

FORÊTS ALÉATOIRES

Jérémie Cabessa

Laboratoire DAVID, UVSQ

INTRODUCTION

- ▶ Les **forêts aléatoires** sont des modèles ensemblistes (*ensemble methods*) basés sur les arbres de décision.
- ▶ L'idée est de combiner plusieurs arbres de décision afin d'obtenir un modèle plus robuste, i.e., avec une variance réduite.
- ▶ La variance d'un modèle représente sa volatilité face à un changement de dataset provenant d'une même distribution.

INTRODUCTION

- ▶ Les **forêts aléatoires** sont des modèles ensemblistes (*ensemble methods*) basés sur les arbres de décision.
- ▶ L'idée est de combiner plusieurs arbres de décision afin d'obtenir un modèle plus robuste, i.e., avec une variance réduite.
- ▶ La variance d'un modèle représente sa volatilité face à un changement de dataset provenant d'une même distribution.

INTRODUCTION

- ▶ Les **forêts aléatoires** sont des modèles ensemblistes (*ensemble methods*) basés sur les arbres de décision.
- ▶ L'idée est de combiner plusieurs arbres de décision afin d'obtenir un modèle plus robuste, i.e., avec une variance réduite.
- ▶ La variance d'un modèle représente sa volatilité face à un changement de dataset provenant d'une même distribution.

UN RÉSULTAT DE STATISTIQUE

- Soient $\hat{f}_1, \dots, \hat{f}_B$ des modèles qui sont tous de variance σ^2 .
Alors le *modèle moyen*

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_i$$

a une variance réduite de $\frac{\sigma^2}{B}$.

- Preuve:

$$\text{Var} \left(\frac{1}{B} \sum_{b=1}^B \hat{f}_i \right) = \frac{1}{B^2} \text{Var} \left(\sum_{b=1}^B \hat{f}_i \right) = \frac{1}{B^2} \sum_{b=1}^B \text{Var}(\hat{f}_i) = \frac{B\sigma^2}{B^2} = \frac{\sigma^2}{B}$$

- Ainsi, en prenant la moyenne de plusieurs modèles, on obtient un modèle ensembliste de variance réduite.

UN RÉSULTAT DE STATISTIQUE

- ▶ Soient $\hat{f}_1, \dots, \hat{f}_B$ des modèles qui sont tous de variance σ^2 .
Alors le *modèle moyen*

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_i$$

a une variance réduite de $\frac{\sigma^2}{B}$.

- ▶ Preuve:

$$\text{Var} \left(\frac{1}{B} \sum_{b=1}^B \hat{f}_i \right) = \frac{1}{B^2} \text{Var} \left(\sum_{b=1}^B \hat{f}_i \right) = \frac{1}{B^2} \sum_{b=1}^B \text{Var} \left(\hat{f}_i \right) = \frac{B\sigma^2}{B^2} = \frac{\sigma^2}{B}$$

- ▶ Ainsi, en prenant la moyenne de plusieurs modèles, on obtient un modèle ensembliste de variance réduite.

UN RÉSULTAT DE STATISTIQUE

- ▶ Soient $\hat{f}_1, \dots, \hat{f}_B$ des modèles qui sont tous de variance σ^2 .
Alors le *modèle moyen*

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b$$

a une variance réduite de $\frac{\sigma^2}{B}$.

- ▶ Preuve:

$$\text{Var} \left(\frac{1}{B} \sum_{b=1}^B \hat{f}_b \right) = \frac{1}{B^2} \text{Var} \left(\sum_{b=1}^B \hat{f}_b \right) = \frac{1}{B^2} \sum_{b=1}^B \text{Var} (\hat{f}_b) = \frac{B\sigma^2}{B^2} = \frac{\sigma^2}{B}$$

- ▶ Ainsi, en prenant la moyenne de plusieurs modèles, on obtient un modèle ensembliste de variance réduite.

BAGGING

- La technique du **bagging** (ou **bootstrap aggregation**) se base sur ce résultat.

Soit un dataset $\mathcal{S} = \{(x_i, y_i) : i = 1, \dots, N\}$ de taille N .

Pour $b = 1, \dots, B$

On sample avec remplacement N éléments de \mathcal{S} pour créer un nouveau dataset \mathcal{S}_b de taille N .

On entraîne un modèle \hat{f}_b sur le dataset \mathcal{S}_b .

On définit le modèle $\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b$.

BAGGING

- La technique du **bagging** (ou **bootstrap aggregation**) se base sur ce résultat.

Soit un dataset $\mathcal{S} = \{(x_i, y_i) : i = 1, \dots, N\}$ de taille N .

Pour $b = 1, \dots, B$

On sample avec remplacement N éléments de \mathcal{S} pour créer un nouveau dataset \mathcal{S}_b de taille N .

On entraîne un modèle \hat{f}_b sur le dataset \mathcal{S}_b .

On définit le modèle $\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b$.

BAGGING

- La technique du **bagging** (ou **bootstrap aggregation**) se base sur ce résultat.

Soit un dataset $\mathcal{S} = \{(x_i, y_i) : i = 1, \dots, N\}$ de taille N .

Pour $b = 1, \dots, B$

On sample avec remplacement N éléments de \mathcal{S} pour créer un nouveau dataset \mathcal{S}_b de taille N .

On entraîne un modèle \hat{f}_b sur le dataset \mathcal{S}_b .

On définit le modèle $\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b$.

BAGGING

- La technique du **bagging** (ou **bootstrap aggregation**) se base sur ce résultat.

Soit un dataset $\mathcal{S} = \{(x_i, y_i) : i = 1, \dots, N\}$ de taille N .

Pour $b = 1, \dots, B$

On sample avec remplacement N éléments de \mathcal{S} pour créer un nouveau dataset \mathcal{S}_b de taille N .

On entraîne un modèle \hat{f}_b sur le dataset \mathcal{S}_b .

On définit le modèle $\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b$.

BAGGING

- La technique du **bagging** (ou **bootstrap aggregation**) se base sur ce résultat.

Soit un dataset $\mathcal{S} = \{(x_i, y_i) : i = 1, \dots, N\}$ de taille N .

Pour $b = 1, \dots, B$

On sample avec remplacement N éléments de \mathcal{S} pour créer un nouveau dataset \mathcal{S}_b de taille N .

On entraîne un modèle \hat{f}_b sur le dataset \mathcal{S}_b .

On définit le modèle $\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b$.

BAGGING

- La technique du **bagging** (ou **bootstrap aggregation**) se base sur ce résultat.

Soit un dataset $\mathcal{S} = \{(x_i, y_i) : i = 1, \dots, N\}$ de taille N .

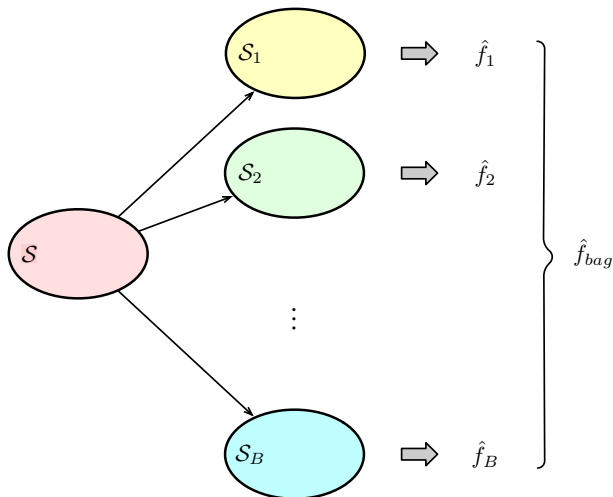
Pour $b = 1, \dots, B$

On sample avec remplacement N éléments de \mathcal{S} pour créer un nouveau dataset \mathcal{S}_b de taille N .

On entraîne un modèle \hat{f}_b sur le dataset \mathcal{S}_b .

On définit le modèle $\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b$.

BAGGING



BAGGING (RÉGRESSION)

- Dans le cas d'une régression, on a donc le modèle

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b$$

- Pour toute data x , on sa prédiction est donnée par:

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(x)$$

BAGGING (RÉGRESSION)

- Dans le cas d'une régression, on a donc le modèle

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b$$

- Pour toute data \mathbf{x} , on sa prédiction est donnée par:

$$\hat{f}_{bag}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(\mathbf{x})$$

BAGGING (CLASSIFICATION)

- ▶ Dans le cas d'une classification, on définit \hat{f}_{bag} par le principe du *vote majoritaire* (*majority vote*) des $\hat{f}_1, \dots, \hat{f}_B$:

$$\hat{f}_{bag} = \text{MajorityVote}([\hat{f}_1, \dots, \hat{f}_B])$$

- ▶ Cela signifie que pour toute data x , la prédiction $\hat{f}_{bag}(x)$ est la valeur qui apparaît le plus souvent parmi $\hat{f}_1(x), \dots, \hat{f}_B(x)$:

$$\hat{f}_{bag}(x) = \text{most_freq_value}([\hat{f}_1(x), \dots, \hat{f}_B(x)])$$

BAGGING (CLASSIFICATION)

- ▶ Dans le cas d'une classification, on définit \hat{f}_{bag} par le principe du *vote majoritaire* (*majority vote*) des $\hat{f}_1, \dots, \hat{f}_B$:

$$\hat{f}_{bag} = \text{MajorityVote}([\hat{f}_1, \dots, \hat{f}_B])$$

- ▶ Cela signifie que pour toute data \mathbf{x} , la prédiction $\hat{f}_{bag}(\mathbf{x})$ est la valeur qui apparaît le plus souvent parmi $\hat{f}_1(\mathbf{x}), \dots, \hat{f}_B(\mathbf{x})$:

$$\hat{f}_{bag}(\mathbf{x}) = \text{most_freq_value}([\hat{f}_1(\mathbf{x}), \dots, \hat{f}_B(\mathbf{x})])$$

FORÊT ALÉATOIRE

- ▶ Une **forêt aléatoire (random forest)** est un modèle obtenu à partir de plusieurs *arbres de décision*.
- ▶ On utilise une technique de *bagging* améliorée pour *agréger* et *décorrélér* les arbres de décision utilisés.
- ▶ La technique de *décorrélation* des arbres de décision a pour but de diminuer la variance du modèle final.

FORÊT ALÉATOIRE

- ▶ Une **forêt aléatoire** (**random forest**) est un modèle obtenu à partir de plusieurs *arbres de décision*.
- ▶ On utilise une technique de *bagging* améliorée pour *agréger* et *décorrélér* les arbres de décision utilisés.
- ▶ La technique de *décorrélation* des arbres de décision a pour but de diminuer la variance du modèle final.

FORÊT ALÉATOIRE

- ▶ Une **forêt aléatoire** (**random forest**) est un modèle obtenu à partir de plusieurs *arbres de décision*.
- ▶ On utilise une technique de *bagging* améliorée pour *agréger* et *décorréliser* les arbres de décision utilisés.
- ▶ La technique de *décorrélation* des arbres de décision a pour but de diminuer la variance du modèle final.

FORÊT ALÉATOIRE

- ▶ Une **forêt aléatoire** (**random forest**) est un modèle \hat{f}_{bag} obtenu par bagging à partir d'arbres de décision $\hat{f}_1, \dots, \hat{f}_B$.
- ▶ Mais, lors de la construction de chaque \hat{f}_i , on modifie la méthode qui détermine le best split " $X_m \leq s$ " comme suit:

À chaque split, on sélectionne un sous-ensemble aléatoire de q features X_{i_1}, \dots, X_{i_q} parmi X_1, \dots, X_p comme candidats au "best split" (on peut prendre $q \sim \sqrt{p}$).

- ▶ Les arbres \hat{f}_i auront alors tendance à être différents les uns des autres, donc décorrélés, ce qui diminue la variance de \hat{f}_{bag} .

FORÊT ALÉATOIRE

- ▶ Une **forêt aléatoire** (**random forest**) est un modèle \hat{f}_{bag} obtenu par bagging à partir d'arbres de décision $\hat{f}_1, \dots, \hat{f}_B$.
- ▶ Mais, lors de la construction de chaque \hat{f}_i , on modifie la méthode qui détermine le best split " $X_m \leq s$ " comme suit:

À chaque split, on sélectionne un sous-ensemble aléatoire de q features X_{i_1}, \dots, X_{i_q} parmi X_1, \dots, X_p comme candidats au "best split" (on peut prendre $q \sim \sqrt{p}$).

- ▶ Les arbres \hat{f}_i auront alors tendance à être différents les uns des autres, donc décorrélés, ce qui diminue la variance de \hat{f}_{bag} .

FORÊT ALÉATOIRE

- ▶ Une **forêt aléatoire** (**random forest**) est un modèle \hat{f}_{bag} obtenu par bagging à partir d'arbres de décision $\hat{f}_1, \dots, \hat{f}_B$.
- ▶ Mais, lors de la construction de chaque \hat{f}_i , on modifie la méthode qui détermine le best split " $X_m \leq s$ " comme suit:

À chaque split, on sélectionne un sous-ensemble aléatoire de q features X_{i_1}, \dots, X_{i_q} parmi X_1, \dots, X_p comme candidats au "best split" (on peut prendre $q \sim \sqrt{p}$).

- ▶ Les arbres \hat{f}_i auront alors tendance à être différents les uns des autres, donc décorrélés, ce qui diminue la variance de \hat{f}_{bag} .

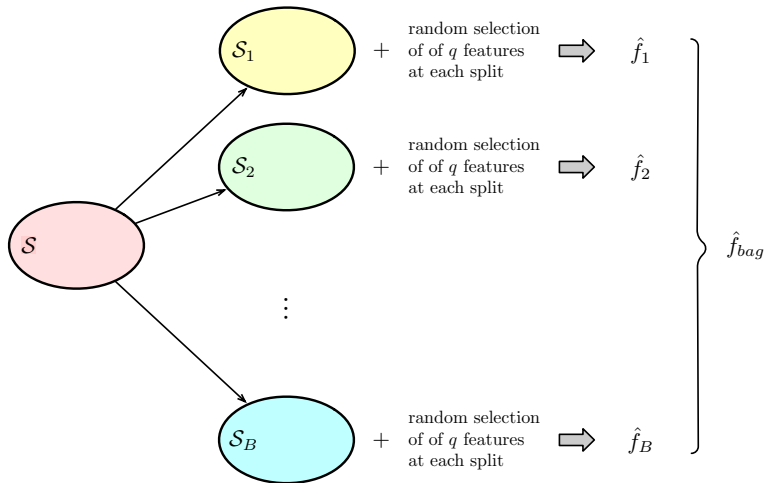
FORÊT ALÉATOIRE

- ▶ Une **forêt aléatoire** (**random forest**) est un modèle \hat{f}_{bag} obtenu par bagging à partir d'arbres de décision $\hat{f}_1, \dots, \hat{f}_B$.
- ▶ Mais, lors de la construction de chaque \hat{f}_i , on modifie la méthode qui détermine le best split " $X_m \leq s$ " comme suit:

À chaque split, on sélectionne un sous-ensemble aléatoire de q features X_{i_1}, \dots, X_{i_q} parmi X_1, \dots, X_p comme candidats au "best split" (on peut prendre $q \sim \sqrt{p}$).

- ▶ Les arbres \hat{f}_i auront alors tendance à être différents les uns des autres, donc décorrélés, ce qui diminue la variance de \hat{f}_{bag} .

FORÊT ALÉATOIRE



FORÊT ALÉATOIRE

- ▶ **Best split:** algorithme modifié de recherche d'un best split d'une region R d'un dataset.
 - On tire aléatoirement q features X_{i_1}, \dots, X_{i_q} parmi les p features X_1, \dots, X_p .
 - Parmi ces features, on test tous les splits possibles (m, s) de R en R_1 et R_2 .
 - On garde celui qui est associé à la plus grande réduction d'entropie $\text{EntRed}(R, R_1, R_2)$.

FORÊT ALÉATOIRE

- ▶ **Best split:** algorithme modifié de recherche d'un best split d'une region R d'un dataset.
- On tire aléatoirement q features X_{i_1}, \dots, X_{i_q} parmi les p features X_1, \dots, X_p .
- Parmi ces features, on test tous les splits possibles (m, s) de R en R_1 et R_2 .
- On garde celui qui est associé à la plus grande réduction d'entropie $\text{EntRed}(R, R_1, R_2)$.

FORÊT ALÉATOIRE

- ▶ **Best split:** algorithme modifié de recherche d'un best split d'une region R d'un dataset.
- On tire aléatoirement q features X_{i_1}, \dots, X_{i_q} parmi les p features X_1, \dots, X_p .
- Parmi ces features, on test tous les splits possibles (m, s) de R en R_1 et R_2 .
- On garde celui qui est associé à la plus grande réduction d'entropie $\text{EntRed}(R, R_1, R_2)$.

FORÊT ALÉATOIRE

- ▶ **Best split:** algorithme modifié de recherche d'un best split d'une region R d'un dataset.
- On tire aléatoirement q features X_{i_1}, \dots, X_{i_q} parmi les p features X_1, \dots, X_p .
- Parmi ces features, on test tous les splits possibles (m, s) de R en R_1 et R_2 .
- On garde celui qui est associé à la plus grande réduction d'entropie $\text{EntRed}(R, R_1, R_2)$.

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: best_split(dataset)

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: best_split(dataset)

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: best_split(dataset)

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: best_split(dataset)

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: best_split(dataset)

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: best_split(dataset)

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: best_split(dataset)

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: best_split(dataset)

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: best_split(dataset)

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: `best_split(dataset)`

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: `best_split(dataset)`

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: MODIFIED BEST SPLIT

Algorithm 1: best_split(dataset)

```
best_info_gain =  $-\infty$ 
selected_features = sample(1, p, q)
for m in selected_features do
    for s in possible_values( $X_m$ ) do
        split = (m, s)
        data_l, data_r = split_data(dataset, split)
        if data_l and data_r not empty then
            info_gain = information_gain(dataset, data_l, data_r)
            if info_gain > best_info_gain then
                best_split = split
                best_info_gain = info_gain
            end
        end
    end
end
return best_split
```

FORÊT ALÉATOIRE: RECURSIVE BINARY SPLITTING

Algorithm 2: `build_tree(dataset, depth = 0)`

```
num_samples = len(dataset)
if num_samples ≥ min_samples and depth ≤ max_depth then
    split = best_split(dataset)
    if split[info_gain] ≥ 0 then
        subtree_left = build_tree(split[dataset_left], depth + 1)
        subtree_right = build_tree(split[dataset_right], depth + 1)
        return DecTree(split, subtree_left, subtree_right)
else
    value = leaf_value(dataset)
    return DecTree(value)
```

FORÊT ALÉATOIRE: RECURSIVE BINARY SPLITTING

Algorithm 2: `build_tree(dataset, depth = 0)`

```
num_samples = len(dataset)
if num_samples  $\geq$  min_samples and depth  $\leq$  max_depth then
    split = best_split(dataset)
    if split[info_gain]  $\geq$  0 then
        subtree_left = build_tree(split[dataset_left], depth + 1)
        subtree_right = build_tree(split[dataset_right], depth + 1)
        return DecTree(split, subtree_left, subtree_right)
    else
        value = leaf_value(dataset)
        return DecTree(value)
```

FORÊT ALÉATOIRE: RECURSIVE BINARY SPLITTING

Algorithm 2: `build_tree(dataset, depth = 0)`

```
num_samples = len(dataset)
if num_samples  $\geq$  min_samples and depth  $\leq$  max_depth then
    split = best_split(dataset)
    if split[info_gain]  $\geq$  0 then
        subtree_left = build_tree(split[dataset_left], depth + 1)
        subtree_right = build_tree(split[dataset_right], depth + 1)
        return DecTree(split, subtree_left, subtree_right)
    else
        value = leaf_value(dataset)
        return DecTree(value)
```

FORÊT ALÉATOIRE: RECURSIVE BINARY SPLITTING

Algorithm 2: `build_tree(dataset, depth = 0)`

```
num_samples = len(dataset)
if num_samples  $\geq$  min_samples and depth  $\leq$  max_depth then
    split = best_split(dataset)
    if split[info_gain]  $\geq$  0 then
        subtree_left = build_tree(split[dataset_left], depth + 1)
        subtree_right = build_tree(split[dataset_right], depth + 1)
        return DecTree(split, subtree_left, subtree_right)
    else
        value = leaf_value(dataset)
        return DecTree(value)
```

FORÊT ALÉATOIRE: RECURSIVE BINARY SPLITTING

Algorithm 2: `build_tree(dataset, depth = 0)`

```
num_samples = len(dataset)
if num_samples  $\geq$  min_samples and depth  $\leq$  max_depth then
    split = best_split(dataset)
    if split[info_gain]  $\geq$  0 then
        subtree_left = build_tree(split[dataset_left], depth + 1)
        subtree_right = build_tree(split[dataset_right], depth + 1)
        return DecTree(split, subtree_left, subtree_right)
else
    value = leaf_value(dataset)
    return DecTree(value)
```

FORÊT ALÉATOIRE: RECURSIVE BINARY SPLITTING

Algorithm 2: `build_tree(dataset, depth = 0)`

```
num_samples = len(dataset)
if num_samples  $\geq$  min_samples and depth  $\leq$  max_depth then
    split = best_split(dataset)
    if split[info_gain]  $\geq$  0 then
        subtree_left = build_tree(split[dataset_left], depth + 1)
        subtree_right = build_tree(split[dataset_right], depth + 1)
        return DecTree(split, subtree_left, subtree_right)
else
    value = leaf_value(dataset)
    return DecTree(value)
```

FORÊT ALÉATOIRE: RECURSIVE BINARY SPLITTING

Algorithm 2: `build_tree(dataset, depth = 0)`

```
num_samples = len(dataset)
if num_samples  $\geq$  min_samples and depth  $\leq$  max_depth then
    split = best_split(dataset)
    if split[info_gain]  $\geq$  0 then
        subtree_left = build_tree(split[dataset_left], depth + 1)
        subtree_right = build_tree(split[dataset_right], depth + 1)
        return DecTree(split, subtree_left, subtree_right)
else
    value = leaf_value(dataset)
    return DecTree(value)
```

FORÊT ALÉATOIRE: RECURSIVE BINARY SPLITTING

Algorithm 2: `build_tree(dataset, depth = 0)`

```
num_samples = len(dataset)
if num_samples  $\geq$  min_samples and depth  $\leq$  max_depth then
    split = best_split(dataset)
    if split[info_gain]  $\geq$  0 then
        subtree_left = build_tree(split[dataset_left], depth + 1)
        subtree_right = build_tree(split[dataset_right], depth + 1)
        return DecTree(split, subtree_left, subtree_right)
else
    value = leaf_value(dataset)
    return DecTree(value)
```

FORÊT ALÉATOIRE: RECURSIVE BINARY SPLITTING

Algorithm 2: `build_tree(dataset, depth = 0)`

```
num_samples = len(dataset)
if num_samples  $\geq$  min_samples and depth  $\leq$  max_depth then
    split = best_split(dataset)
    if split[info_gain]  $\geq$  0 then
        subtree_left = build_tree(split[dataset_left], depth + 1)
        subtree_right = build_tree(split[dataset_right], depth + 1)
        return DecTree(split, subtree_left, subtree_right)
else
    value = leaf_value(dataset)
    return DecTree(value)
```

FORÊT ALÉATOIRE: BAGGING

Algorithm 3: bagging(dataset)

```
n = len(dataset)
dec_trees_l = []
for b = 1 to B do
    | dataset_new = sample(dataset, n)
    | dec_tree = build_tree(dataset_new, depth = 0)
    | dec_trees_l.append(dec_tree)
end
RandomForest = aggregate(dec_trees_l)
return RandomForest
```

FORÊT ALÉATOIRE: BAGGING

Algorithm 3: bagging(dataset)

```
n = len(dataset)
dec_trees_l = []
for b = 1 to B do
    dataset_new = sample(dataset, n)
    dec_tree = build_tree(dataset_new, depth = 0)
    dec_trees_l.append(dec_tree)
end
RandomForest = aggregate(dec_trees_l)
return RandomForest
```

FORÊT ALÉATOIRE: BAGGING

Algorithm 3: bagging(dataset)

```
n = len(dataset)
dec_trees_l = []
for b = 1 to B do
    | dataset_new = sample(dataset, n)
    | dec_tree = build_tree(dataset_new, depth = 0)
    | dec_trees_l.append(dec_tree)
end
RandomForest = aggregate(dec_trees_l)
return RandomForest
```

FORÊT ALÉATOIRE: BAGGING

Algorithm 3: bagging(dataset)

```
n = len(dataset)
dec_trees_l = []
for b = 1 to B do
    | dataset_new = sample(dataset, n)
    | dec_tree = build_tree(dataset_new, depth = 0)
    | dec_trees_l.append(dec_tree)
end
RandomForest = aggregate(dec_trees_l)
return RandomForest
```

FORÊT ALÉATOIRE: BAGGING

Algorithm 3: bagging(dataset)

```
n = len(dataset)
dec_trees_l = []
for b = 1 to B do
    | dataset_new = sample(dataset, n)
    | dec_tree = build_tree(dataset_new, depth = 0)
    | dec_trees_l.append(dec_tree)
end
RandomForest = aggregate(dec_trees_l)
return RandomForest
```

FORÊT ALÉATOIRE: BAGGING

Algorithm 3: bagging(dataset)

```
n = len(dataset)
dec_trees_l = []
for b = 1 to B do
    | dataset_new = sample(dataset, n)
    | dec_tree = build_tree(dataset_new, depth = 0)
    | dec_trees_l.append(dec_tree)
end
RandomForest = aggregate(dec_trees_l)
return RandomForest
```

FORÊT ALÉATOIRE: BAGGING

Algorithm 3: bagging(dataset)

```
n = len(dataset)
dec_trees_l = []
for b = 1 to B do
    | dataset_new = sample(dataset, n)
    | dec_tree = build_tree(dataset_new, depth = 0)
    | dec_trees_l.append(dec_tree)
end
RandomForest = aggregate(dec_trees_l)
return RandomForest
```

FORÊT ALÉATOIRE: BAGGING

Algorithm 3: bagging(dataset)

```
n = len(dataset)
dec_trees_l = []
for b = 1 to B do
    dataset_new = sample(dataset, n)
    dec_tree = build_tree(dataset_new, depth = 0)
    dec_trees_l.append(dec_tree)
end
RandomForest = aggregate(dec_trees_l)
return RandomForest
```

CONCLUSION

- ▶ Les modèles ensemblistes peuvent donner des résultats très intéressants.
- ▶ Les techniques de bagging et les forêts aléatoires améliorent grandement les arbres de décision.
- ▶ Il existe d'autres technique d'agrégation pour des arbres de décision: le *boosting* et les *bayesian additive regression trees (BART)*.

CONCLUSION

- ▶ Les modèles ensemblistes peuvent donner des résultats très intéressants.
- ▶ Les techniques de bagging et les forêts aléatoires améliorent grandement les arbres de décision.
- ▶ Il existe d'autres technique d'agrégation pour des arbres de décision: le *boosting* et les *bayesian additive regression trees (BART)*.

CONCLUSION

- ▶ Les modèles ensemblistes peuvent donner des résultats très intéressants.
- ▶ Les techniques de bagging et les forêts aléatoires améliorent grandement les arbres de décision.
- ▶ Il existe d'autres technique d'agrégation pour des arbres de décision: le *boosting* et les *bayesian additive regression trees (BART)*.

BIBLIOGRAPHIE



James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013).

An Introduction to Statistical Learning: with Applications in R, volume 103 of *Springer Texts in Statistics*.

Springer, New York.