

Linear Regression : Matrix notation

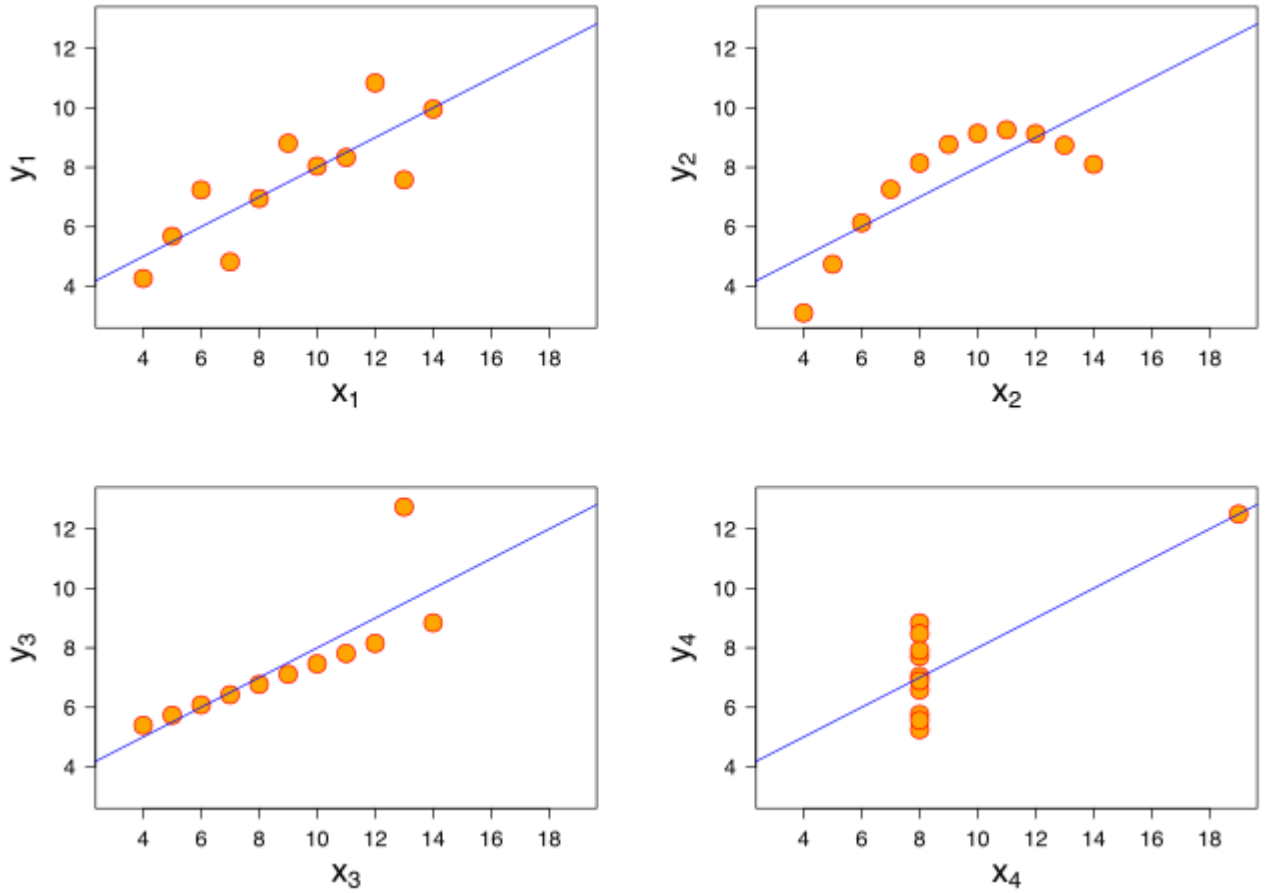


$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

$$X = \begin{pmatrix} \mathbf{x}_1^{\mathsf{T}} \\ \mathbf{x}_2^{\mathsf{T}} \\ \vdots \\ \mathbf{x}_n^{\mathsf{T}} \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix},$$

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

Linear Regression : Examples

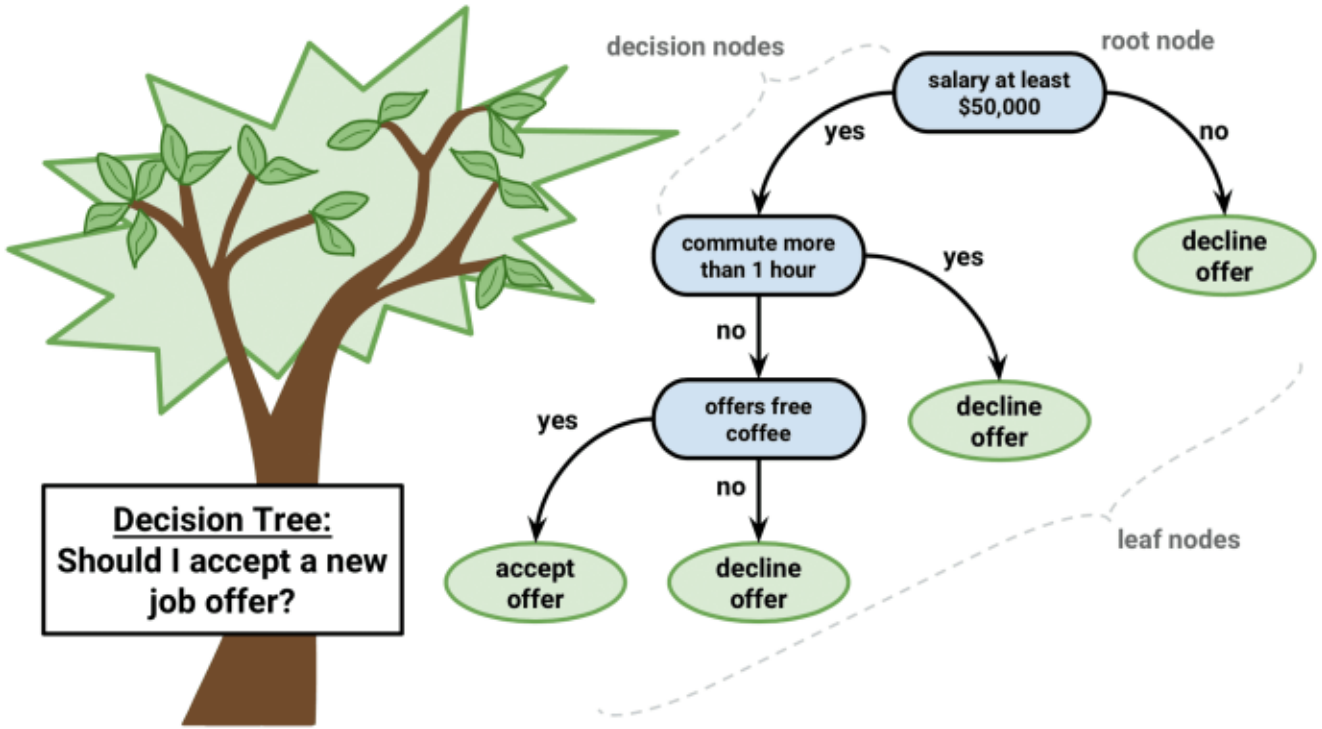


Decision tree

A **decision tree** is a flowchart-like structure in which :

- each **internal node** represents a "test" on an attribute,
- each **branch** represents the outcome of the test,
- and each **leaf node** represents a class label (decision taken after computing all attributes).

The paths from root to leaf represent **classification rules**.



Decision tree : Creation idea

Shannon's entropy is defined for a system with N possible states as follows:

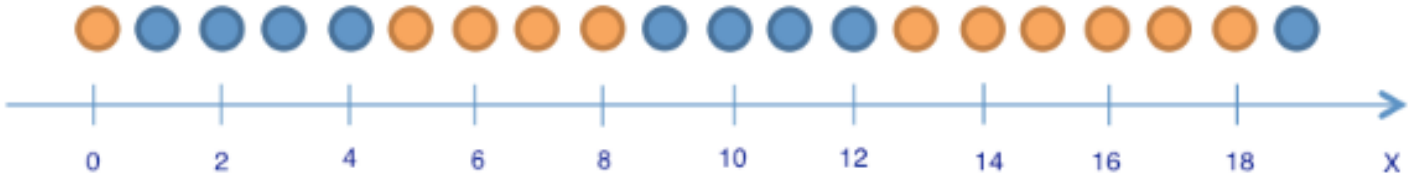
$$\boldsymbol{S} = -\sum_{i=1}^{N} p_i \log_2(p_i),$$

where p_i is the probability of finding the system in the *i* -th state.

<u>Goal :</u> Minimize S

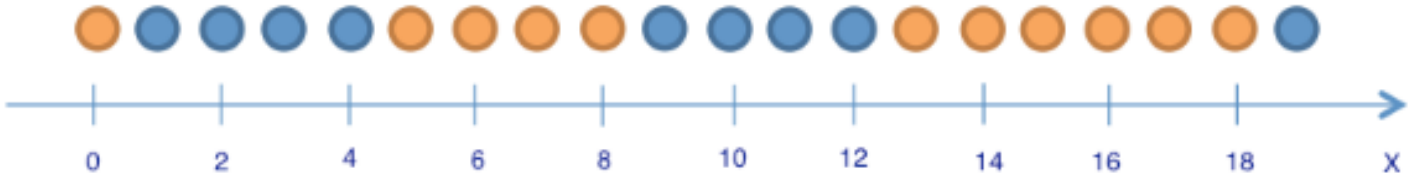
Decision tree : Toy example

Predict the color of the ball based on its position :



Decision tree : Toy example

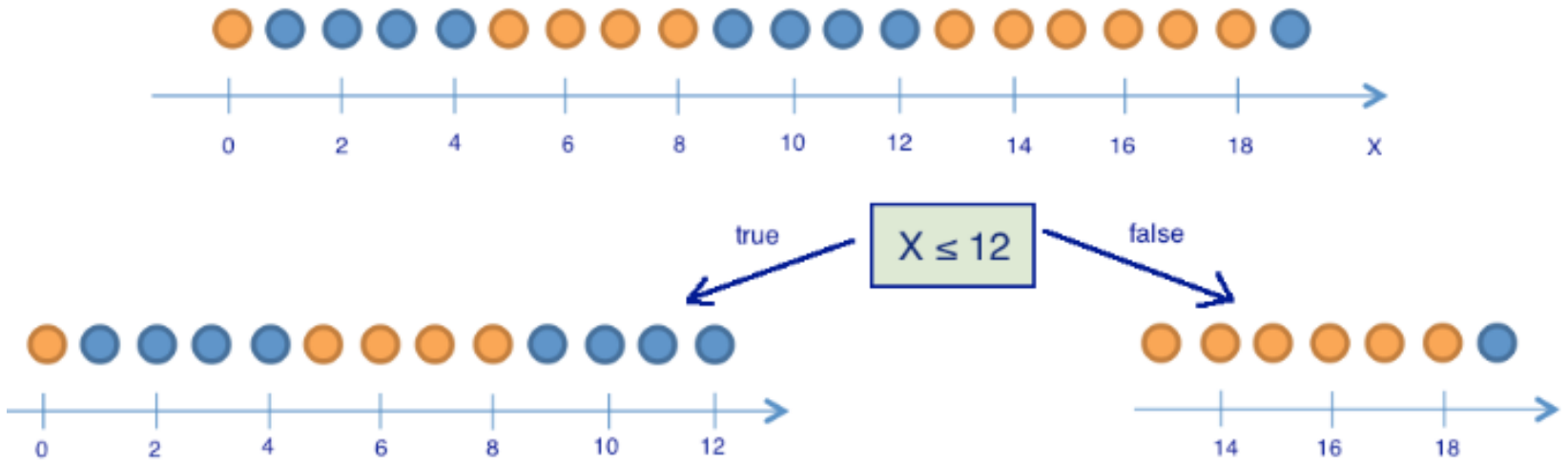
Predict the color of the ball based on its position :



- There are **20 balls** : **9 blue** balls and **11 yellow** balls.
- If we randomly pull out a ball, then it will be blue with probability $\boldsymbol{p_1 = \frac{9}{20}}$ and yellow with probability $\boldsymbol{p_2 = \frac{11}{20}}$, which gives us an entropy :

$$\boldsymbol{S_0 = -\frac{9}{20}\log_2\frac{9}{20} - \frac{11}{20}log_2\frac{11}{20} \approx 1}$$

Decision tree : Toy example



$$S_1 = -\frac{5}{13}\log_2\frac{5}{13} - \frac{8}{13}log_2\frac{8}{13} \approx 0.96$$

$$S_2 = -\frac{1}{7}\log_2\frac{1}{7} - \frac{6}{7}log_2\frac{6}{7} \approx 0.6$$

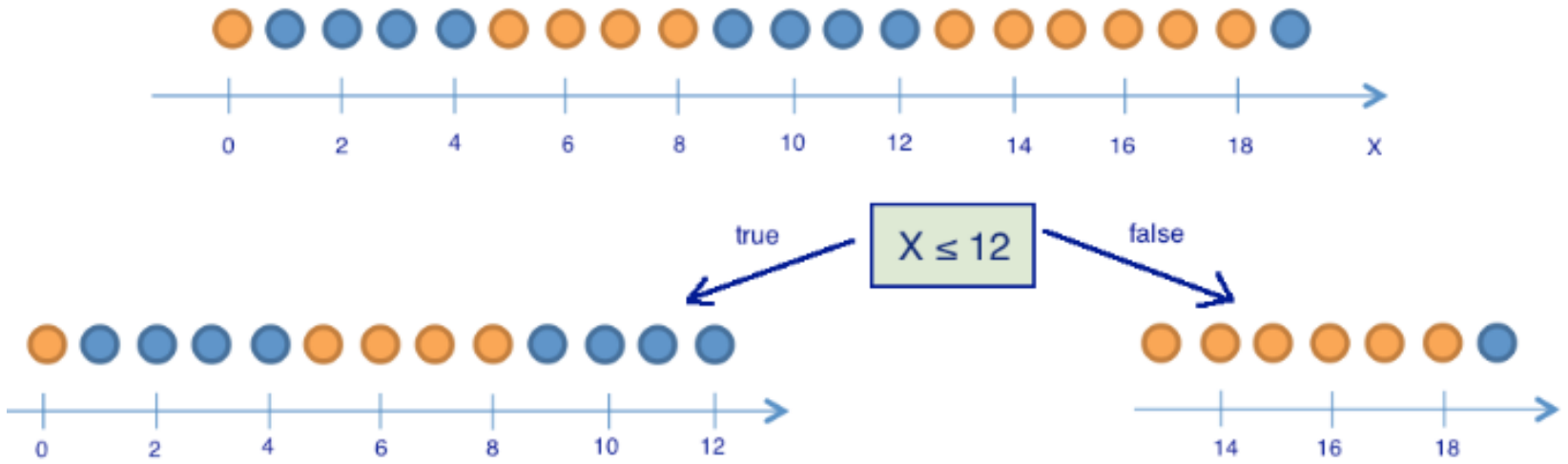
Decision tree : Toy example

Information gain (IG) for a split based on the variable Q (in this example it's a variable " x≤12 ") is defined as :

$$IG(Q) = S_O - \sum_{i=1}^{q} \frac{N_i}{N} S_i,$$

where q is the number of groups after the split, N_i is number of objects from the sample in which variable Q is equal to the i -th value.

Decision tree : Toy example

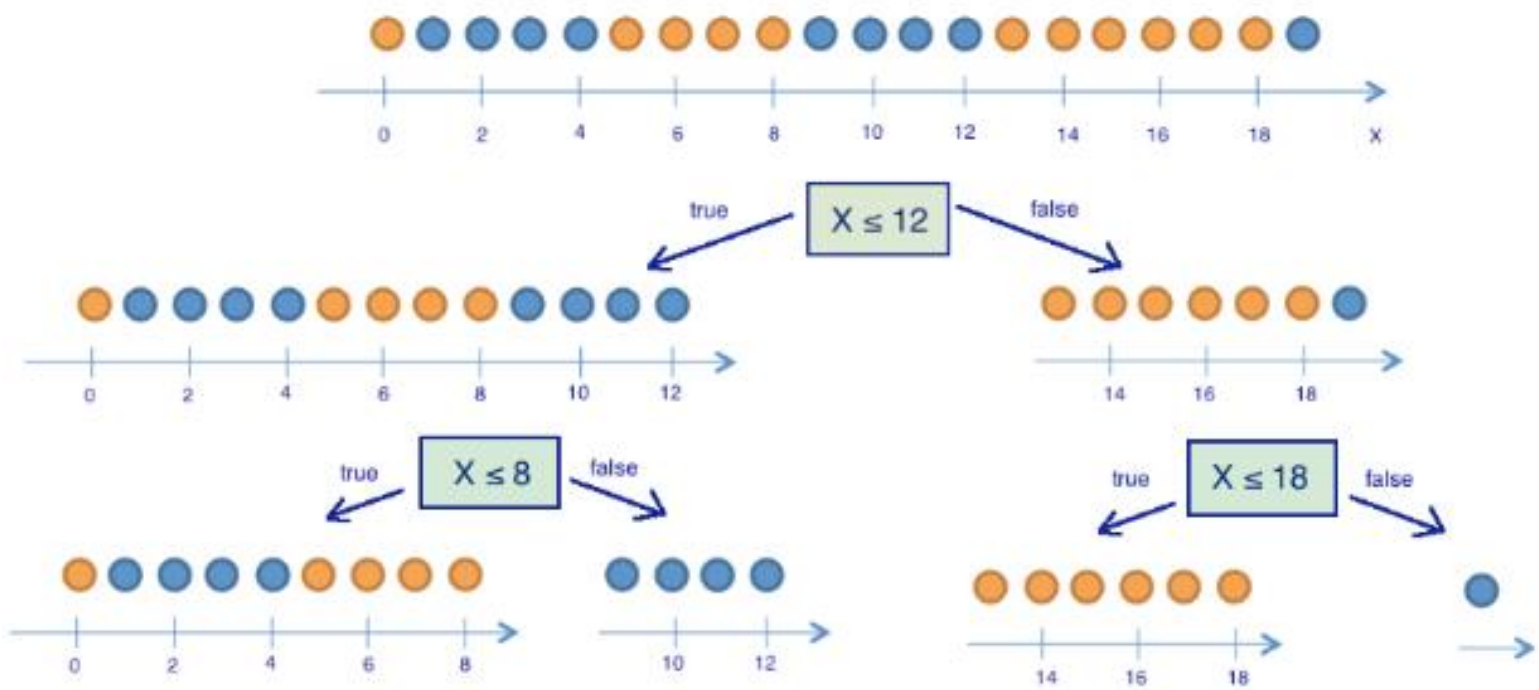


$$S_1 = -\frac{5}{13}\log_2\frac{5}{13} - \frac{8}{13}log_2\frac{8}{13} \approx 0.96$$

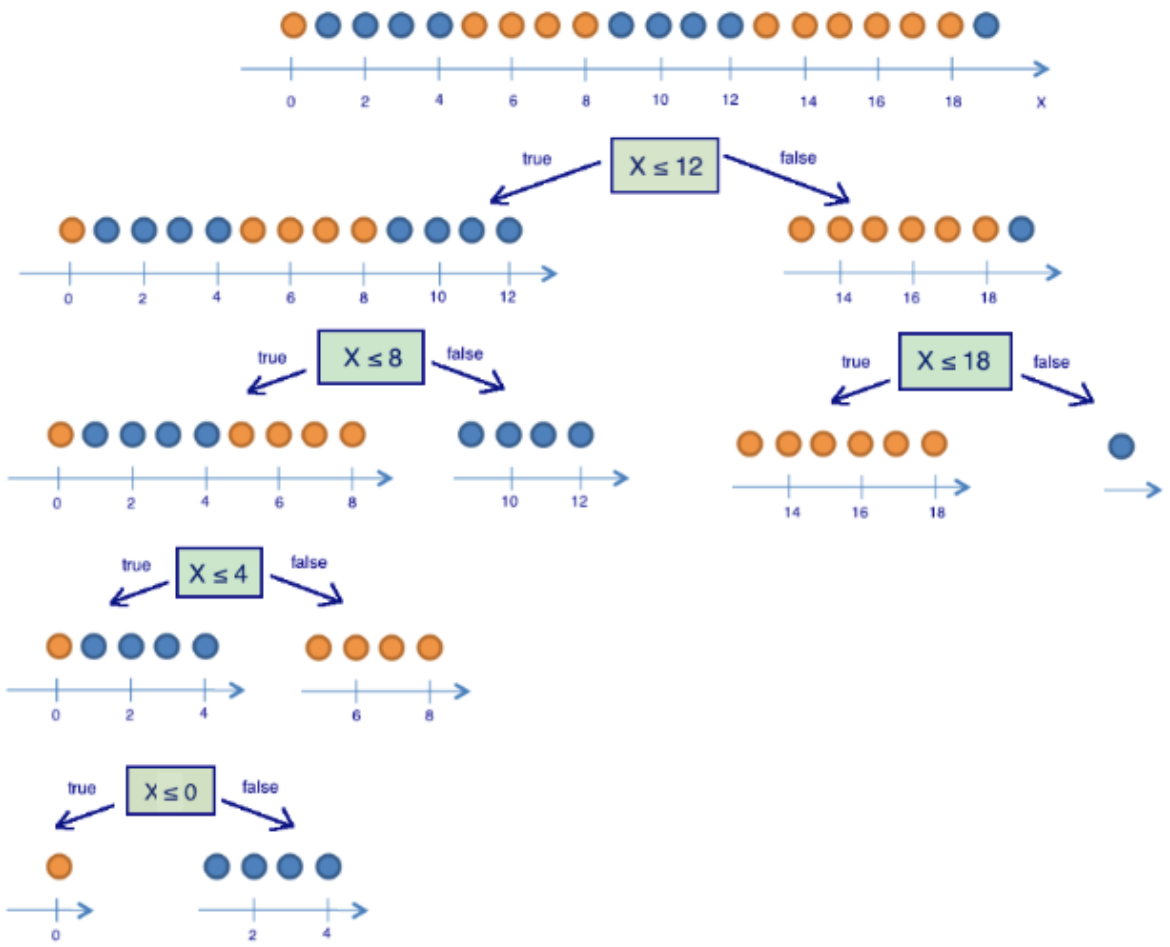
$$S_2 = -\frac{1}{7}\log_2\frac{1}{7} - \frac{6}{7}log_2\frac{6}{7} \approx 0.6$$

$$IG(x \leq 12) = S_0 - \frac{13}{20}S_1 - \frac{7}{20}S_2 \approx 0.16.$$

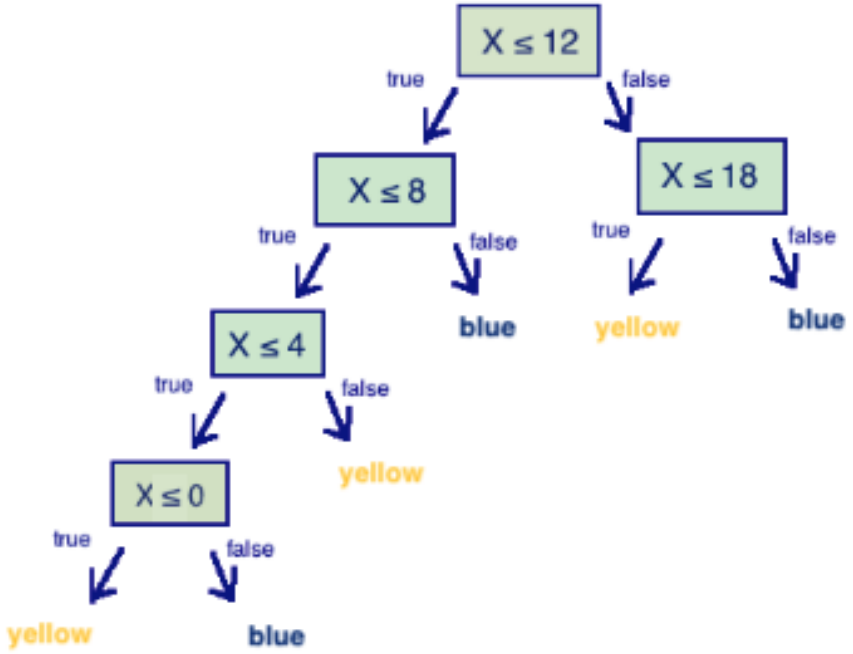
Decision tree : Toy example



Decision tree : Toy example



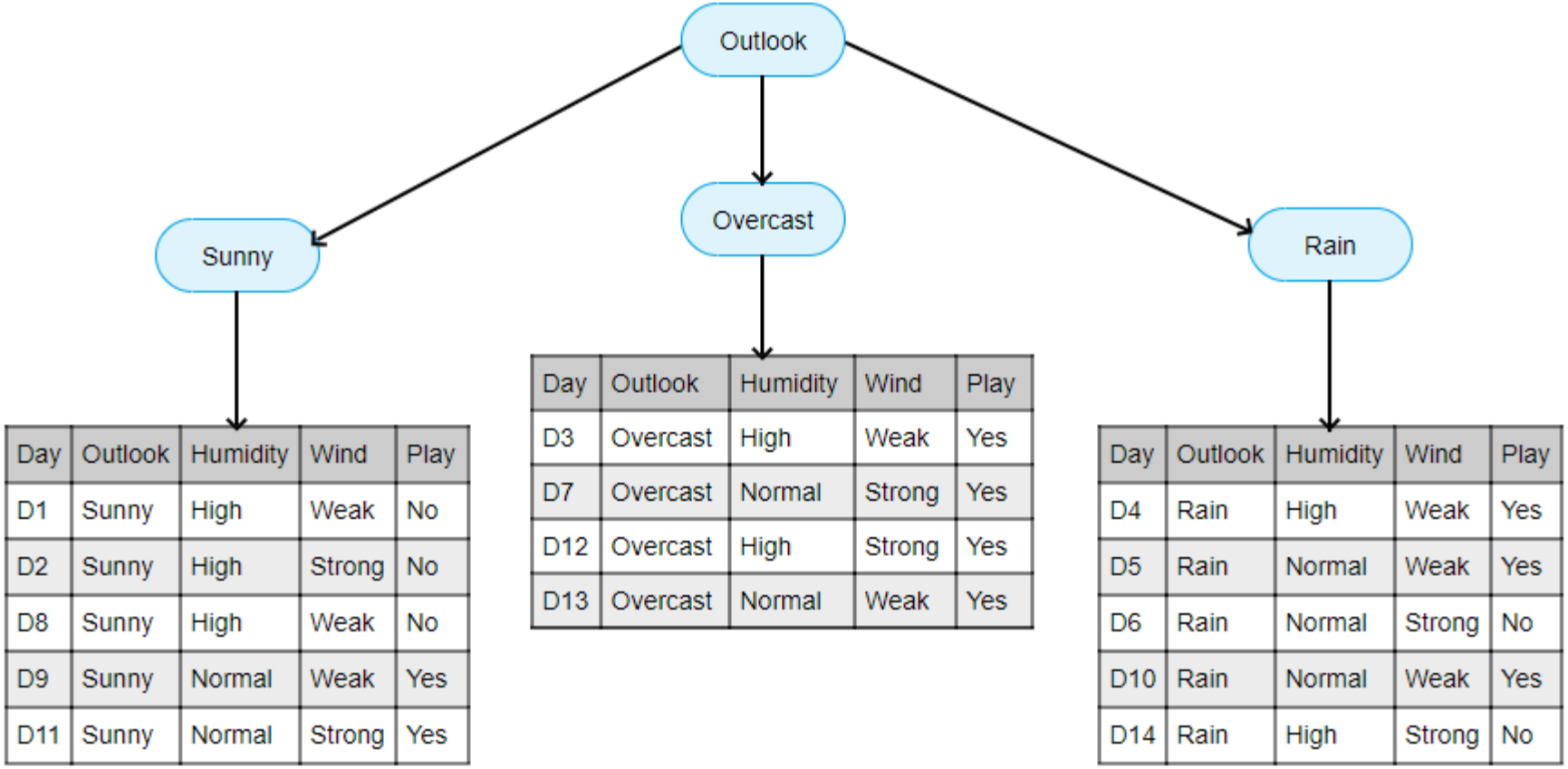
Decision tree : Toy example



Decision tree : Example

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

Decision tree : Example

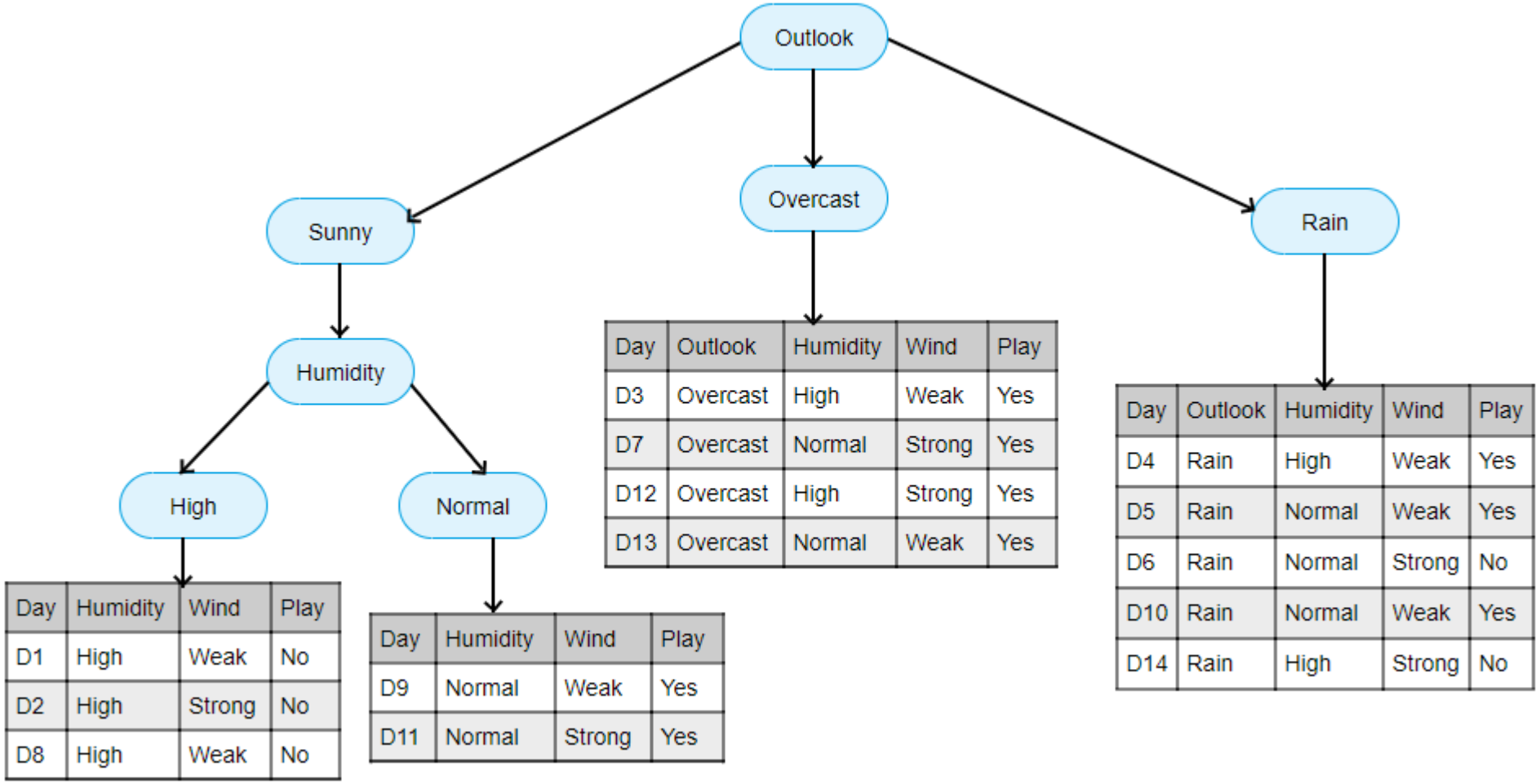


Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes

Day	Outlook	Humidity	Wind	Play
D3	Overcast	High	Weak	Yes
D7	Overcast	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes

Day	Outlook	Humidity	Wind	Play
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D10	Rain	Normal	Weak	Yes
D14	Rain	High	Strong	No

Decision tree : Example



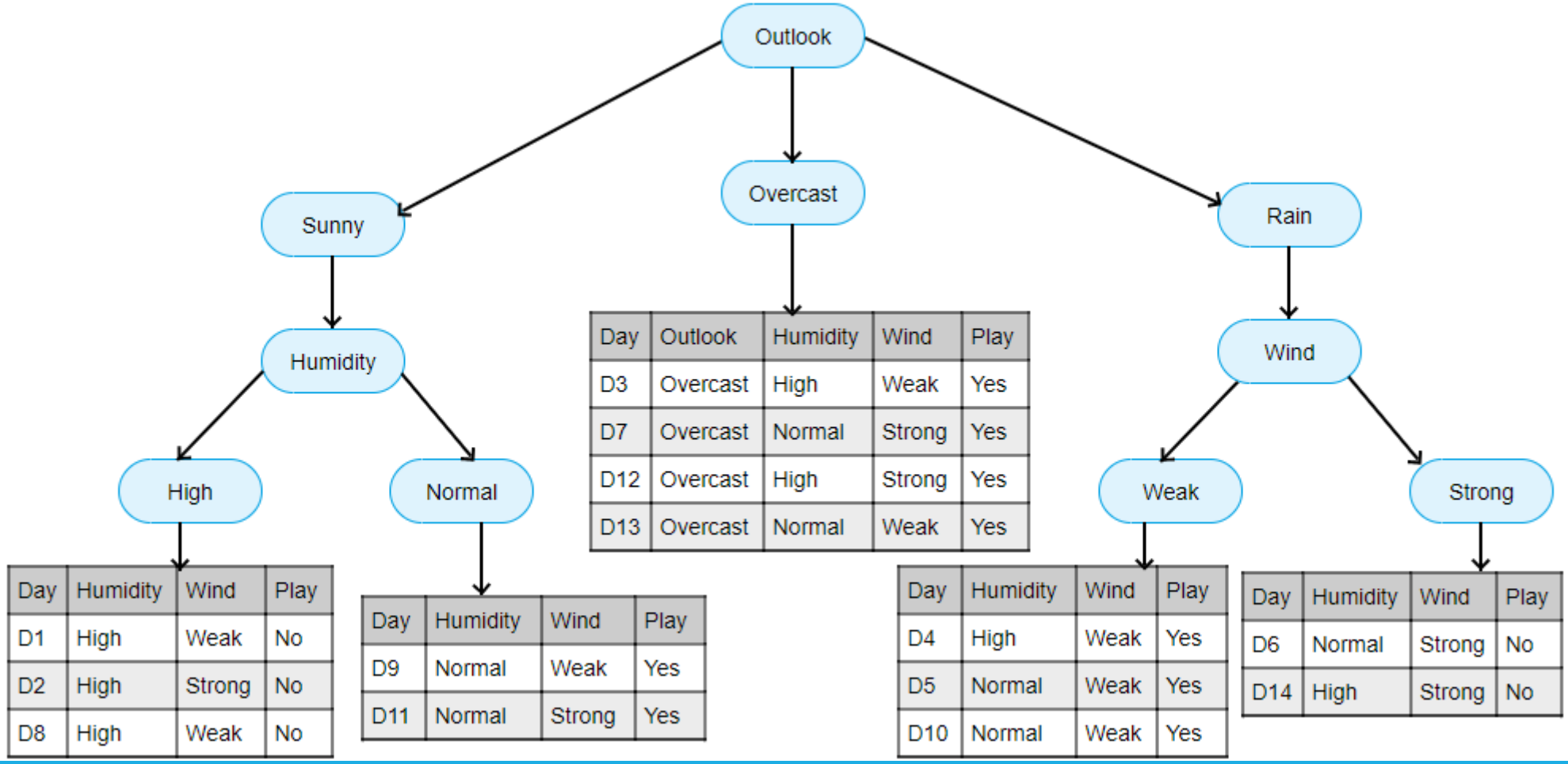
Day	Outlook	Humidity	Wind	Play
D3	Overcast	High	Weak	Yes
D7	Overcast	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes

Day	Outlook	Humidity	Wind	Play
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D10	Rain	Normal	Weak	Yes
D14	Rain	High	Strong	No

Day	Humidity	Wind	Play
D1	High	Weak	No
D2	High	Strong	No
D8	High	Weak	No

Day	Humidity	Wind	Play
D9	Normal	Weak	Yes
D11	Normal	Strong	Yes

Decision tree : Example



Day	Outlook	Humidity	Wind	Play
D3	Overcast	High	Weak	Yes
D7	Overcast	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes

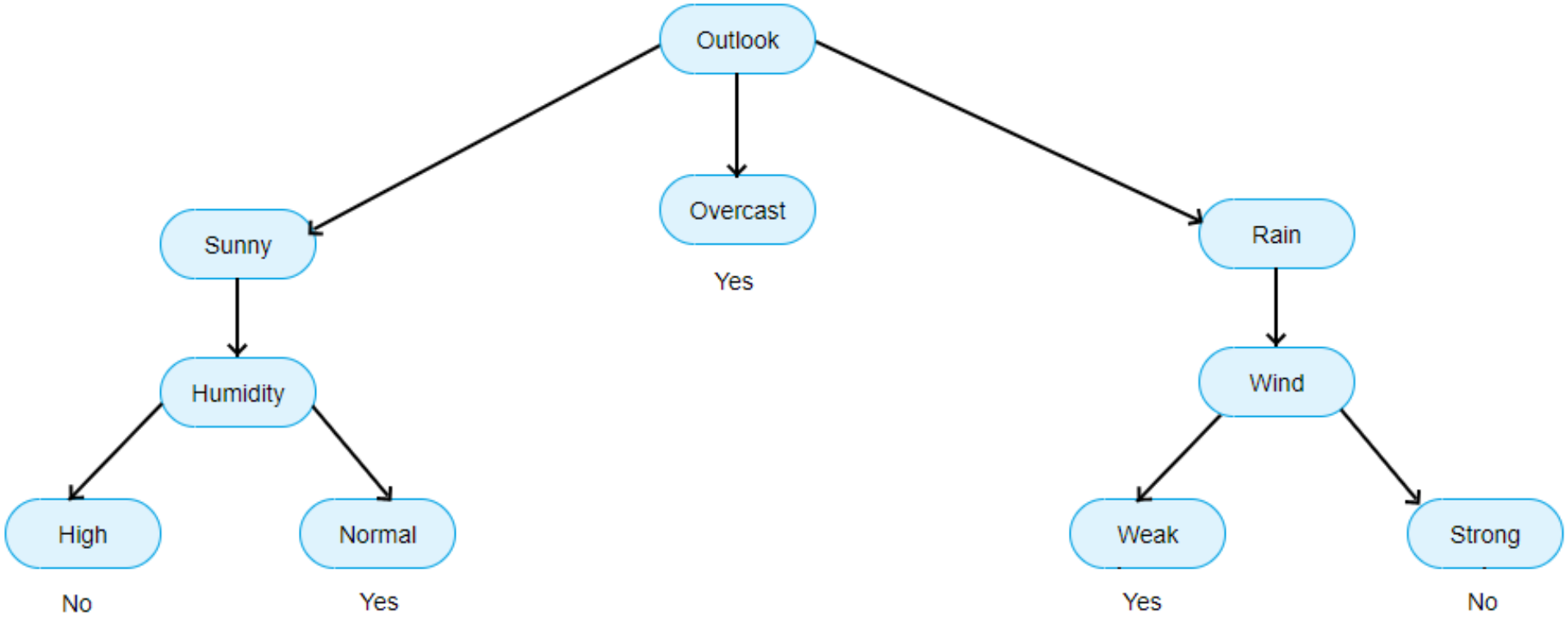
Day	Humidity	Wind	Play
D1	High	Weak	No
D2	High	Strong	No
D8	High	Weak	No

Day	Humidity	Wind	Play
D9	Normal	Weak	Yes
D11	Normal	Strong	Yes

Day	Humidity	Wind	Play
D4	High	Weak	Yes
D5	Normal	Weak	Yes
D10	Normal	Weak	Yes

Day	Humidity	Wind	Play
D6	Normal	Strong	No
D14	High	Strong	No

Decision tree : Example



Decision tree : Tree-building Algorithm

```
def build(L):
    create node t
    if the stopping criterion is True:
        assign a predictive model to t
    else:
        Find the best binary split L = L_left + L_right
        t.left = build(L_left)
        t.right = build(L_right)
    return t
```

Decision tree : Regression problem

When predicting a numeric variable, the idea of a tree construction remains the same, but the quality criteria changes:

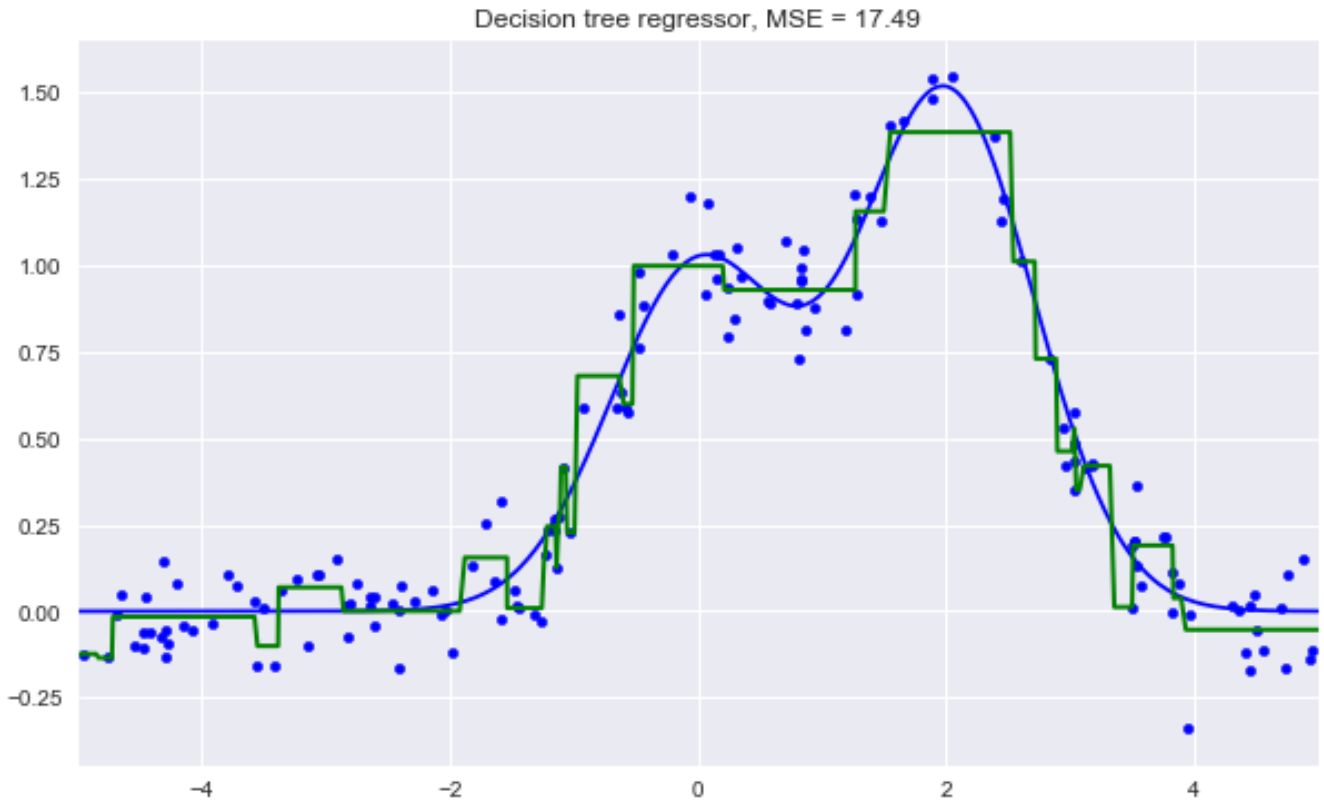
Variance:

$$D = \frac{1}{\ell}\sum_{i=1}^{\ell}(y_i - \frac{1}{\ell}\sum_{j=1}^{\ell} y_j)^2,$$

where ℓ is the number of samples in a leaf, y_i is the value of the target variable.

Simply put, by minimizing the variance, we look for features that divide the training set in such a way that the values of the target feature in each leaf are roughly equal.

Decision tree : Regression problem



Decision tree : Advantages & Drawbacks

- **High interpretability of the model :**
  - Generation of clear human-understandable classification rules
  - Can be easily visualized
- **Fast training and forecasting**
- Small number of model parameters
- Supports both numerical and categorical features

- **High sensitivity to the noise** in input data; the whole model could change if the training set is slightly modified (e.g. remove a feature, add some objects).
- **Rough separating borders**, could overfit
- **Difficulties to support missing values** in the data
- **The model can only interpolate but not extrapolate**
- The optimal decision tree search problem is NP-complete.

KNN (K Nearest Neighbours) Method

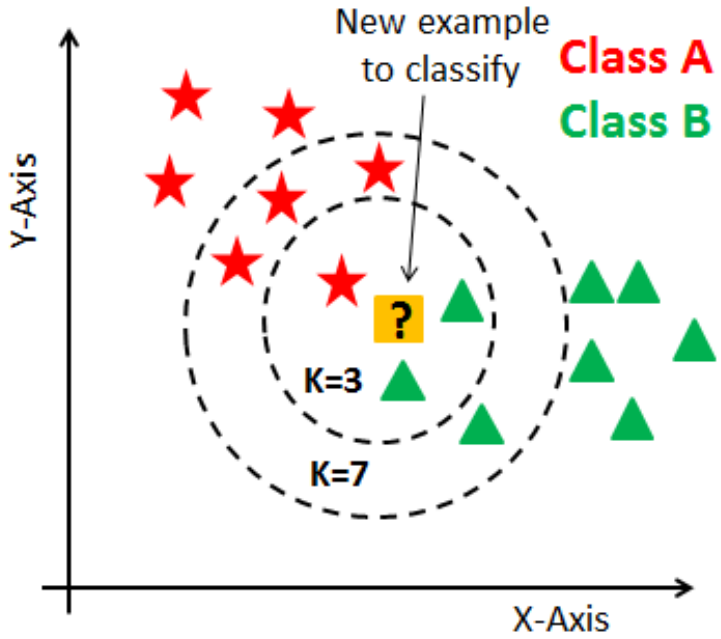
Base of the method is **Compactness hypothesis**:

> "If the distance between the examples is measured well enough, then similar examples are much more likely to belong to the same class."

KNN Method

To classify each sample from the test set, one needs to perform the following operations in order:

1. Calculate the distance to each of the samples in the training set.
2. Select ***K*** samples from the training set with the minimal distance to them.
3. The class of the test sample will be the most frequent class among those K nearest neighbors.

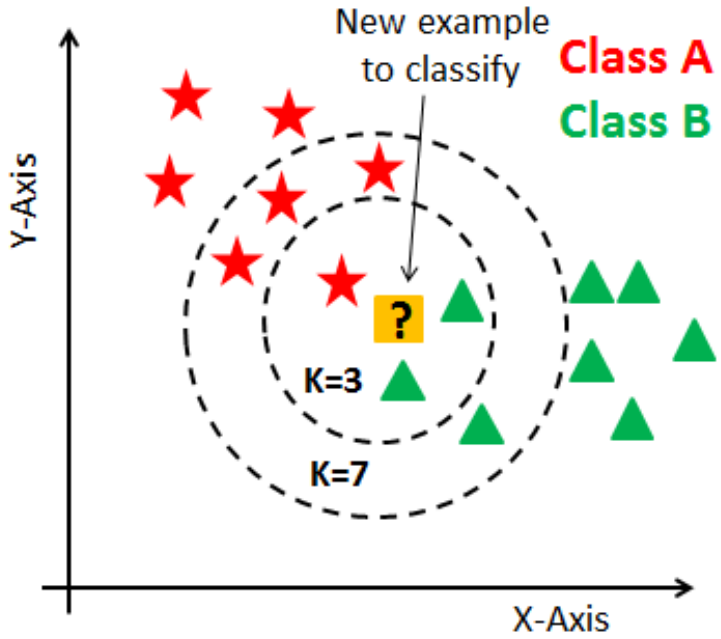


KNN Method

To classify each sample from the test set, one needs to perform the following operations in order:

1. Calculate the distance to each of the samples in the training set.
2. Select ***K*** samples from the training set with the minimal distance to them.
3. The class of the test sample will be the most frequent class among those K nearest neighbors.

No model is constructed from the training examples beforehand.
Calculations are only done during the prediction phase, when a test sample needs to be classified.

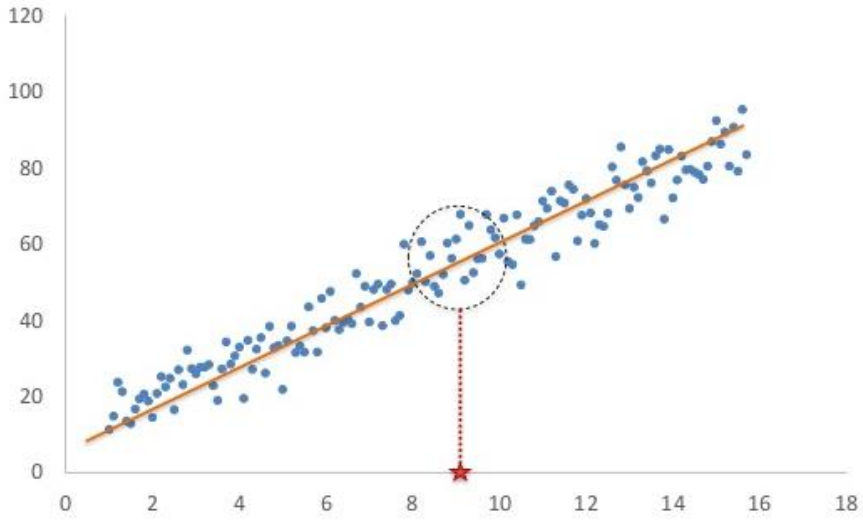


KNN Method

To classify each sample from the test set, one needs to perform the following operations in order:

1. Calculate the distance to each of the samples in the training set.
2. Select ***K*** samples from the training set with the minimal distance to them.
3. The class of the test sample will be the most frequent class among those K nearest neighbors.

The method adapts quite easily for the **regression problem**: on step 3, it returns not the class, but the number – a mean (or median) of the target variable among neighbors.



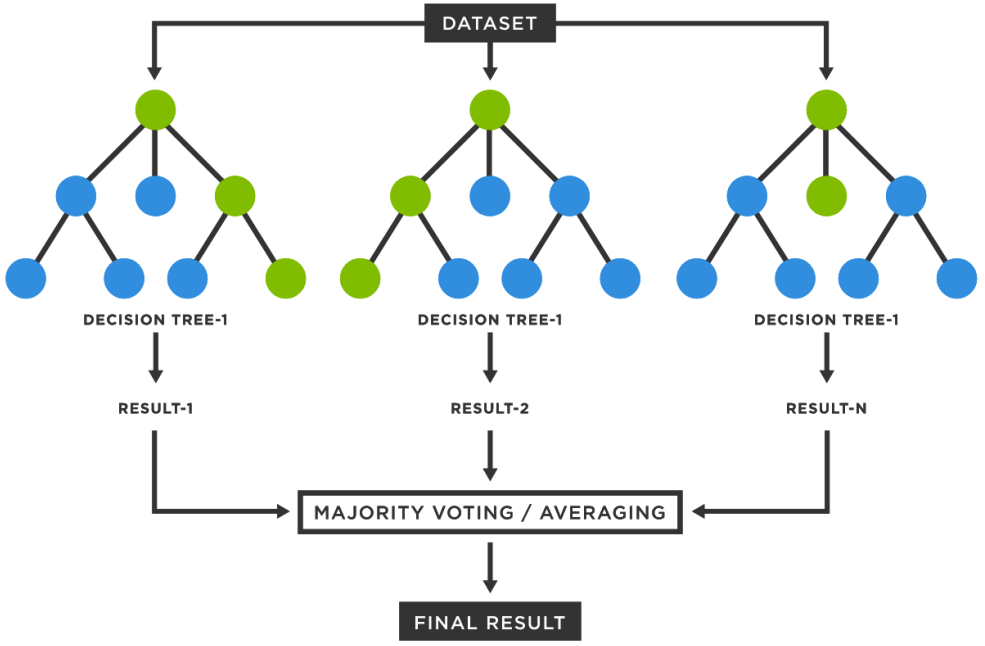
KNN Method : Advantages & Drawbacks

- Well-studied approach
- **Good interpretability** (for small K)
- **Fast forecasting** (for small K)
- **Simple implementation**
- Used as a first solution
- It can be adapted to any problem by choosing the right metrics or kernel

- There are **no theoretical ways to choose the number of neighbours**
- Dependency on the selected distance metric between the objects
- It does not work well when there are a lot of features due to the "**curse of dimensionality**"
- In real problems **K is usually large**, lose in interpretability and forecasting

Random Forest Method

1. Set of simple Decision Trees (DTs)
2. Decorrelated training of DTs :
   - Each DT learn sub-set of data (randomly assigned)
   - Each DT choose randomly the features to use
3. Voting to make classification decision

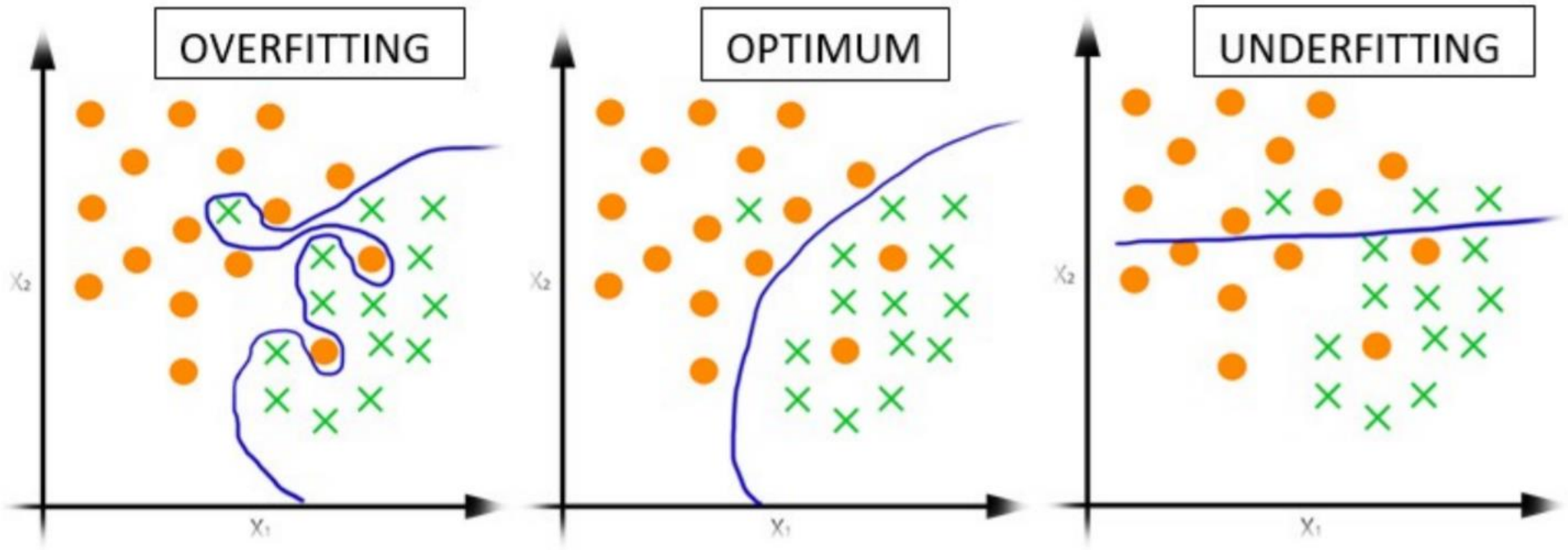


Random Forest : Advantages & Drawbacks

- **High prediction accuracy**
- Robust to outliers
- Insensitive to the scaling of features as well as any other monotonic transformations
- Handles both continuous and discrete variables equally well
- **Rarely overfits**
- Works well with missing data
- Etc.

- **Bad interpretability**
- Performs worse than linear methods in the case of sparse data
- Unable to extrapolate (maybe an advantage)
- Prone to overfitting in some problems, especially, when dealing with noisy data
- **The resulting model is large and requires a lot of RAM**

Examples of classifiers



Model evaluation methods

Performance measures

Classification :

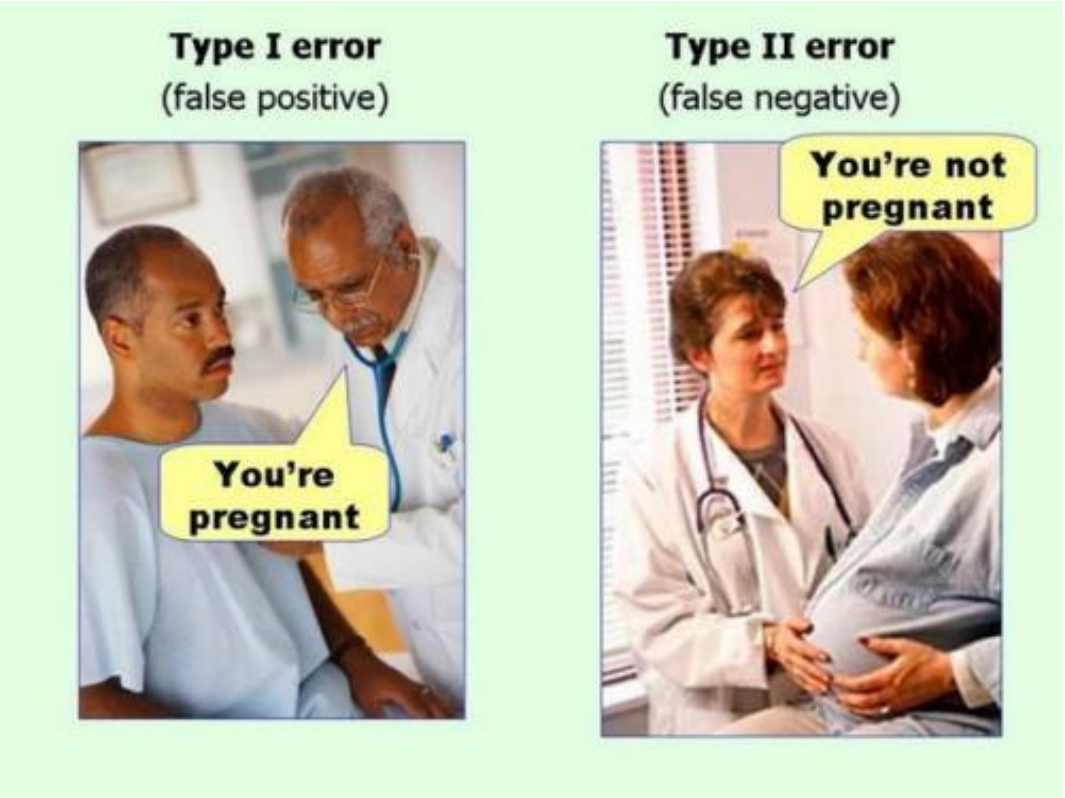
- Simple Accuracy
- Precision
- Recall
- F-beta mesure
- ROC (and AUC)

Regression :

- Mean Absolute Error
- Sum of Squares Error
- Root Mean Square Error

Confusion matrix

		Predicted class: P	Predicted class: N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)



Metrics

- $\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} = \frac{(TP + TN)}{N}$
  (fraction of correct predictions)
- $\text{Precision} = \frac{TP}{(TP+FP)}$
  (fraction of correctly predicted positive values to all values predicted positive)
- $\text{Recall} = \frac{TP}{(TP + FN)}$
  (completeness, fraction of correctly predicted positive values to all positive values)

- $\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} = \frac{(TP + TN)}{N}$
  (fraction of correct predictions)
- $\text{Precision} = \frac{TP}{(TP+FP)}$
  (fraction of correctly predicted positive values to all values predicted positive)
- $\text{Recall} = \frac{TP}{(TP + FN)}$
  (completeness, fraction of correctly predicted positive values to all positive values)

Metrics

- $\mathbf{F_\beta} = \frac{\mathbf{1}}{\left(\boldsymbol{\beta} * \frac{\mathbf{1}}{\mathbf{Precision}} + (\mathbf{1} - \boldsymbol{\beta}) * \frac{\mathbf{1}}{\mathbf{Recall}}\right)}$ (greater β, greater importance of Precision)
- $\mathbf{F_1} = \frac{\mathbf{2TP}}{\mathbf{(2TP + FP + FN)}}$ (harmonic mean of precision and recall, $\beta = 0.5$)

Model validation techniques

Model validation techniques

The main task of learning algorithms is to be able to generalize to unseen data.

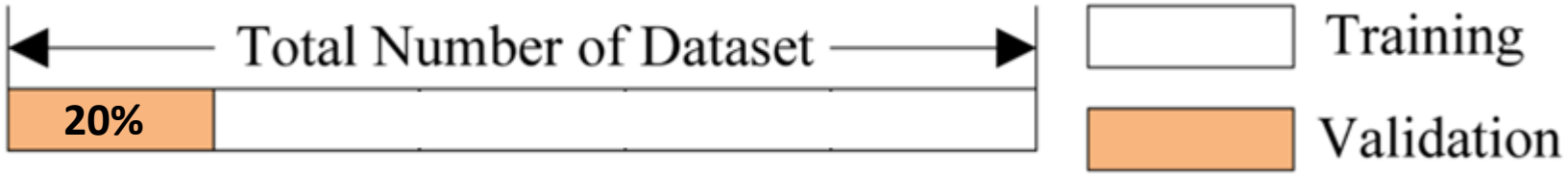
Since we cannot immediately check the model performance on new, incoming data (because we do not know the true values (labels) of the target variable yet), it is necessary to sacrifice a small portion of the data to check the quality of the model on it.

This is often done in one of two ways:

- Setting aside a part of the dataset (**held-out/hold-out set**);
- **Cross-validation.**

Hold-out set (popular)

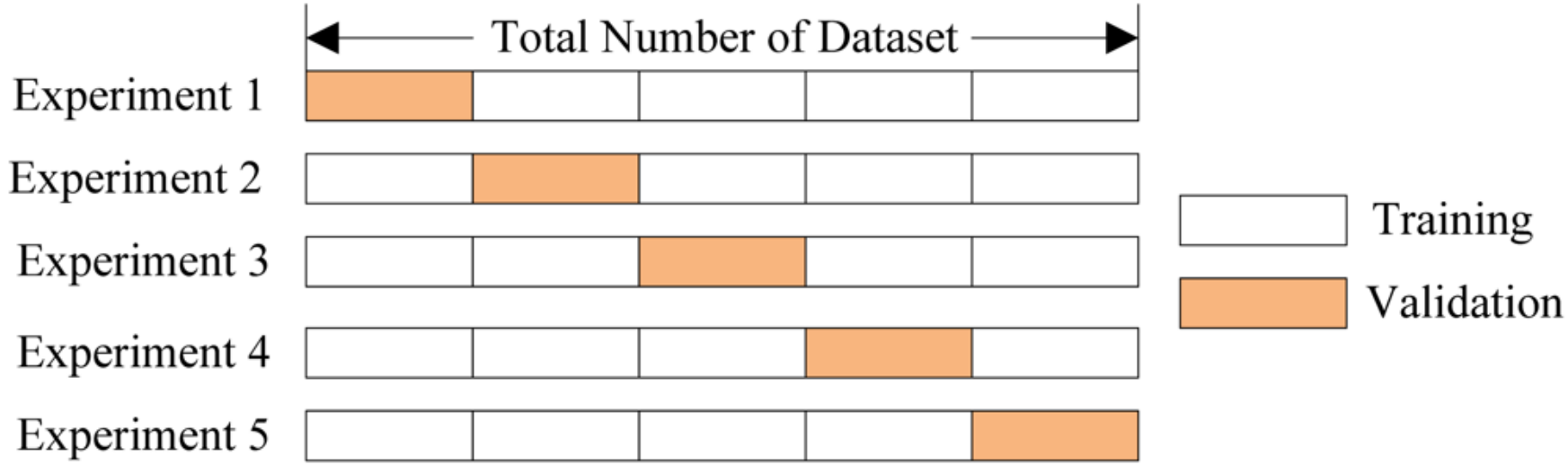
We reserve a fraction of the training set (typically from 20% to 40%), train the model on the remaining data (60-80% of the original set), and compute performance metrics for the model (e.g accuracy) on the hold-out set.



Cross-validation (more informative)

In **K-fold cross-validation :**

- Model is trained K times on different (K−1) subsets of the original dataset (in white).
- Model is checked on the remaining subset (each time a different one, shown above in orange).
- We obtain K model quality assessments that are usually averaged to give an overall average quality of classification/regression.



Sources

- MIT course “Introduction to Computational Thinking and Data Science” (Prof. Eric Grimson, Prof. John Guttag)
- Open Machine Learning Course (by Yury Kashnitsky, mlcourse.ai)
- YouTube lections “Algorithms and Concepts” (by CodeEmporium)