IFT 615 – Intelligence Artificielle

Apprentissage avec les arbres de décision

Professeur: Froduald Kabanza
Assistants: D'Jeff Nkashama

Université de Sherbrooke

Sujets couverts par cette leçon

- Concept d’un arbre de décision

- Algorithme d’apprentissage d’un arbre de décision

- *Random forest* (Forêts aléatoires)

Exemple – Attendre pour un resto

$x_i \equiv [x_{i1}, \quad x_{i2}, \; x_{i3}, \; x_{i4}, \quad x_{i5}, \quad x_{i6}, \quad x_{i7}, \quad x_{i8}, \; x_{i9}, \; x_{i10}]$

$\equiv$ [Alt, Bar, Fri, Hun, Pat, Price, Rain, Res, Type, Est]

$y_i \equiv$ [WillWait]

- *Alternate*: Il y un resto alternatif tout proche ou non
- *Bar*: le resto a un bar confortable pour y attendre ou non
- *Fri / Sat*: On est vendredi ou samedi ou non
- *Hungry*: J’ai beaucoup faim ou non
- *Patrons*: Achalandage en ce moment (valeurs: *None*, *Some*, *Full*)
- *Price*: la gamme de prix du resto ($, $$, $$$)
- *Raining*: Il pleut ou non
- *Reservation*: J’ai réservé ou non
- *Type*: Type de resto (French, Italian, Thai ou Burger)
- *WaitEstimate*: Temps d’attente (valeurs: 0-10, 01-30, 30-60, > 60 min)

Exemple 1 – Attendre pour un resto

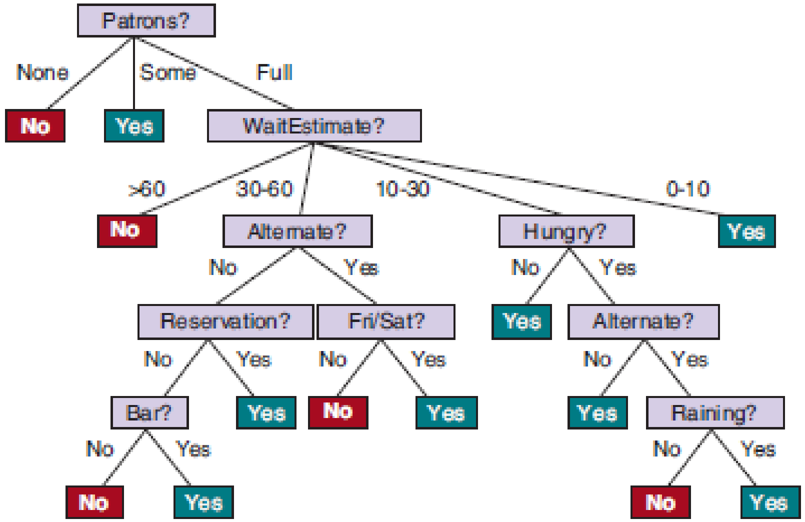
Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Jeu de données pour l’application du resto

Arbre de décision

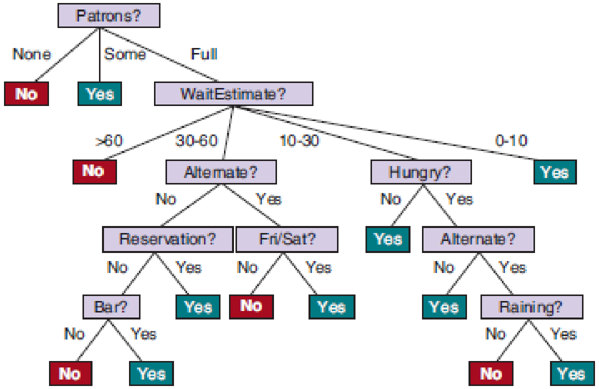
- Un arbre de décision est un modèle dont l'inférence correspondant à une séquence de tests. Chaque nœud intérieur représente un test sur une variable x_i du vecteur d'entrées x et un nœud feuille représente la valeur de la variable cible y.

Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

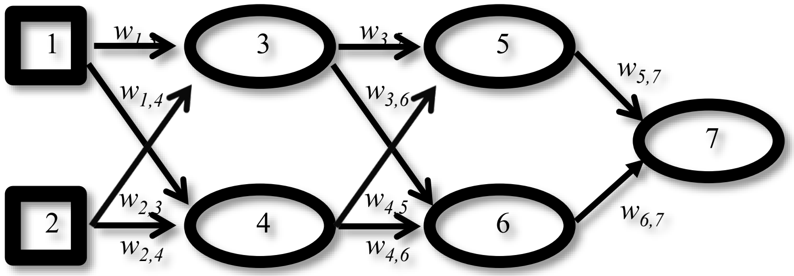


Arbre de décision *vs* réseau de neurones

- De façon générale, un modèle (appris) est une fonction $y = f(x)=f(x_1, .., x_n)$:
  - $x=[x_1, ...,x_n]$: entrée (c.-a.d-. x_i sont les variables d'entrée)
  - y : cible d'apprentissage.
- Pour un réseau de neurones, la fonction $f \equiv f_w$ est représentée par le vecteur de poids w : $y = f_w(x)=f_w(x_1, .., x_n)$.
- Dans le cas d'un arbre de décision, la fonction $f \equiv f_T$ est représentée par un arbre de décisions T. Chaque nœud intérieur est un test sur une variable d'entrée. Chaque feuille la variable cible.

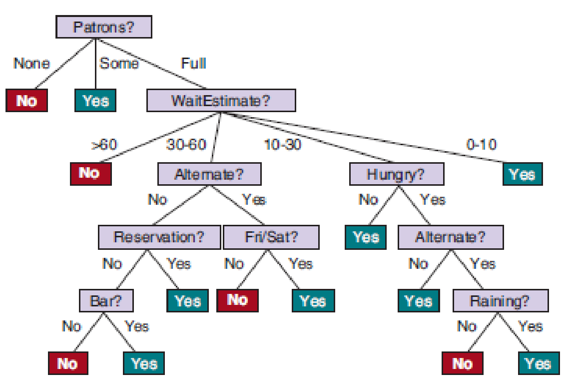


VS

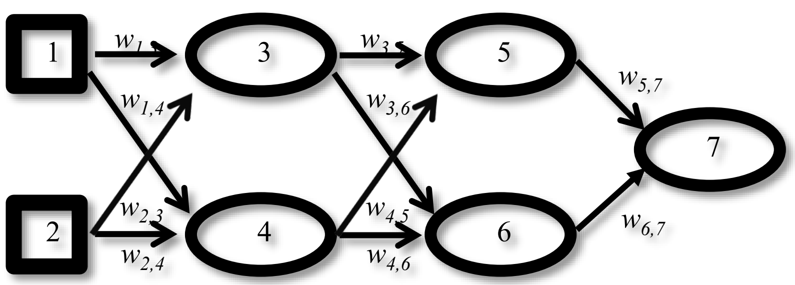


Arbre de décision *vs* réseau de neurones

- Avec les réseaux de neurones, les arbres de décision sont actuellement les deux types de représentations les plus utilisées pour l'apprentissage supervisé dans l'industrie.
- Tout comme un réseau de neurones, un arbre de décision est un modèle paramétrique.
- Un *arbre de décision* est un modèle symbolique, plus facile à interpréter.



VS

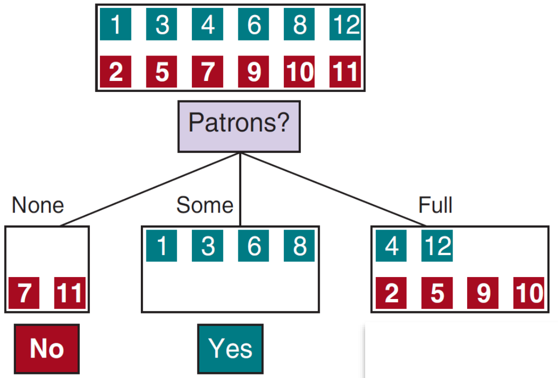


Algorithme d’apprentissage

- Un nœud correspond à un test sur une variable et ensemble d’exemples.
- Le nœud racine correspond à tous les exemples.
- Itérativement
  - Choisir la **variable la plus importante** non encore choisie jusqu'ici
  - Lui appliquer un test sur ses valeurs
  - Pour chaque valeur du test
    - » Générer un enfant contenant les exemples consistent avec le test

function LEARN-DECISION-TREE(*examples*, *attributes*, *parent_examples*) **returns** a

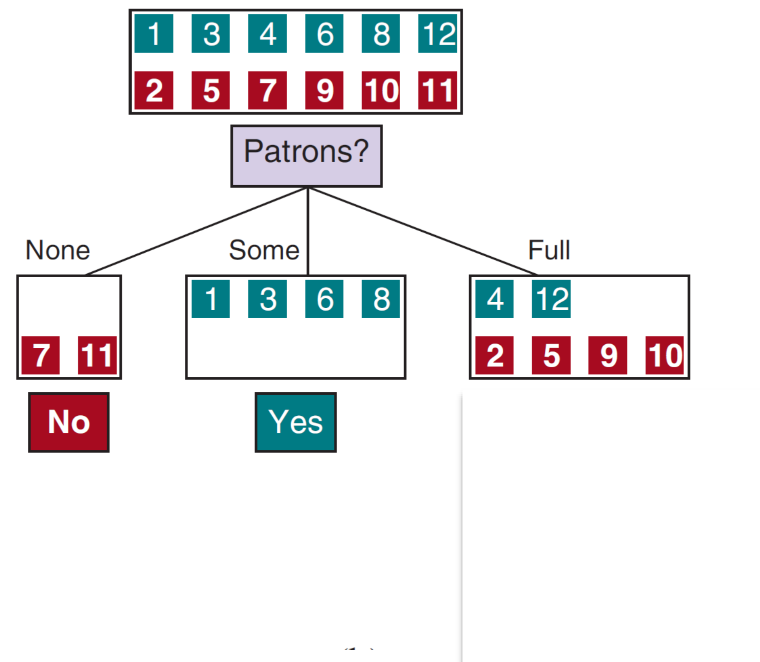
  if *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
  else if all *examples* have the same classification **then return** the classification
  else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
  else
    $A \leftarrow \mathrm{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
    tree ← a new decision tree with root test A
    for each value v of A **do**
      $exs \leftarrow \{e : e \in examples$ **and** $e.A = v\}$
      subtree ← LEARN-DECISION-TREE(*exs*, *attributes* − A, *examples*)
      add a branch to *tree* with label $(A = v)$ and subtree *subtree*
    return *tree*



Exemple – Attendre pour un resto

- Itérativement
  - Choisir la **variable la plus importante** non encore choisie jusqu'ici
  - Lui appliquer un test sur ses valeurs
  - Pour chaque valeur du test
    - » Générer un enfant contenant les exemples consistent avec le test

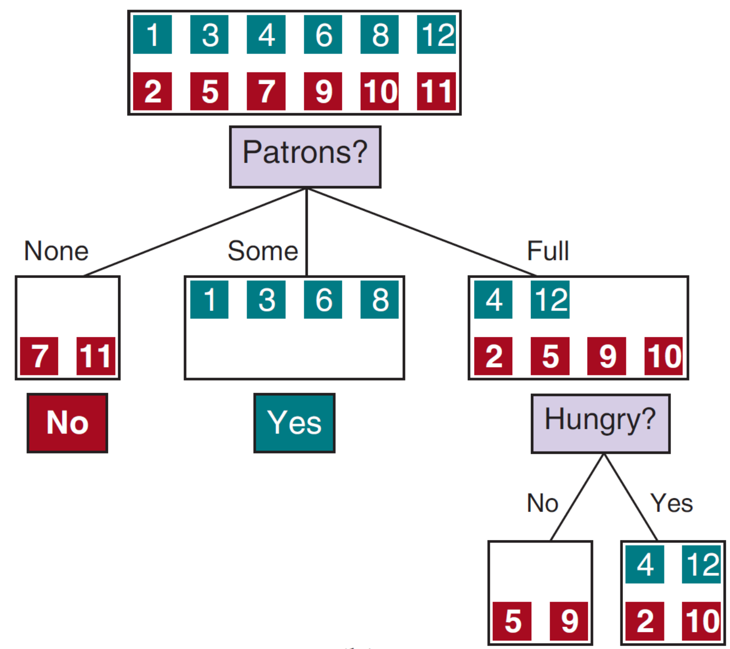
Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$



Exemple – Attendre pour un resto

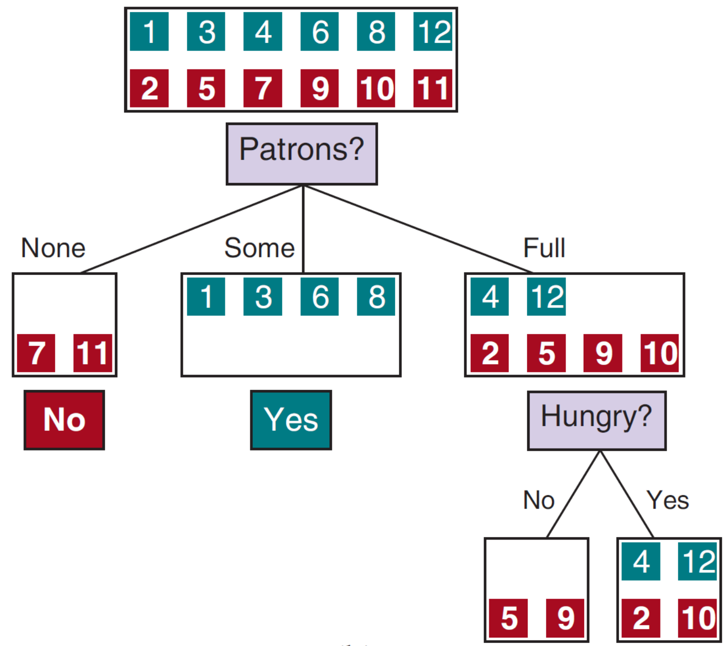
- Itérativement
  - la **variable la plus importante** non encore choisie jusqu'ici
  - Lui appliquer un test sur ses valeurs
  - Pour chaque valeur du test
    - » Générer un enfant contenant les exemples consistent avec le test

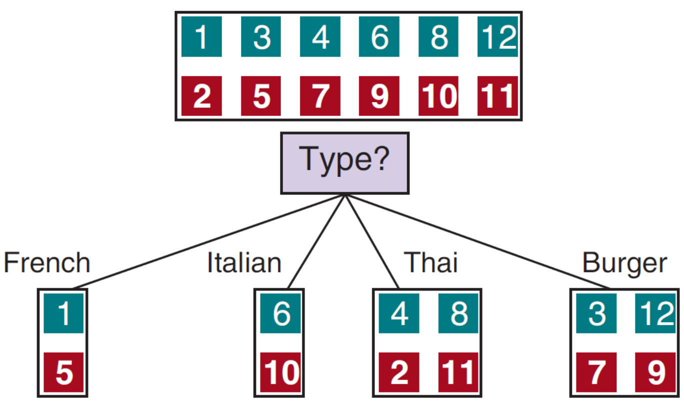
Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$



Exemple – Attendre pour un resto

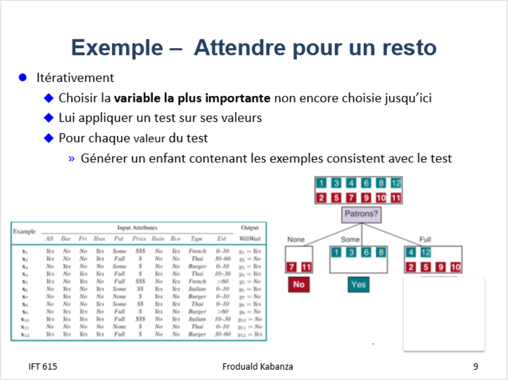
Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$





Choix de la variable à tester

- On veut choisir la variable qui réduit le plus l'incertitude dans les exemples restants à classer



- Cela nous amène à définir d'abord les concepts:
  - Entropie comme mesure d'incertitude
  - Gain d'information en terme de réduction d'entropie

- On va alors choisir l'attribut qui apporte le plus de gain d'information

Entropie

- L'entropie d'une variable aléatoire V ayant les valeurs possibles v_k avec la distribution de probabilité $P(v_k)$ est définie comme étant

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = - \sum_k P(v_k) \log_2 P(vk)$$

Exemples

- L'entropie d'un choix pile ou face:

  $H(PileOuFace) = - (0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1$ *bit*

- Si la pièce est biaisée à 99% pour la face :

  $H(PileOuFaceBiaise) = - (0.99 \log_2 0.99 + 0.01 \log_2 0.01) \approx 0.08$ *bits*

- L'entropie d'un dé à 4 faces non pipé

  $H(de\text{-}4) = - (0.25 \log_2 0.25 + 0.25 \log_2 0.25$
  $+ 0.25 \log_2 0.25 + 0.25 \log_2 0.25) = 2$

Entropie

- $H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(vk)$

- Notons B(q), l'entropie d'une variable aléatoire binaire qui est vraie avec une probabilité q :

$$B(q) = -(q \log_2 q + (1 - q) \log_2 (1 - q))$$

- Par exemple :

$$H(PileOuFace) = B(0.5) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1 \text{ bit}$$

- Pour un arbre de décision, si un ensemble de données contient *p* exemples positifs et *n* exemples négatifs, l'entropie de la sortie de l'arbre de décision sur cet ensemble est:

$$H(Sortie) = B(\frac{p}{p+n})$$

Entropie d’un nœud parent et des enfants

- $B(q) = -(q \log_2 q + (1 - q) \log_2 (1 - q))$

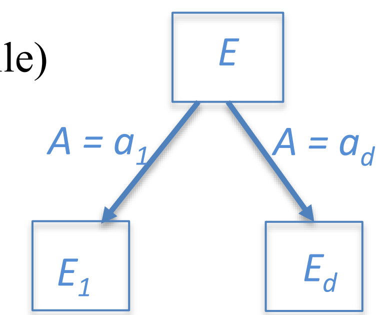
  Entropie pour une variable booléenne vraie avec une probabilité *q*

- $H(E_{pn}) = B(\frac{P}{P+n})$

  Entropie pour un **nœud parent**, avec *p* exemples positifs et *n* exemples négatives

- Un **attribue *a* avec *d* valeurs distinctes** sépare l’ensemble d’entrainement parent *(E)* en sous-ensembles enfants $E_1 \ldots E_d$.

- **L’entropie d’un enfant E_k** est $B(\frac{P_k}{P_k+n_k})$ (parce que E_k a p_k exemples positifs et n_k exemples négatifs). Autrement dit, si on choisit le $k^{ième}$ enfant, on aura besoin de $B(\frac{P_k}{P_k+n_k})$ bits supplémentaires pour arriver à un nœud pur (feuille)



- **La probabilité d’un enfant E_k est $(\frac{P_k + n_k}{P+n})$** (Intuitivement, un exemple choisi aléatoirement sera dans E_k (c.-à-d., aura la $k^{ème}$ valeur de l’attribut) avec cette probabilité)

- L’entropie de l’ensemble des enfants (c.à-.d, entropie restante après avoir testé *a*) est donc

$$Remainder(a) = \sum_{k=1}^{d} \mathrm{P}(E_k)\, \mathrm{H}(E_k) = \sum_{k=1}^{d} \left(\frac{P_k + n_k}{P+n}\right) B\left(\frac{P_k}{P_k + n_k}\right)$$

Gain d'information pour une variable

- $B(q) = -(q \log_2 q + (1-q) \log_2 (1-q))$ Entropie pour une variable booléenne vrai avec une probabilité *q*
- $H(E_{pn}) = B(\frac{p}{p+n})$ Entropie pour un **nœud parent E** ayant *p* exemples positifs et *n* exemples négatives
- L'entropie des **enfants** $\boldsymbol{E_k}$ (entropie restante après avoir testé A)

$$Remainder(a) = \sum_{k=1}^{d} \mathrm{P}(E_k)\, \mathrm{H}(E_k) = \sum_{k=1}^{d} (\frac{p_k + n_k}{p+n})\; B(\frac{p_k}{p_k + n_k})$$

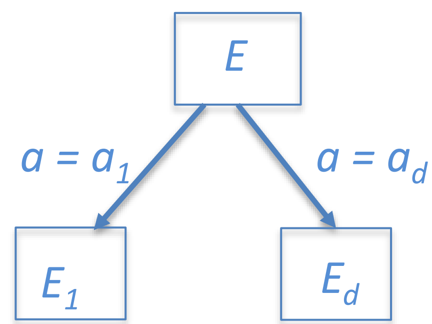
- Le gain d'information pour l'attribut A est

$$Gain\ (a) = \mathrm{H(Parent)} - Remainder(A)$$

$$= H(E) - \sum_{k=1}^{d} \mathrm{P}(E_k)\, \mathrm{H}(E_k)$$

$$= B(\frac{p}{p+n}) - \sum_{k=1}^{d} (\frac{p_k + n_k}{p+n})\; B(\frac{p_k}{p_k + n_k})$$

E a p exemples n et exemples négatifs
E_k a p_k exemples et n_k exemples négatifs



- La fonction *IMPORTANCE(a, examples)* retourne *Gain(a)*.
- L'algorithme choisit la variable qui donne le plus grand gain d'information.

Exemple

- $B(q) = -(q \log_2 q + (1-q)\log_2 (1-q))$
- $H(E_{pn}) = B(\frac{P}{P+n})$
- $Gain\ (A) = B(\frac{P}{P+n}) - \sum_{k=1}^{d} (\frac{P_k + n_k}{P+n})\ B(\frac{P_k}{P_k+n_k})$

Entropie du parent

Probabilité de l'enfant

Entropie de de l'enfant

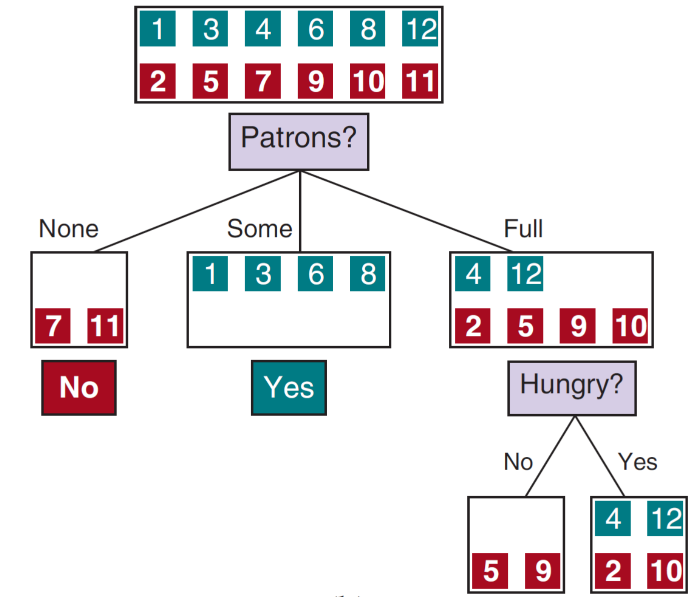
- $Gain\ (Patrons) =$

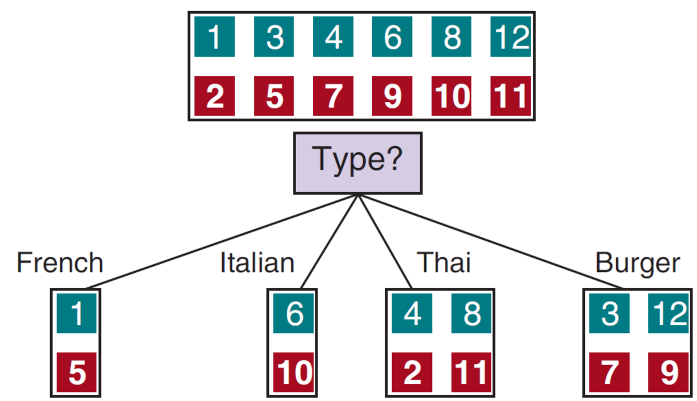
$$1\text{-}\ [\ \frac{2}{12} B(\frac{0}{2}) + \frac{4}{12} B(\frac{4}{4}) + \frac{6}{12} B(\frac{2}{6})\ ] \approx 0.541\ bits$$

- $Gain\ (Type) =$

$$1\text{-}\ [\ \frac{2}{12} B(\frac{1}{2}) + \frac{2}{12} B(\frac{1}{2}) + \frac{4}{12} B(\frac{2}{4}) + \frac{4}{12}\ B(\frac{2}{4})]$$

$$\approx 0\ bits$$





Expressivité des arbres de décision

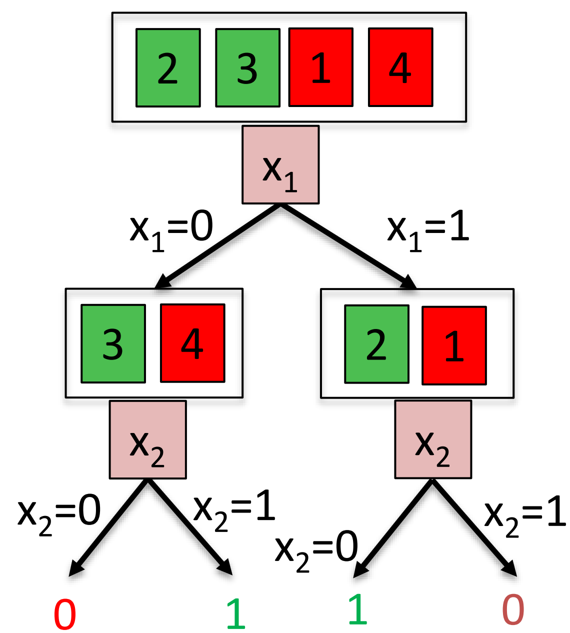
- Un arbre de décision binaire est équivalent à une formule propositionnelle sous la forme normale conjonctive : $Chemin_1 \wedge Chemin_2 \wedge \ldots$

$$Chemin_i \equiv [\ (x_1 = v_1 \wedge x_2 = v_2 \wedge \ldots) \rightarrow\ y = y_i \quad (x_i : \text{variable d'entrée};\ y_i : \text{variable cible})$$

$$\equiv \neg (x_1 = v_1)\ V \neg\ (x_2 = x_2)\ V \ldots\ V\ y = y_{i\ i}$$

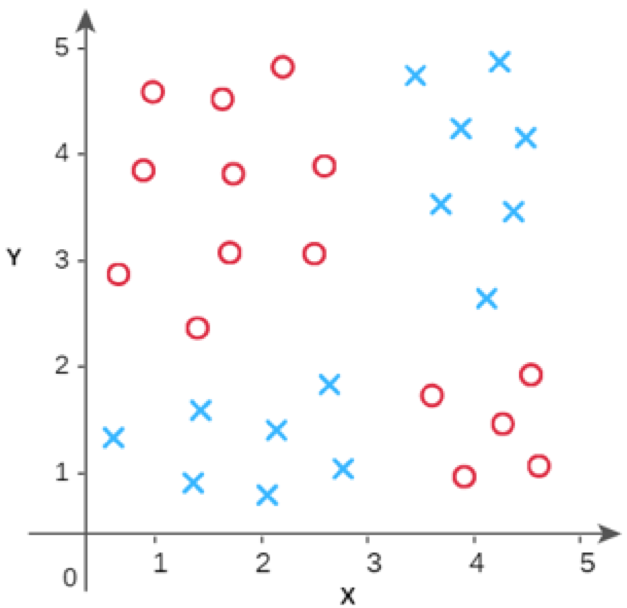
- Cela veut dire que toute formule de logique propositionnelle peut être exprimée par un arbre de décision
- Un arbre de décision est plus expressif qu'un perceptron

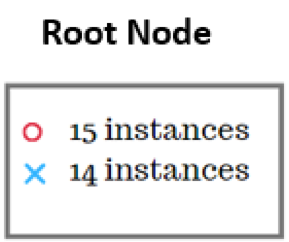
#	x_1	x_2	y
1	1	1	0
2	1	0	1
3	0	1	1
4	0	0	0



Exemple - Générique

- Itérativement
  - Choisir la **variable la plus importante** non encore choisie jusqu'ici
  - Lui appliquer un test sur ses valeurs
  - Pour chaque valeur du test
    - » Générer un enfant contenant les exemples consistents avec le test

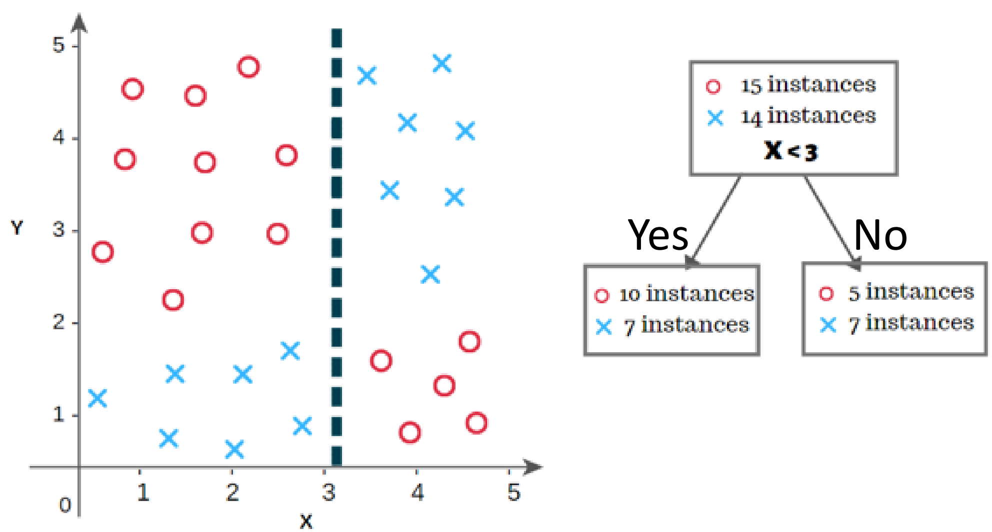




Source: [Amelia, Towards Data Science, 2019]

Exemple – Générique

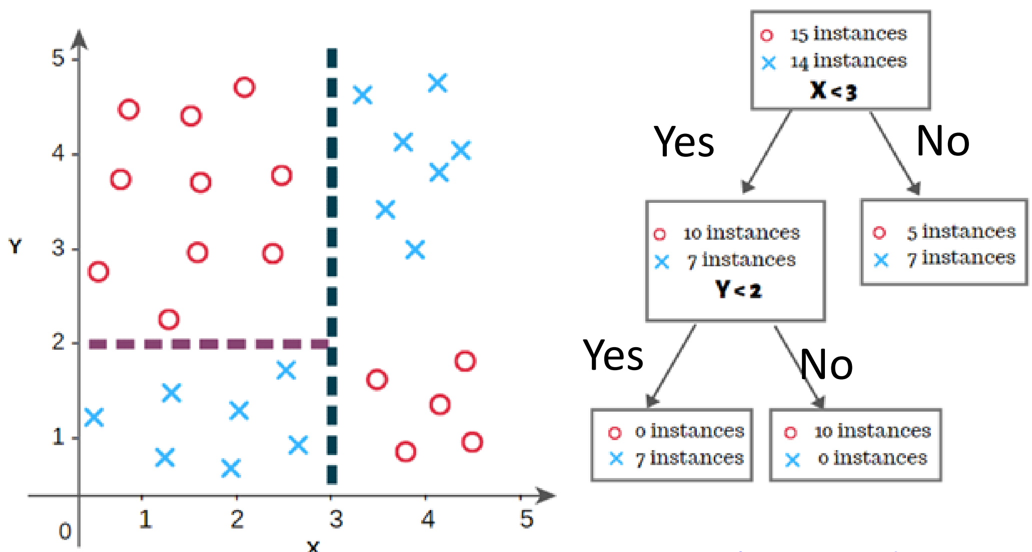
- Itérativement
  - Choisir la **variable la plus importante** non encore choisie jusqu'ici
  - Lui appliquer un test sur ses valeurs
  - Pour chaque valeur du test
    - » Générer un enfant contenant les exemples consistent avec le test



Source: [Amelia, Towards Data Science, 2019]

Exemple – Générique

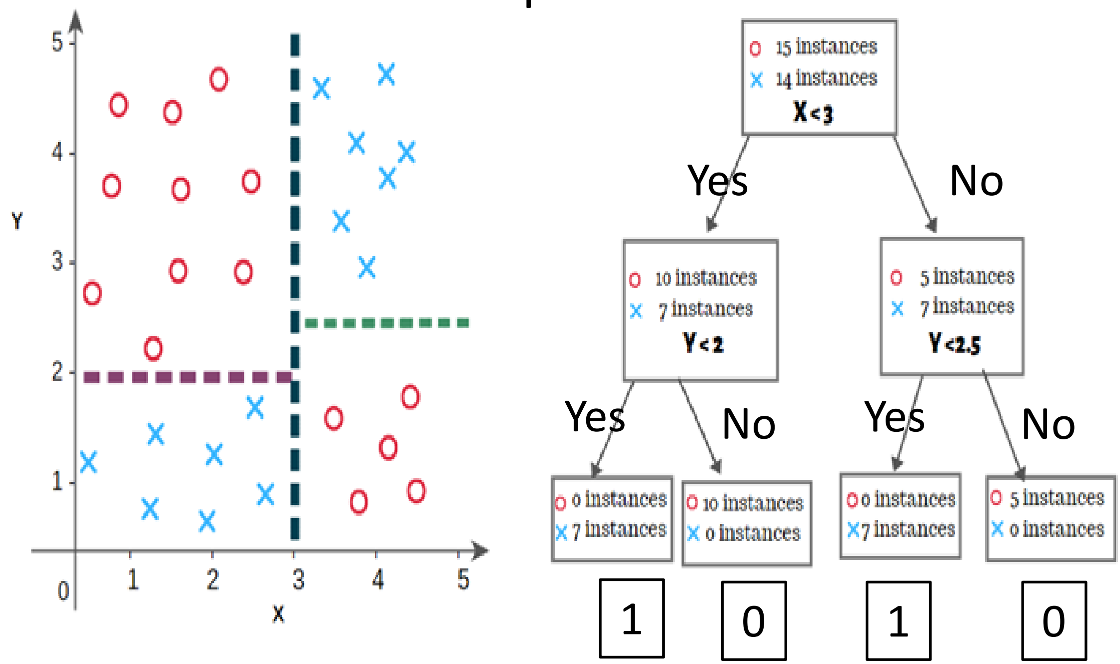
- Itérativement
  - Choisir la **variable la plus importante** non encore choisie jusqu'ici
  - Lui appliquer un test sur ses valeurs
  - Pour chaque valeur du test
    - » Générer un enfant contenant les exemples consistent avec le test



Source: [Amelia, Towards Data Science, 2019]

Exemple – Générique

- Itérativement
  - Choisir la **variable la plus importante** non encore choisie jusqu’ici
  - Lui appliquer un test sur ses valeurs
  - Pour chaque valeur du test
    - » Générer un enfant contenant les exemples consistent avec le test



Source: [Amelia, Towards Data Science, 2019]

Évitement du surapprentissage

Les arbres de décisions sont sensibles aux petites variations dans les données.

Pour réduire la variance du modèle (la variance mène au surapprentissage), une combinaison de plusieurs techniques peut être utilisées:

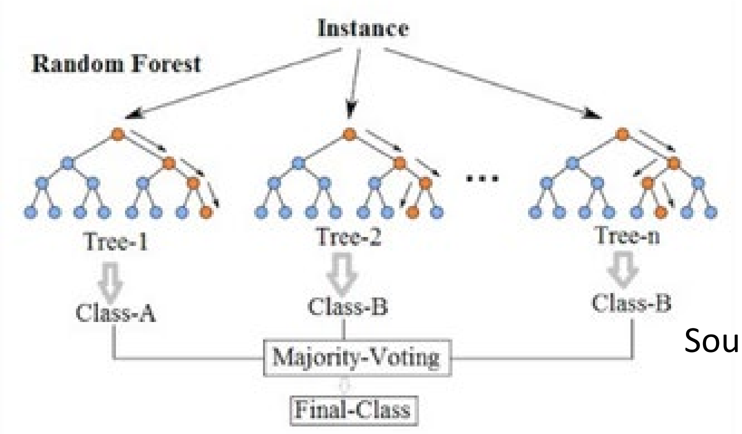
- **Élagage des nœuds** qui ne paraissent pas pertinents. Par exemple:
  - Le nombre minimum d’exemples qu’un nœud doit avoir
  - La profondeur limite de l’arbre
  - Un ratio entre la classe minoritaire et la classe majoritaire.

- **Réduction de la dimensionnalité** (e.g., en utilisant *PCA – Principal Component Analysis* – que nous ne voyons pas dans ce cours; pas couvert à l’examen)

- Les **forêts aléatoires** agrègent plusieurs petits arbres de décision.

Forêt aléatoires (*random forest*)

- Le ***random forest*** est un modèle dont la décision est une agrégation des décisions de plusieurs arbres de décision (***bagging***). Ces arbres sont générés à partir de données obtenus par échantillonnage avec remplacement du jeu de donnée (***bootstrapping***).



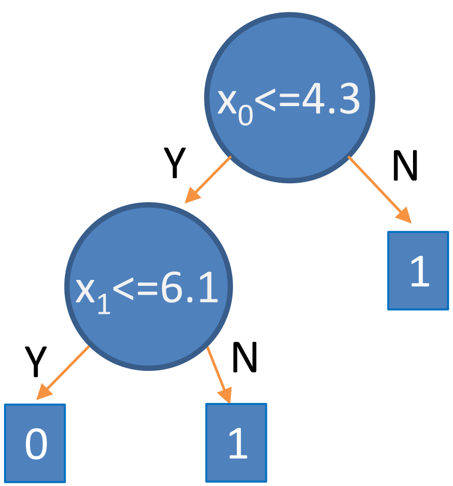
Source : [wikipedia]

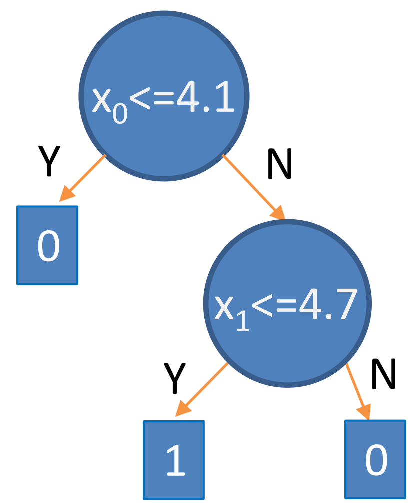
- C'est un cas particulier de ce qu'on appelle ***ensemble learning***. *Ensemble learning* est une famille de techniques qui consistent à agréger les décisions de plusieurs modèles.
- Le *random forest* est plus robuste qu'un arbre de décision seul.

Sensibilité à de petites variations

id	x0	x1	x2	x3	x4	y
0	4.3	4.9	4.1	4.7	5.5	0
1	3.9	6.1	5.9	5.5	5.9	0
2	2.7	4.8	4.1	5.0	5.6	0
3	6.6	4.4	4.5	3.9	5.9	1
4	6.5	2.9	4.7	4.6	6.1	1
5	2.7	6.7	4.2	5.3	4.8	1

id	x0	x1	x2	x3	x4	y
0	4.3	4.9	4.1	4.7	5.5	0
1	6.5	4.1	5.9	5.5	5.9	0
2	2.7	4.8	4.1	5.0	5.6	0
3	6.6	4.4	4.5	3.9	5.9	1
4	6.5	2.9	4.7	4.6	6.1	1
5	2.7	6.7	4.2	5.3	4.8	1





Source : [YouTube Normalized Nerd]

Algorithme Random Forest

1. **Bootstraping**. Construire plusieurs nouveaux jeux de données en *échantillonnant avec remplacement* du jeu de données de départ. Chaque jeu de données correspond à un sous-ensemble de variables. La taille du sous-ensemble est un hyper-paramètre (typiquement: racine-carrée ou logarithme naturel de la taille du jeu de donnée de départ)

2. **Bagging.** Agrégation des arbres de décision générés à partir de chaque jeu de données
   1. Pour chaque jeu de données, générer l'arbre décision correspondant

   2. Étant donnée une entrée, la sortie est l'aggrégation des sorties des arbres de décision: majorité pour la classification; moyenne pour la régression.

Bootstraping - Exemple

id	x0	x1	x2	x3	x4	y
0	4.3	4.9	4.1	4.7	5.5	0
1	3.9	6.1	5.9	5.5	5.9	0
2	2.7	4.8	4.1	5.0	5.6	0
3	6.6	4.4	4.5	3.9	5.9	1
4	6.5	2.9	4.7	4.6	6.1	1
5	2.7	6.7	4.2	5.3	4.8	1

id
2
0
2
4
5
5

x_o, x_1

id
2
1
3
1
4
4

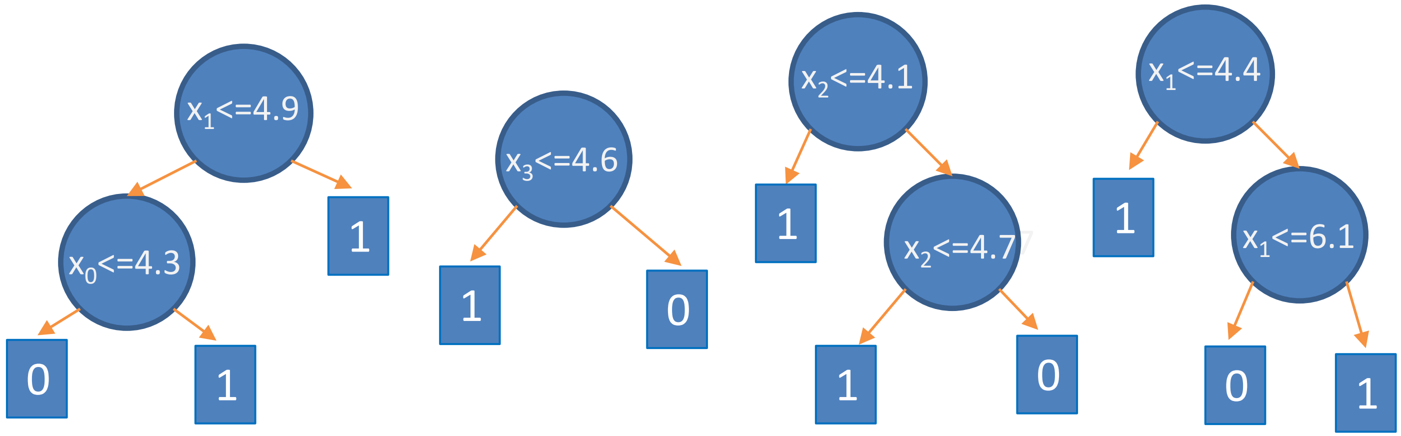
x_2, x_3

id
4
1
3
0
0
2

x_2, x_4

id
3
3
2
5
1
2

x_1, x_3



Source : [YouTube Normalized Nerd]

Bagging - Exemple

id	x0	x1	x2	x3	x4	y
0	4.3	4.9	4.1	4.7	5.5	0
1	3.9	6.1	5.9	5.5	5.9	0
2	2.7	4.8	4.1	5.0	5.6	0
3	6.6	4.4	4.5	3.9	5.9	1
4	6.5	2.9	4.7	4.6	6.1	1
5	2.7	6.7	4.2	5.3	4.8	1

6	**2.8**	**6.2**	**4.3**	**5.3**	**5.5**	**?** 1

id
2
0
2
4
5
5

x_o, x_1

id
2
1
3
1
4
4

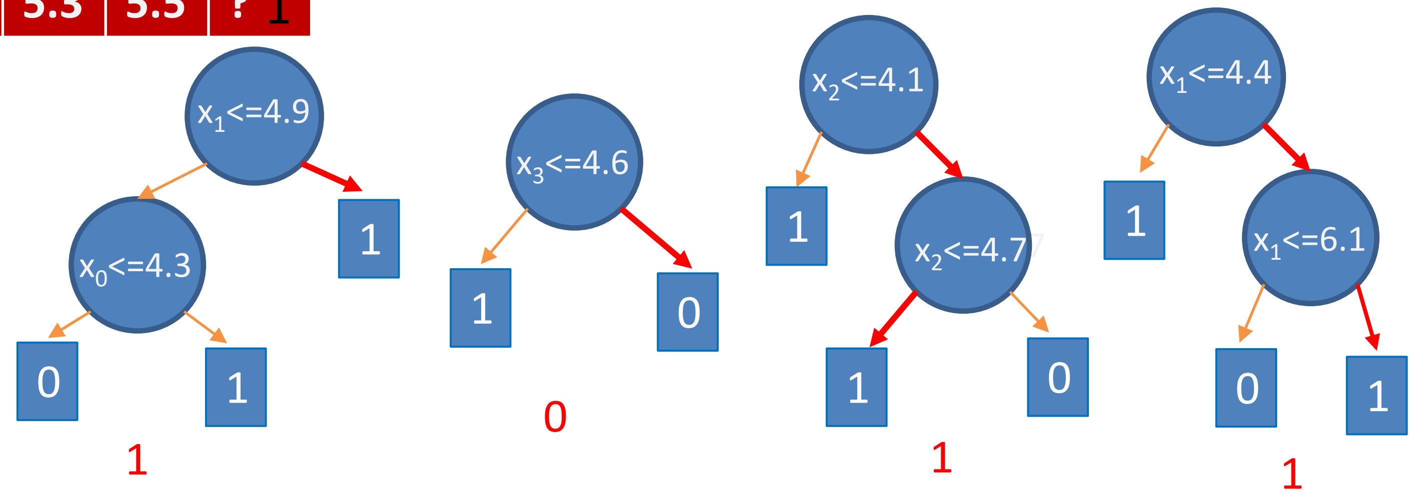
x_2, x_3

id
4
1
3
0
0
2

x_2, x_4

id
3
3
2
5
1
2

x_1, x_3



Source : YouTube Normalized Nerd

Apprentissage ensembliste (ensemble learning)

- L'apprentissage ensembliste consiste à choisir un ensemble de modèles $h_1, \ldots, h_n$ et de combiner leurs prédiction en faisant une moyenne, en faisant voter les modèles ou en utilisant une autre agrégation.

- On fait ça enfin de:
  - réduire le biais ou
  - réduire la variance

Apprentissage ensembliste (ensemble learning)

- **Bagging** consiste à
  - Générer K ensembles d'entrainement par échantillonnage avec replacement dans le jeu de données.
  - Pour chaque ensemble *k* des *K* ensembles, entrainer un modèle h_k . Tous les h_k utilisent le même type d'algorithme.
  - h(x) = $\frac{1}{K}\sum_{i=1}^{K} h_i(x)$

- A tendance à réduire le biais lorsqu'on a un petit jeu de données d'entrainement.

- On vient de le voir, Random *Forest* est une application de *bagging* aux arbres de décision.

Apprentissage ensembliste (ensemble learning)

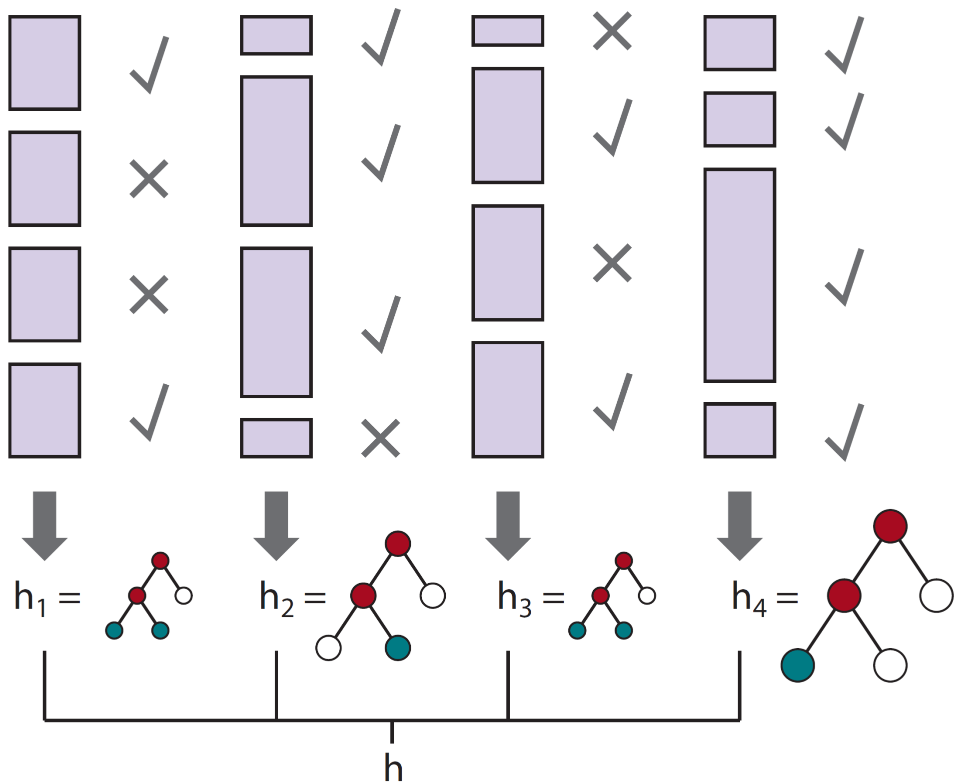
- **Stacking** consiste à
  - Entrainer K modèles différents (e.g., réseau de neurone + arbre de décision) sur le même jeu de données
  - h(x) = $\frac{1}{K}\sum_{i=1}^{K} h_i(x)$

Apprentissage ensembliste (ensemble learning)

- **Boosting** consiste à
  - À chaque exemple du jeu de données, on associe un poids.
  - Au départ, les poids sont les mêmes pour chaque exemple.
  - Pour *i* de 1 à *K* modèles :
    - » Générer un modèle h_i
    - » Mettre à jour les poids des exemples pour les biaiser en faveur des exemples mal classifiés
  - Chaque modèle reçoit un poids en fonction de sa performance.
  - $h(x) = \frac{1}{K}\sum_{i=1}^{K} z_i \, h_i(x)$, z_i étant le poids du modèle h_i

- *ADABOOST* est un algorithme très utilisé, basé sur cette technique. Il est utilisé le plus souvent avec les arbres de décision.

Illustration de *Boosting*



La taille de l'exemple illustre son poids relatif.
De même, la taille de l'arbre illustre son poid relatif

Optimisation d’hyper-paramètres

- Nous avons vu que les algorithmes d’apprentissage ont plusieurs hyperparamètres.

- Le choix des hyperparamètres est un problème d’optimisation en soi : recherche d’une configuration optimale des hyperparamètres.

- La fonction objective n’étant pas différentiable, on ne peut pas utiliser la descente du gradient.
  - Nous verrons des algorithmes appropriés plus tard dans le cours, lors que nous aborderons le thème de *recherche locale*, qui couvre, entre autres, les algorithmes génétiques.

Conclusion

- Un arbre de décision est un modèle d’apprentissage interprétable.

- L’algorithme d’apprentissage de base est instable et présente beaucoup de variance (surapprentissage).

- *Random Forest* – une technique d’apprentissage ensembliste avec des arbres de décision.

Vous devriez être capable de...

- Décrire ce qu’un arbre de décision.
- Décrire et simuler l’algorithme d’apprentissage d’un arbre de décision sur un exemple.
  - Expliquer et appliquer le calcul de l’entropie et du gain d’information pour choisir la prochaine variable durant l’algorithme d’apprentissage.
- Expliquer comment éviter le surapprentissage lors de l’apprentissage d’un arbre de décision
- Expliquer en grandes lignes en quoi l’algorithme *Random Forest* diffère de l’algorithme d’apprentissage d’arbre de décision de base

Sujets couverts par le cours

Concepts et algorithmes

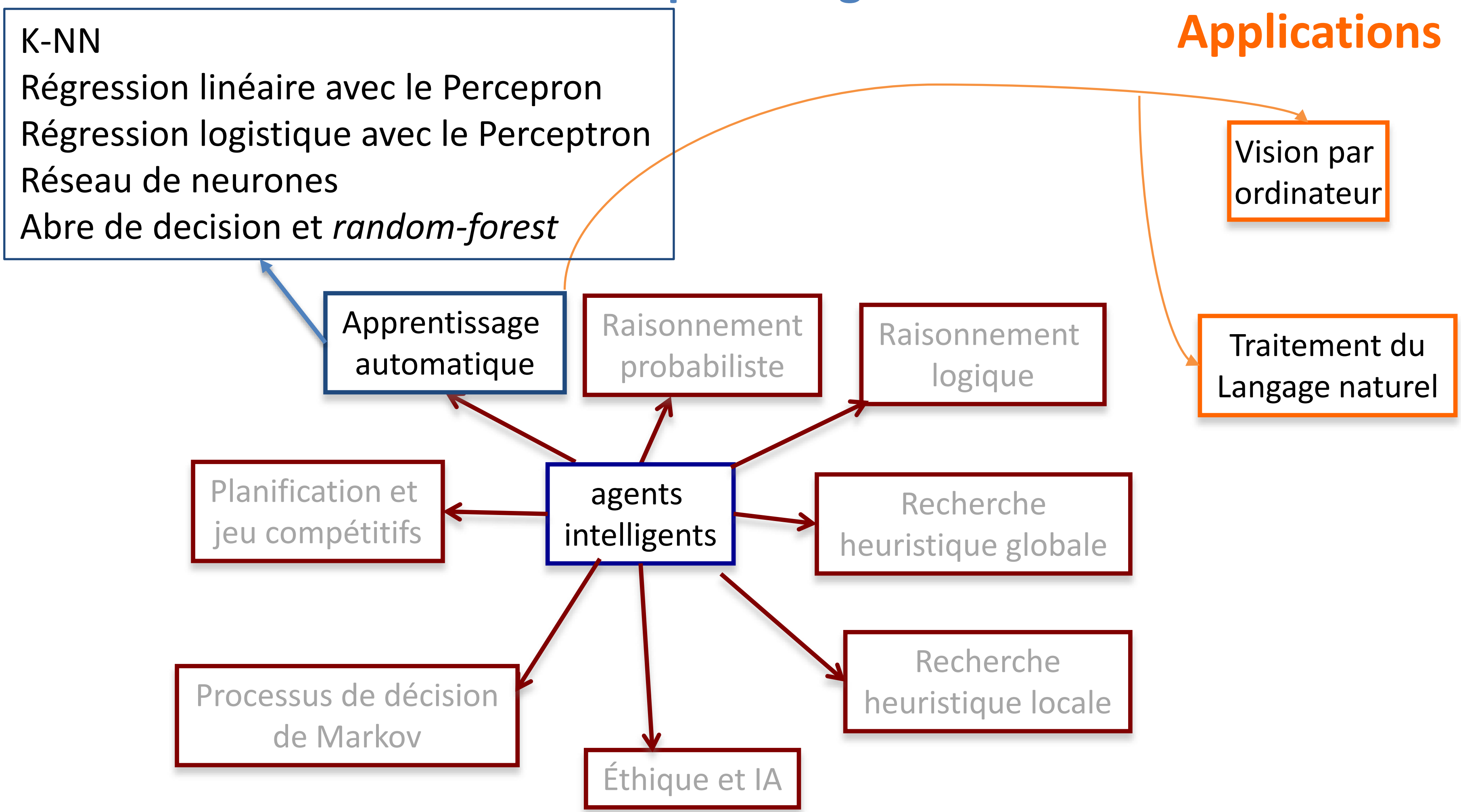
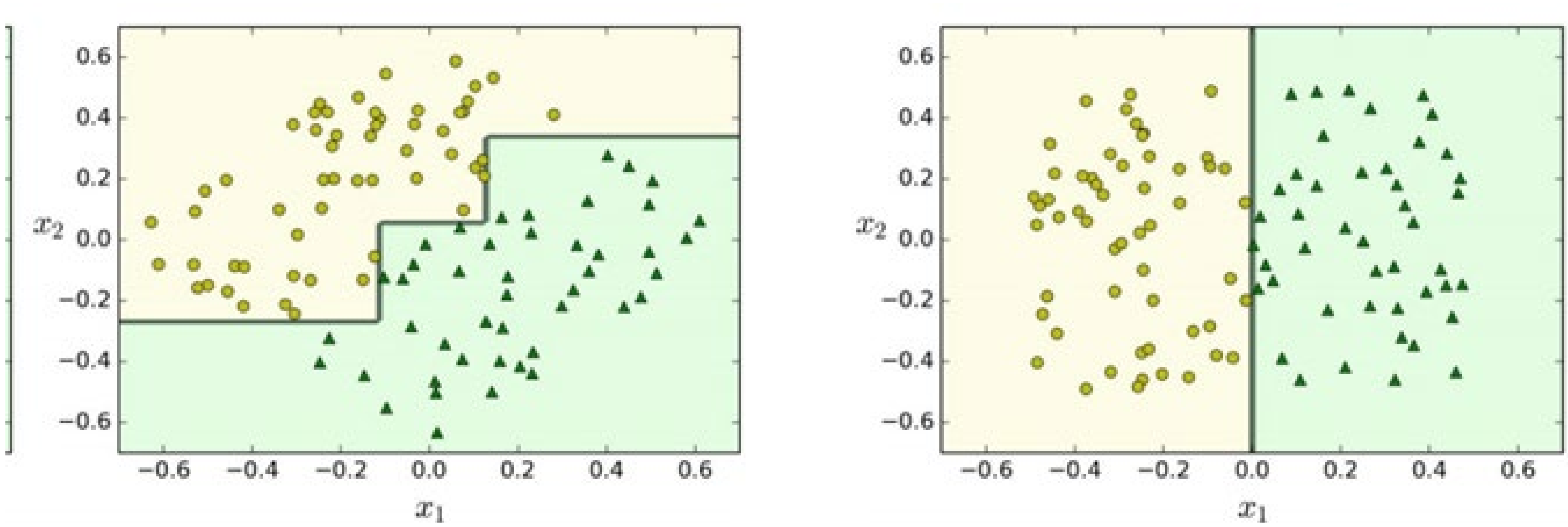


Illustration de la réduction de la dimensionalité

- Réduction de dimensionnalité (e.g., en utilisant *PCA – Principal Component Analysis* – que nous ne voyons pas dans ce cours)



Source: [Amelia, Towards Data Science, 2019]