

ARBRES DE DÉCISION

Jérémie Cabessa

Laboratoire DAVID, UVSQ

INTRODUCTION

- ▶ Les arbres de décision sont des algorithmes d'apprentissage supervisé.
- ▶ Ils peuvent être utilisés dans des contextes de *régression* et de *classification*.
- ▶ Les arbres de décisions sont en général moins performants que d'autres méthodes d'apprentissage classiques.
- ▶ Mais ils peuvent être généralisés de manières très performantes: bagging, random forests (forêts aléatoires), boosting.

INTRODUCTION

- ▶ Les arbres de décision sont des algorithmes d'*apprentissage supervisé*.
- ▶ Ils peuvent être utilisés dans des contextes de *régression* et de *classification*.
- ▶ Les arbres de décisions sont en général moins performants que d'autres méthodes d'apprentissage classiques.
- ▶ Mais ils peuvent être généralisés de manières très performantes: *bagging*, *random forests* (forêts aléatoires), *boosting*.

INTRODUCTION

- ▶ Les arbres de décision sont des algorithmes d'*apprentissage supervisé*.
- ▶ Ils peuvent être utilisés dans des contextes de *régression* et de *classification*.
- ▶ Les arbres de décisions sont en général moins performants que d'autres méthodes d'apprentissage classiques.
- ▶ Mais ils peuvent être généralisés de manières très performantes: bagging, random forests (forêts aléatoires), boosting.

INTRODUCTION

- ▶ Les **arbres de décision** sont des algorithmes d'*apprentissage supervisé*.
- ▶ Ils peuvent être utilisés dans des contextes de *régression* et de *classification*.
- ▶ Les arbres de décisions sont en général moins performants que d'autres méthodes d'apprentissage classiques.
- ▶ Mais ils peuvent être généralisés de manières très performantes: **bagging, random forests (forêts aléatoires), boosting**.

INTRODUCTION

- ▶ Les arbres de régression peuvent modéliser une relation *non-linéaire* entre les prédicteurs X_1, \dots, X_p et la réponse Y .
- ▶ Idée générale: Un arbre de décision *partitionne* l'espace des prédicteurs $X_1 \times \dots \times X_p$ en K régions distinctes R_1, \dots, R_K (*cellules, boxes*) au sein desquelles les data se ressemblent.
- ▶ Les exemples x_i qui appartiennent à une même région R_k ont des réponses y_i qui sont proches les unes des autres.

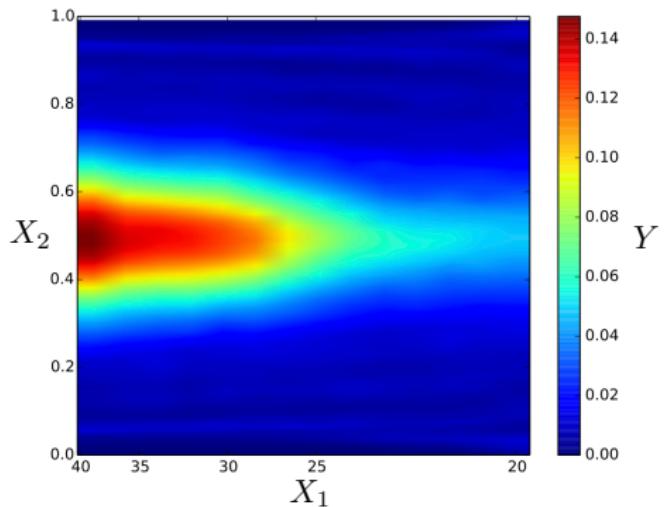
INTRODUCTION

- ▶ Les arbres de régression peuvent modéliser une relation *non-linéaire* entre les prédicteurs X_1, \dots, X_p et la réponse Y .
- ▶ **Idée générale:** Un arbre de décision *partitionne* l'espace des prédicteurs $X_1 \times \dots \times X_p$ en K régions distinctes R_1, \dots, R_K (*cellules, boxes*) au sein desquelles les data se ressemblent.
- ▶ Les exemples x_i qui appartiennent à une même région R_k ont des réponses y_i qui sont proches les unes des autres.

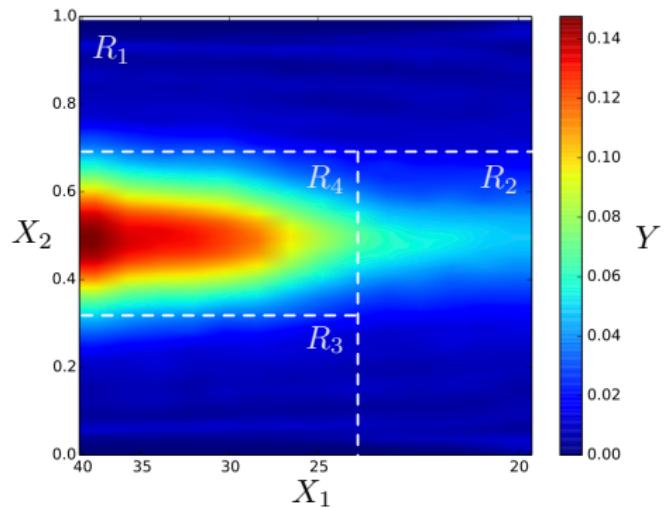
INTRODUCTION

- ▶ Les arbres de régression peuvent modéliser une relation *non-linéaire* entre les prédicteurs X_1, \dots, X_p et la réponse Y .
- ▶ **Idée générale:** Un arbre de décision *partitionne* l'espace des prédicteurs $X_1 \times \dots \times X_p$ en K régions distinctes R_1, \dots, R_K (*cellules, boxes*) au sein desquelles les data se ressemblent.
- ▶ Les exemples x_i qui appartiennent à une même région R_k ont des réponses y_i qui sont proches les unes des autres.

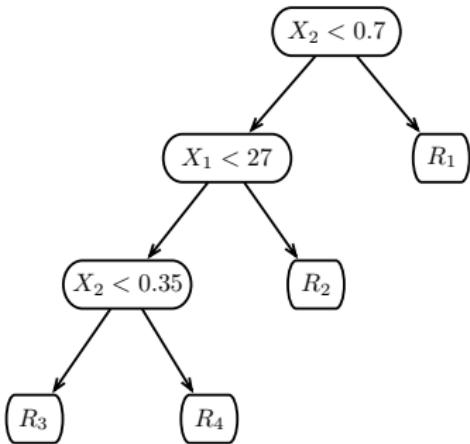
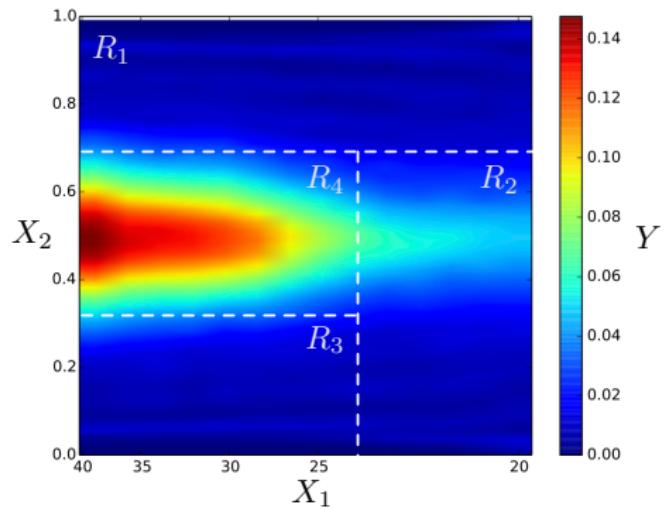
INTRODUCTION: ARBRE DE RÉGRESSION



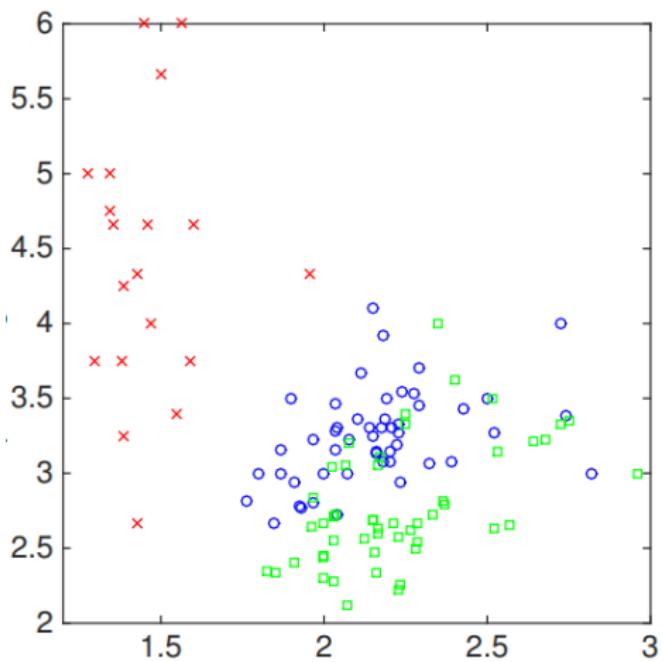
INTRODUCTION: ARBRE DE RÉGRESSION



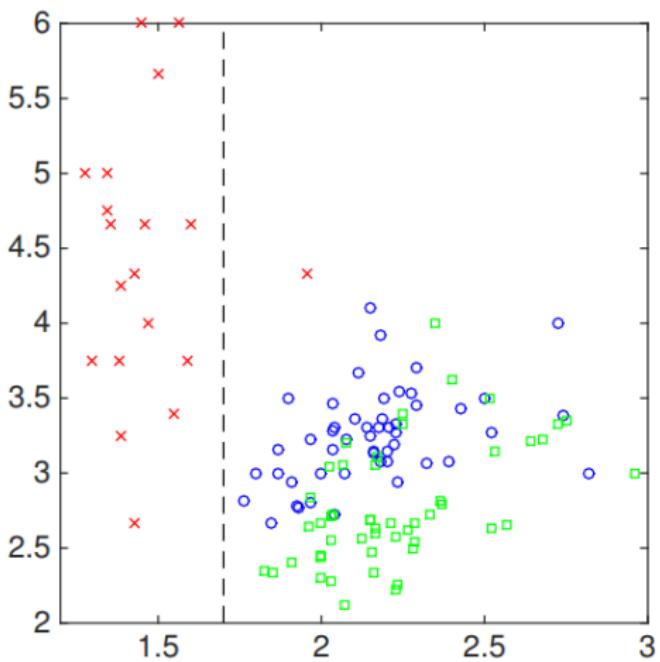
INTRODUCTION: ARBRE DE RÉGRESSION



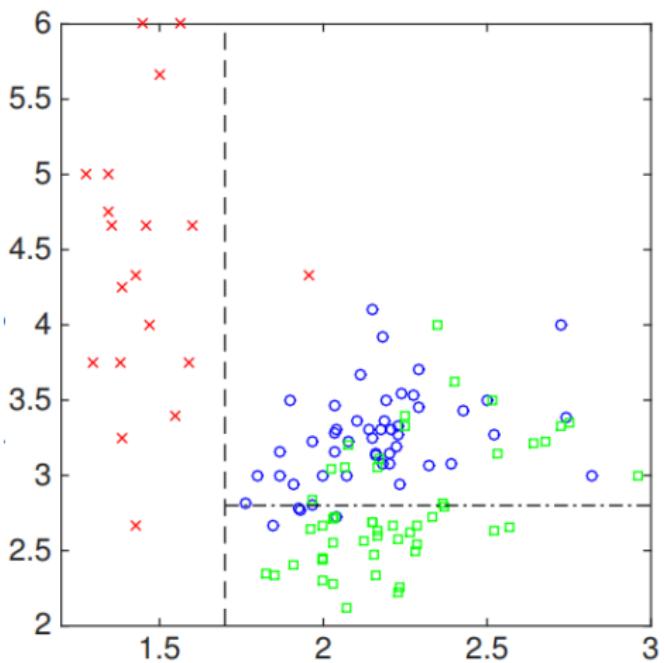
INTRODUCTION: ARBRE DE CLASSIFICATION



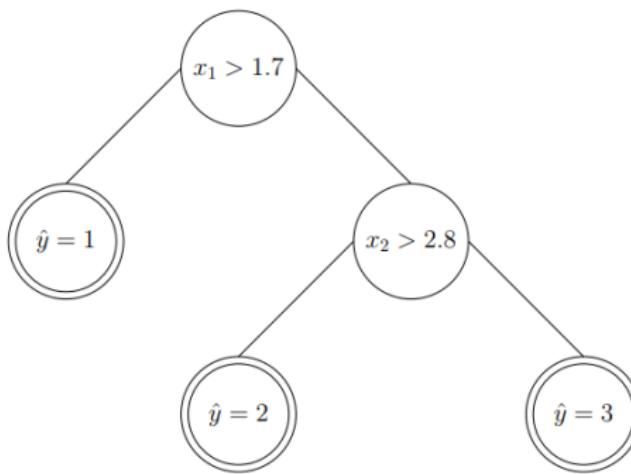
INTRODUCTION: ARBRE DE CLASSIFICATION



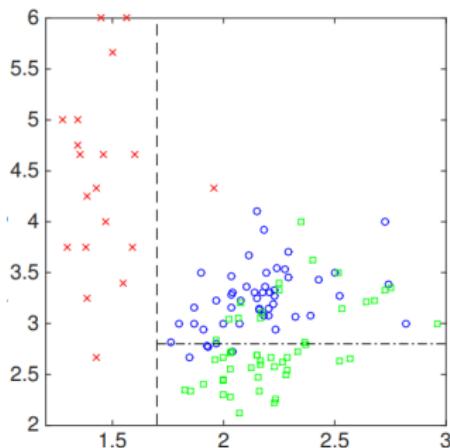
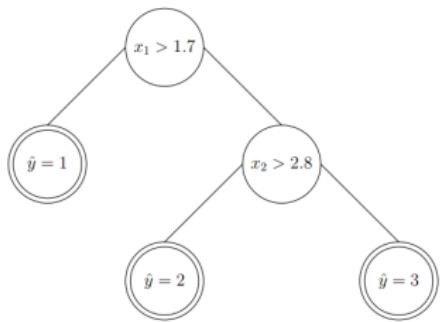
INTRODUCTION: ARBRE DE CLASSIFICATION



INTRODUCTION: ARBRE DE CLASSIFICATION



INTRODUCTION: ARBRE DE CLASSIFICATION



ARBRES DE RÉGRESSION (REGRESSION TREES)

- ▶ Les arbres de régression (**regression trees**) sont utilisés lorsque la variable réponse est *quantitative* (continue).
- ▶ Un *arbre de décision* T correspond à une *partition* R_1, \dots, R_K de l'espace des prédicteurs $X_1 \times \dots \times X_p$.
- ▶ Comment construire cette partition?

ARBRES DE RÉGRESSION (REGRESSION TREES)

- ▶ Les arbres de régression (**regression trees**) sont utilisés lorsque la variable réponse est *quantitative* (continue).
- ▶ Un *arbre de décision* T correspond à une *partition* R_1, \dots, R_K de l'espace des prédicteurs $X_1 \times \dots \times X_p$.
- ▶ Comment construire cette partition?

ARBRES DE RÉGRESSION (REGRESSION TREES)

- ▶ Les arbres de régression (**regression trees**) sont utilisés lorsque la variable réponse est *quantitative* (continue).
- ▶ Un *arbre de décision* T correspond à une *partition* R_1, \dots, R_K de l'espace des prédicteurs $X_1 \times \dots \times X_p$.
- ▶ Comment construire cette partition?

ARBRES DE RÉGRESSION

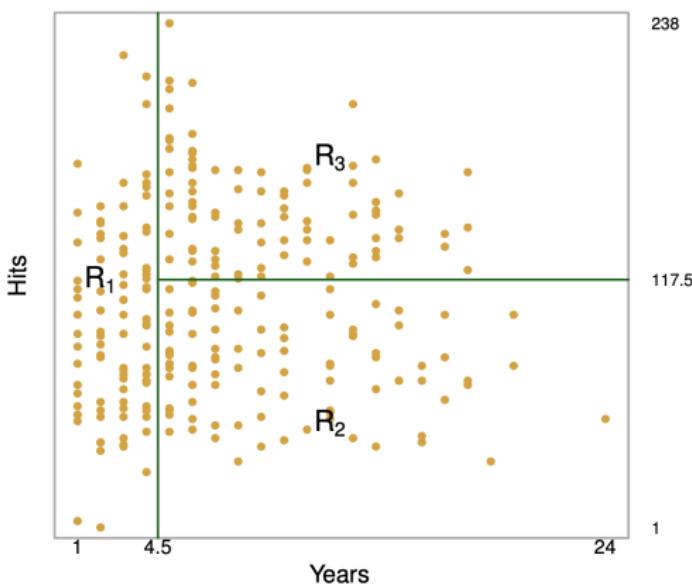
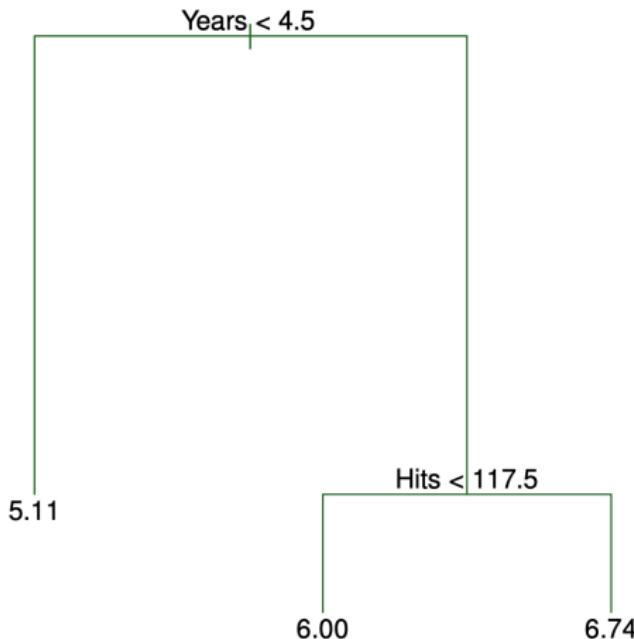
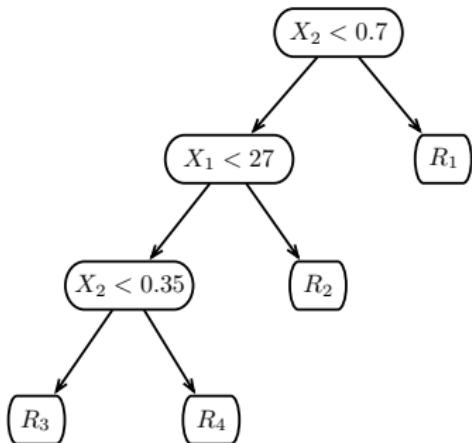
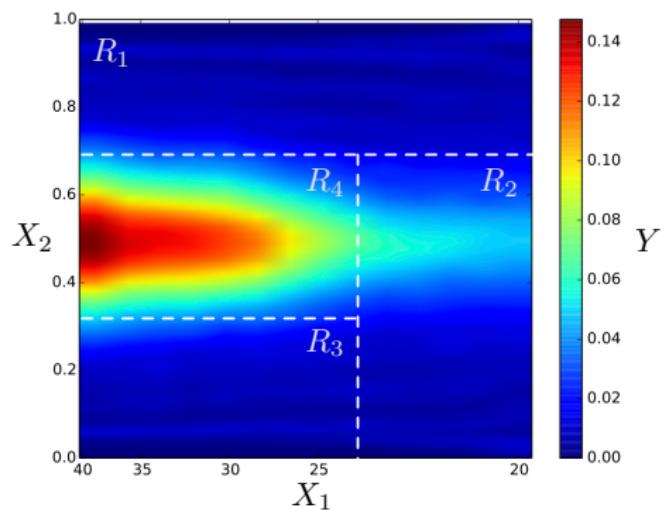


FIGURE 8.2. The three-region partition for the **Hitters** data set from the regression tree illustrated in Figure 8.1.

ARBRES DE RÉGRESSION



ARBRES DE RÉGRESSION



ARBRES DE RÉGRESSION

- ▶ Soit le train set $S_{\text{train}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R} : i = 1, \dots, N\}$.
- ▶ On cherche la partition $R_1, \dots, R_K \subseteq X_1 \times \dots \times X_p$ qui minimise la somme des carrés des résidus (residual sum of squares)

$$RSS(R_1, \dots, R_K) = \sum_{k=1}^K \sum_{\{i : \mathbf{x}_i \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

On cherche à minimiser la somme des carrés des résidus (RSS) pour trouver la partition optimale.

La partition optimale est obtenue par un processus d'itération et de division récursive.

Le processus se termine lorsque les nœuds sont suffisamment petits ou lorsque la RSS n'est plus réduite.

Le résultat final est un arbre de régression qui peut être utilisé pour faire des prédictions.

ARBRES DE RÉGRESSION

- ▶ Soit le train set $S_{\text{train}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R} : i = 1, \dots, N\}$.
- ▶ On cherche la partition $R_1, \dots, R_K \subseteq X_1 \times \dots \times X_p$ qui minimise la somme des carrés des résidus (residual sum of squares)

$$RSS(R_1, \dots, R_K) = \sum_{k=1}^K \sum_{\{i: \mathbf{x}_i \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

où \bar{y}_{R_k} est la moyenne des réponses associées aux x_i de la région R_k :

$$\bar{y}_{R_k} = \frac{1}{|R_k|} \sum_{\{i: \mathbf{x}_i \in R_k\}} y_i$$

ARBRES DE RÉGRESSION

- ▶ Soit le train set $S_{\text{train}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R} : i = 1, \dots, N\}$.
- ▶ On cherche la partition $R_1, \dots, R_K \subseteq X_1 \times \dots \times X_p$ qui minimise la somme des carrés des résidus (residual sum of squares)

$$RSS(R_1, \dots, R_K) = \sum_{k=1}^K \sum_{\{i: \mathbf{x}_i \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

où \bar{y}_{R_k} est la moyenne des réponses associées aux x_i de la région R_k :

$$\bar{y}_{R_k} = \frac{1}{|R_k|} \sum_{\{i: \mathbf{x}_i \in R_k\}} y_i$$

ARBRES DE RÉGRESSION

- ▶ Soit le train set $S_{\text{train}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R} : i = 1, \dots, N\}$.
- ▶ On cherche la partition $R_1, \dots, R_K \subseteq X_1 \times \dots \times X_p$ qui minimise la somme des carrés des résidus (residual sum of squares)

$$RSS(R_1, \dots, R_K) = \sum_{k=1}^K \sum_{\{i: \mathbf{x}_i \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

où \bar{y}_{R_k} est la moyenne des réponses associées aux \mathbf{x}_i de la région R_k :

$$\bar{y}_{R_k} = \frac{1}{|R_k|} \sum_{\{i: \mathbf{x}_i \in R_k\}} y_i$$

ARBRES DE RÉGRESSION

- ▶ Computationnellement, on ne peut pas tester toutes les partitions R_1, \dots, R_K possibles (le problème de minimisation de la RSS est NP-hard).
- ▶ Pour $N = 51$ points et $K = 6$ cellules, on a $\binom{51-1}{6-1} > 2M$ partitions possibles (résultat combinatoire).
- ▶ Pour construire la partition R_1, \dots, R_K , on utilise un algorithme top-down, récursif et gourmand (greedy): **recursive binary splitting**.

ARBRES DE RÉGRESSION

- ▶ Computationnellement, on ne peut pas tester toutes les partitions R_1, \dots, R_K possibles (le problème de minimisation de la RSS est NP-hard).
- ▶ Pour $N = 51$ points et $K = 6$ cellules, on a $\binom{51-1}{6-1} > 2M$ partitions possibles (résultat combinatoire).
- ▶ Pour construire la partition R_1, \dots, R_K , on utilise un algorithme top-down, récursif et gourmand (greedy): recursive binary splitting.

ARBRES DE RÉGRESSION

- ▶ Computationnellement, on ne peut pas tester toutes les partitions R_1, \dots, R_K possibles (le problème de minimisation de la RSS est NP-hard).
- ▶ Pour $N = 51$ points et $K = 6$ cellules, on a $\binom{51-1}{6-1} > 2M$ partitions possibles (résultat combinatoire).
- ▶ Pour construire la partition R_1, \dots, R_K , on utilise un algorithme top-down, récursif et gourmand (greedy): **recursive binary splitting**.

ARBRES DE RÉGRESSION

- ▶ **Idée générale (rappel):** Un arbre de décision *partitionne* l'espace des prédicteurs $X_1 \times \cdots \times X_p$ en K régions distinctes R_1, \dots, R_K (*cellules, boxes*) au sein desquelles les data se ressemblent.
- ▶ Les exemples x_i qui appartiennent à une même région R_k ont des réponses y_i qui sont proches les unes des autres.
- ▶ On veut minimiser la *variance* des réponses des données au sein des régions R_1, \dots, R_K .

ARBRES DE RÉGRESSION

- ▶ **Idée générale (rappel):** Un arbre de décision *partitionne* l'espace des prédicteurs $X_1 \times \cdots \times X_p$ en K régions distinctes R_1, \dots, R_K (*cellules, boxes*) au sein desquelles les data se ressemblent.
- ▶ Les exemples x_i qui appartiennent à une même région R_k ont des réponses y_i qui sont proches les unes des autres.
- ▶ On veut minimiser la *variance* des réponses des données au sein des régions R_1, \dots, R_K .

ARBRES DE RÉGRESSION

- ▶ **Idée générale (rappel):** Un arbre de décision *partitionne* l'espace des prédicteurs $X_1 \times \cdots \times X_p$ en K régions distinctes R_1, \dots, R_K (*cellules, boxes*) au sein desquelles les data se ressemblent.
- ▶ Les exemples x_i qui appartiennent à une même région R_k ont des réponses y_i qui sont proches les unes des autres.
- ▶ On veut minimiser la *variance* des réponses des données au sein des régions R_1, \dots, R_K .

ARBRES DE RÉGRESSION

- ▶ Pour une région R_k , la variance des réponses est donnée par

$$\text{Var}(R_k) = \frac{1}{|R_k| - 1} \sum_{\{i: x_i \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

où \bar{y}_{R_k} est la moyenne des réponses associées aux x_i de la région R_k .

- ▶ Ainsi, la minimisation des variances individuelles des régions R_1, \dots, R_K correspond bien à la minimisation de la RSS

$$\begin{aligned} \text{RSS}(R_1, \dots, R_K) &= \sum_{k=1}^K \sum_{\{i: x_i \in R_k\}} (y_i - \bar{y}_{R_k})^2 \\ &= \sum_{k=1}^K (|R_k| - 1) \text{Var}(R_k) \end{aligned}$$

ARBRES DE RÉGRESSION

- ▶ Pour une région R_k , la variance des réponses est donnée par

$$\text{Var}(R_k) = \frac{1}{|R_k| - 1} \sum_{\{i: x_i \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

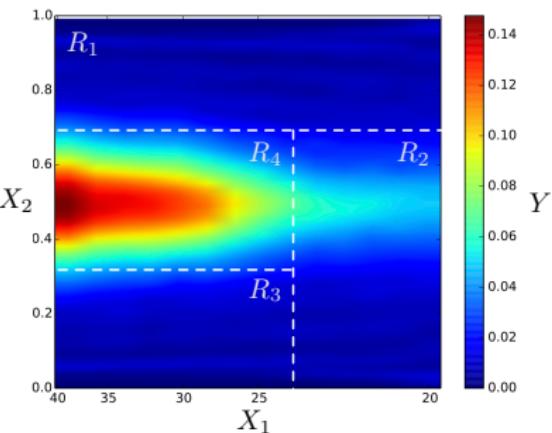
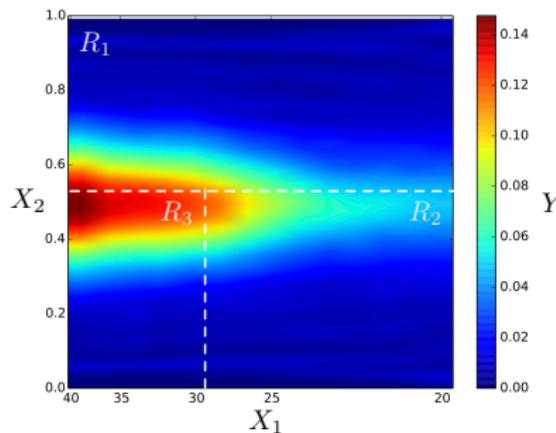
où \bar{y}_{R_k} est la moyenne des réponses associées aux x_i de la région R_k .

- ▶ Ainsi, la minimisation des variances individuelles des régions R_1, \dots, R_K correspond bien à la minimisation de la RSS

$$\begin{aligned} \text{RSS}(R_1, \dots, R_K) &= \sum_{k=1}^K \sum_{\{i: x_i \in R_k\}} (y_i - \bar{y}_{R_k})^2 \\ &= \sum_{k=1}^K (|R_k| - 1) \text{Var}(R_k) \end{aligned}$$

INTRODUCTION: ARBRE DE RÉGRESSION

- ▶ La partition de droite est plus *homogène* (variances réduites) que celle de gauche.



ARBRES DE RÉGRESSION

- ▶ Un *split* de la région R est un tuple (m, s) qui partitionne R en deux sous-régions

$$R_1 = \{\mathbf{x} \in R : x_m \leq s\} \text{ et } R_2 = \{\mathbf{x} \in R : x_m > s\}$$

où m est le *feature index* et s le *threshold*.

- ▶ Un split (m, s) de R en R_1 et R_2 est dit *bon* s'il engendre une réduction de variance, i.e.,

$$\text{Var}(R_1) + \text{Var}(R_2) < \text{Var}(R) \text{ i.e.}$$

$$\text{VarRed}(R, R_1, R_2) = \text{Var}(R) - (\text{Var}(R_1) + \text{Var}(R_2)) > 0$$

- ▶ Un split (m, s) de R est dit *optimal* s'il engendre une réduction de variance plus grande que celle associée à tout autre split.

ARBRES DE RÉGRESSION

- ▶ Un *split* de la région R est un tuple (m, s) qui partitionne R en deux sous-régions

$$R_1 = \{\mathbf{x} \in R : x_m \leq s\} \text{ et } R_2 = \{\mathbf{x} \in R : x_m > s\}$$

où m est le *feature index* et s le *threshold*.

- ▶ Un split (m, s) de R en R_1 et R_2 est dit *bon* s'il engendre une réduction de variance, i.e.,

$$\text{Var}(R_1) + \text{Var}(R_2) < \text{Var}(R) \text{ i.e.}$$

$$\text{VarRed}(R, R_1, R_2) = \text{Var}(R) - (\text{Var}(R_1) + \text{Var}(R_2)) > 0$$

- ▶ Un split (m, s) de R est dit *optimal* s'il engendre une réduction de variance plus grande que celle associée à tout autre split.

ARBRES DE RÉGRESSION

- ▶ Un *split* de la région R est un tuple (m, s) qui partitionne R en deux sous-régions

$$R_1 = \{\mathbf{x} \in R : x_m \leq s\} \text{ et } R_2 = \{\mathbf{x} \in R : x_m > s\}$$

où m est le *feature index* et s le *threshold*.

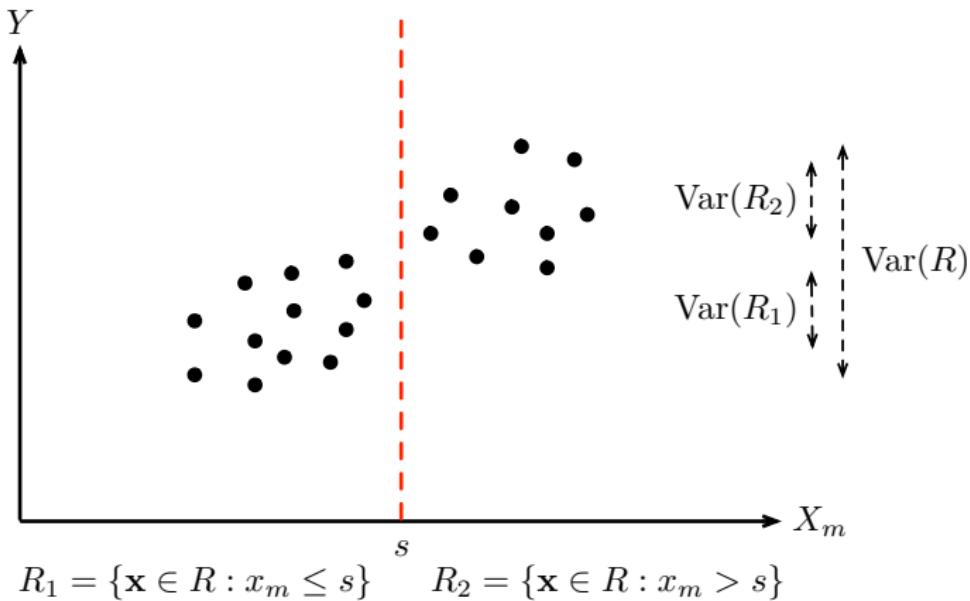
- ▶ Un split (m, s) de R en R_1 et R_2 est dit *bon* s'il engendre une réduction de variance, i.e.,

$$\text{Var}(R_1) + \text{Var}(R_2) < \text{Var}(R) \text{ i.e.}$$

$$\text{VarRed}(R, R_1, R_2) = \text{Var}(R) - (\text{Var}(R_1) + \text{Var}(R_2)) > 0$$

- ▶ Un split (m, s) de R est dit *optimal* s'il engendre une réduction de variance plus grande que celle associée à tout autre split.

ARBRES DE RÉGRESSION



ARBRES DE RÉGRESSION

- ▶ **Best split:** algorithme de recherche d'un best split d'une region R d'un dataset.
 - 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 de variance $\text{VarRed}(R, R_1, R_2)$.

ARBRES DE RÉGRESSION

- ▶ **Best split:** algorithme de recherche d'un best split d'une region R d'un dataset.
 - 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 de variance $\text{VarRed}(R, R_1, R_2)$.

ARBRES DE RÉGRESSION

- ▶ **Best split:** algorithme de recherche d'un best split d'une region R d'un dataset.
 - 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 de variance $\text{VarRed}(R, R_1, R_2)$.

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

Algorithm 1: best_split(dataset)

```
best_var_red = -∞
for m = 1 to nb_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
            var_red = variance_reduction(dataset, data_l, data_r)
            if var_red > best_var_red then
                best_split = split
                best_var_red = var_red
            end
        end
    end
end
return {"split": best_split, "var_reduction": best_var_red,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE RÉGRESSION

► **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.

- On commence avec le dataset complet R .
- On cherche son best split (m, s) .
- Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
- Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
- On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE RÉGRESSION

- ▶ **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.
- On commence avec le dataset complet R .
- On cherche son best split (m, s) .
- Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
- Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
- On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE RÉGRESSION

- ▶ **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.
- On commence avec le dataset complet R .
- On cherche son best split (m, s) .
- Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
- Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
- On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE RÉGRESSION

- ▶ **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.
- On commence avec le dataset complet R .
- On cherche son best split (m, s) .
- Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
- Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
- On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE RÉGRESSION

- ▶ **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.
 - On commence avec le dataset complet R .
 - On cherche son best split (m, s) .
 - Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
 - Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
 - On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE RÉGRESSION

- ▶ **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.
 - On commence avec le dataset complet R .
 - On cherche son best split (m, s) .
 - Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
 - Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
 - On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE RÉGRESSION

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[var_reduction] ≥ 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)
```

ARBRES DE RÉGRESSION

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[var_reduction] ≥ 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)
```

ARBRES DE RÉGRESSION

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[var_reduction] ≥ 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)
```

ARBRES DE RÉGRESSION

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[var_reduction] ≥ 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)
```

ARBRES DE RÉGRESSION

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[var_reduction] ≥ 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)
```

ARBRES DE RÉGRESSION

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[var_reduction] ≥ 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)
```

ARBRES DE RÉGRESSION

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[var_reduction] ≥ 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)
```

ARBRES DE RÉGRESSION

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[var_reduction] ≥ 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)
```

ARBRES DE RÉGRESSION

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[var_reduction] ≥ 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)
```

ARBRES DE RÉGRESSION

- ▶ Une fois l'arbre de décision construit (par Recursive Binary Splitting), la *prédiction* \hat{y} associée à la data x est obtenue de la manière suivante:
- ▶ On cherche la région R_k de l'espace des features $X_1 \times \dots \times X_p$ à laquelle appartient x ;
- ▶ On prend comme prédiction \hat{y} la moyenne des targets de cette région R_k :

$$\hat{y} = \frac{1}{|R_k|} \sum_{\{i: x_i \in R_k\}} y_i$$

ARBRES DE RÉGRESSION

- ▶ Une fois l'arbre de décision construit (par Recursive Binary Splitting), la *prédiction* \hat{y} associée à la data x est obtenue de la manière suivante:
- ▶ On cherche la région R_k de l'espace des features $X_1 \times \cdots \times X_p$ à laquelle appartient x ;
- ▶ On prend comme prédiction \hat{y} la moyenne des targets de cette région R_k :

$$\hat{y} = \frac{1}{|R_k|} \sum_{\{i: x_i \in R_k\}} y_i$$

ARBRES DE RÉGRESSION

- ▶ Une fois l'arbre de décision construit (par Recursive Binary Splitting), la *prédiction* \hat{y} associée à la data x est obtenue de la manière suivante:
- ▶ On cherche la région R_k de l'espace des features $X_1 \times \cdots \times X_p$ à laquelle appartient x ;
- ▶ On prend comme prédiction \hat{y} la moyenne des targets de cette région R_k :

$$\hat{y} = \frac{1}{|R_k|} \sum_{\{i: x_i \in R_k\}} y_i$$

ARBRES DE CLASSIFICATION (CLASSIFICATION TREES)

- ▶ Les **arbres de classification** (**classification trees**) sont utilisés lorsque la variable réponse est *qualitative* (discrète).
- ▶ Un *arbre de classification* T correspond aussi à une *partition* R_1, \dots, R_K de l'espace des prédicteurs $X_1 \times \dots \times X_p$.
- ▶ Le principe est très similaire à celui d'un arbre de régression.

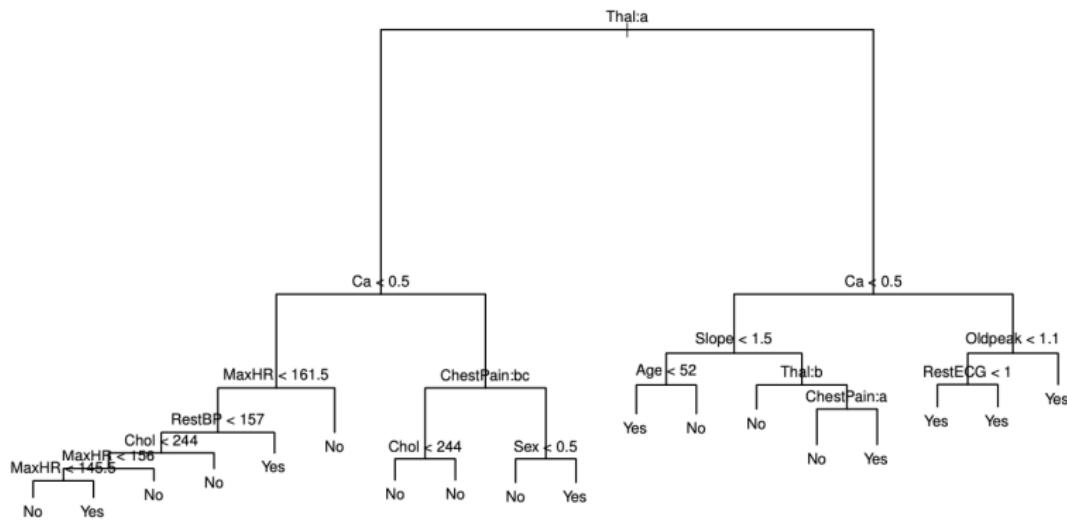
ARBRES DE CLASSIFICATION (CLASSIFICATION TREES)

- ▶ Les **arbres de classification** (**classification trees**) sont utilisés lorsque la variable réponse est *qualitative* (discrète).
- ▶ Un *arbre de classification* T correspond aussi à une *partition* R_1, \dots, R_K de l'espace des prédicteurs $X_1 \times \dots \times X_p$.
- ▶ Le principe est très similaire à celui d'un arbre de régression.

ARBRES DE CLASSIFICATION (CLASSIFICATION TREES)

- ▶ Les **arbres de classification** (**classification trees**) sont utilisés lorsque la variable réponse est *qualitative* (discrète).
- ▶ Un *arbre de classification* T correspond aussi à une *partition* R_1, \dots, R_K de l'espace des prédicteurs $X_1 \times \dots \times X_p$.
- ▶ Le principe est très similaire à celui d'un arbre de régression.

ARBRES DE CLASSIFICATION



ARBRES DE CLASSIFICATION

- ▶ Soit le train set $S_{\text{train}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathcal{C} : i = 1, \dots, N\}$.
- ▶ On cherche la partition $R_1, \dots, R_K \subseteq X_1 \times \dots \times X_p$ qui minimise une certaine fonction de loss.
- ▶ Cette fois, on ne considère pas la somme des carrés des résidus (residual sum of squares) $RSS(R_1, \dots, R_K)$.
- ▶ On aimerait que les régions R_1, \dots, R_K soient les plus *pures* (homogènes) possible en termes de targets y_i .

ARBRES DE CLASSIFICATION

- ▶ Soit le train set $S_{\text{train}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathcal{C} : i = 1, \dots, N\}$.
- ▶ On cherche la partition $R_1, \dots, R_K \subseteq X_1 \times \dots \times X_p$ qui minimise une certaine fonction de loss.
- ▶ Cette fois, on ne considère pas la somme des carrés des résidus (residual sum of squares) $RSS(R_1, \dots, R_K)$.
- ▶ On aimerait que les régions R_1, \dots, R_K soient les plus *pures* (homogènes) possible en termes de targets y_i .

ARBRES DE CLASSIFICATION

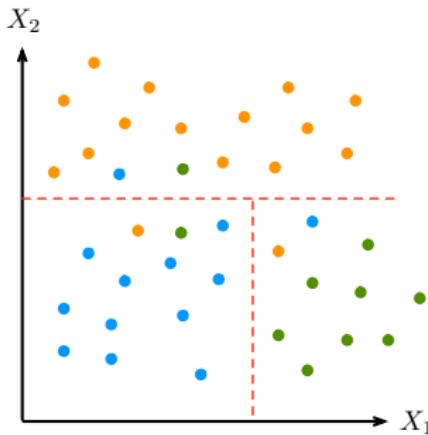
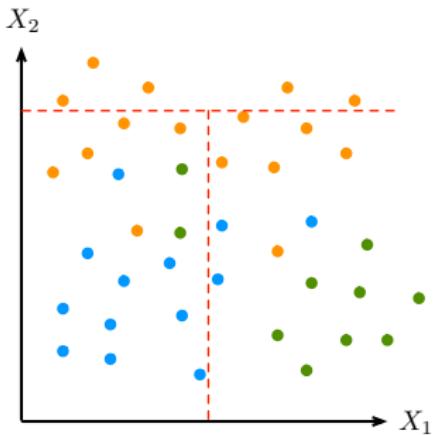
- ▶ Soit le train set $S_{\text{train}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathcal{C} : i = 1, \dots, N\}$.
- ▶ On cherche la partition $R_1, \dots, R_K \subseteq X_1 \times \dots \times X_p$ qui minimise une certaine fonction de loss.
- ▶ Cette fois, on ne considère pas la somme des carrés des résidus (residual sum of squares) $RSS(R_1, \dots, R_K)$.
- ▶ On aimerait que les régions R_1, \dots, R_K soient les plus *pures* (homogènes) possible en termes de targets y_i .

ARBRES DE CLASSIFICATION

- ▶ Soit le train set $S_{\text{train}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathcal{C} : i = 1, \dots, N\}$.
- ▶ On cherche la partition $R_1, \dots, R_K \subseteq X_1 \times \dots \times X_p$ qui minimise une certaine fonction de loss.
- ▶ Cette fois, on ne considère pas la somme des carrés des résidus (residual sum of squares) $RSS(R_1, \dots, R_K)$.
- ▶ On aimerait que les régions R_1, \dots, R_K soient les plus *pures* (homogènes) possible en termes de targets y_i .

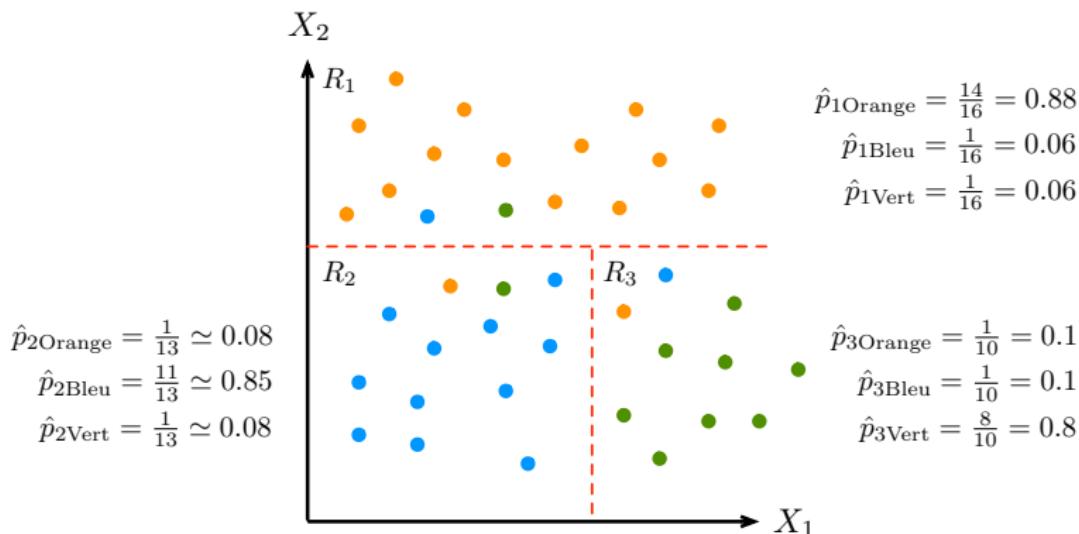
ARBRES DE CLASSIFICATION

- ▶ La partition de droite est plus *pure* (homogène) que celle de gauche.



ARBRES DE CLASSIFICATION

- Soit \hat{p}_{kc} la proportion de y_i égaux à c dans la région R_k .



ARBRES DE CLASSIFICATION

- ▶ La pureté d'une région R_k est donnée par son *indice de Gini* ou son *entropie*.

$$\text{Gini}(R_k) = \sum_{c=1}^C \hat{p}_{kc} (1 - \hat{p}_{kc})$$

$$\text{Ent}(R_k) = - \sum_{c=1}^C \hat{p}_{kc} \log(\hat{p}_{kc})$$

- ▶ Remarque: plus la région R_k est *pure*, plus les \hat{p}_{kc} successifs sont proches de 0 ou de 1, et donc plus $\text{Gini}(R_k)$ et $\text{Ent}(R_k)$ sont proches de 0.

ARBRES DE CLASSIFICATION

- ▶ La pureté d'une région R_k est donnée par son *indice de Gini* ou son *entropie*.

$$\text{Gini}(R_k) = \sum_{c=1}^C \hat{p}_{kc} (1 - \hat{p}_{kc})$$

$$\text{Ent}(R_k) = - \sum_{c=1}^C \hat{p}_{kc} \log(\hat{p}_{kc})$$

- ▶ **Remarque:** plus la région R_k est *pure*, plus les \hat{p}_{kc} successifs sont proches de 0 ou de 1, et donc plus $\text{Gini}(R_k)$ et $\text{Ent}(R_k)$ sont proches de 0.

ARBRES DE CLASSIFICATION

- ▶ Ainsi, la minimisation des indices de Gini ou des entropies des régions R_1, \dots, R_K correspond à la minimisation des fonctions de coût suivantes:

$$\text{Gini}(R_1, \dots, R_K) = \sum_{k=1}^K \sum_{c=1}^C \hat{p}_{kc} (1 - \hat{p}_{kc})$$

$$\text{Entropy}(R_1, \dots, R_K) = - \sum_{k=1}^K \sum_{c=1}^C \hat{p}_{kc} \log(\hat{p}_{kc})$$

ARBRES DE CLASSIFICATION

- ▶ Un *split* de la région R est un tuple (m, s) qui partitionne R en deux sous-régions

$$R_1 = \{\mathbf{x} \in R : x_m \leq s\} \text{ et } R_2 = \{\mathbf{x} \in R : x_m > s\}$$

où m est le *feature index* et s le *threshold*.

- ▶ Un split (m, s) de R en R_1 et R_2 est *bon* s'il engendre un *gain informationnel*, c'est-à-dire une réduction de l'entropie (ou de l'indice de Gini):

$$\text{Ent}(R_1) + \text{Ent}(R_2) < \text{Ent}(R) \text{ i.e.}$$

$$\text{EntRed}(R, R_1, R_2) = \text{Ent}(R) - (\text{Ent}(R_1) + \text{Ent}(R_2)) > 0$$

- ▶ Un split (m, s) de R est dit *optimal* s'il engendre un gain informationnel plus grand que celui associé à tout autre split.

ARBRES DE CLASSIFICATION

- ▶ Un *split* de la région R est un tuple (m, s) qui partitionne R en deux sous-régions

$$R_1 = \{\boldsymbol{x} \in R : x_m \leq s\} \text{ et } R_2 = \{\boldsymbol{x} \in R : x_m > s\}$$

où m est le *feature index* et s le *threshold*.

- ▶ Un split (m, s) de R en R_1 et R_2 est *bon* s'il engendre un *gain informationnel*, c'est-à-dire une réduction de l'entropie (ou de l'indice de Gini):

$$\text{Ent}(R_1) + \text{Ent}(R_2) < \text{Ent}(R) \text{ i.e.}$$

$$\text{EntRed}(R, R_1, R_2) = \text{Ent}(R) - (\text{Ent}(R_1) + \text{Ent}(R_2)) > 0$$

- ▶ Un split (m, s) de R est dit *optimal* s'il engendre un gain informationnel plus grand que celui associé à tout autre split.

ARBRES DE CLASSIFICATION

- ▶ Un *split* de la région R est un tuple (m, s) qui partitionne R en deux sous-régions

$$R_1 = \{\boldsymbol{x} \in R : x_m \leq s\} \text{ et } R_2 = \{\boldsymbol{x} \in R : x_m > s\}$$

où m est le *feature index* et s le *threshold*.

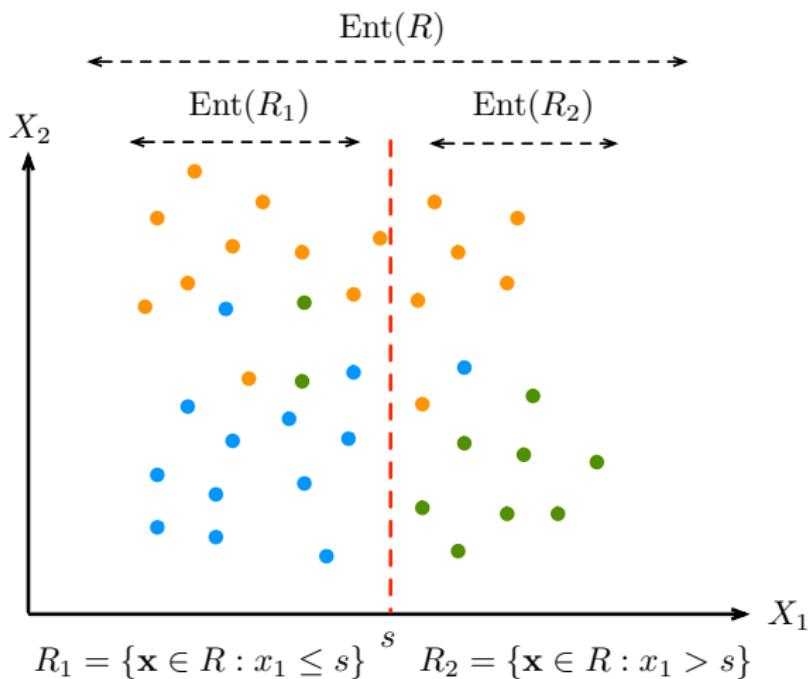
- ▶ Un split (m, s) de R en R_1 et R_2 est *bon* s'il engendre un *gain informationnel*, c'est-à-dire une réduction de l'entropie (ou de l'indice de Gini):

$$\text{Ent}(R_1) + \text{Ent}(R_2) < \text{Ent}(R) \text{ i.e.}$$

$$\text{EntRed}(R, R_1, R_2) = \text{Ent}(R) - (\text{Ent}(R_1) + \text{Ent}(R_2)) > 0$$

- ▶ Un split (m, s) de R est dit *optimal* s'il engendre un gain informationnel plus grand que celui associé à tout autre split.

ARBRES DE CLASSIFICATION



ARBRES DE CLASSIFICATION

- ▶ **Best split:** algorithme de recherche d'un best split d'une region R d'un dataset.
 - 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 $\text{EntRed}(R, R_1, R_2)$.

ARBRES DE CLASSIFICATION

- ▶ **Best split:** algorithme de recherche d'un best split d'une region R d'un dataset.
- 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 $\text{EntRed}(R, R_1, R_2)$.

ARBRES DE CLASSIFICATION

- ▶ **Best split:** algorithme de recherche d'un best split d'une region R d'un dataset.
 - 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 $\text{EntRed}(R, R_1, R_2)$.

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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
return {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

Algorithm 3: best_split(dataset)

```
best_info_gain = -∞
for m = 1 to nb_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 {"split": best_split, "info_gain": best_info_gain,
         "dataset_left": data_l, "dataset_right": data_r}
```

ARBRES DE CLASSIFICATION

► **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.

- On commence avec le dataset complet R .
- On cherche son best split (m, s) .
- Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
- Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
- On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE CLASSIFICATION

- ▶ **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.
- On commence avec le dataset complet R .
- On cherche son best split (m, s) .
- Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
- Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
- On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE CLASSIFICATION

- ▶ **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.
- On commence avec le dataset complet R .
- On cherche son best split (m, s) .
- Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
- Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
- On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE CLASSIFICATION

- ▶ **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.
 - On commence avec le dataset complet R .
 - On cherche son best split (m, s) .
 - Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
 - Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
 - On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE CLASSIFICATION

- ▶ **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.
- On commence avec le dataset complet R .
- On cherche son best split (m, s) .
- Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
- Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
- On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE CLASSIFICATION

- ▶ **Recursive Binary Splitting:** algorithme récursif qui construit l'arbre de décision de haut en bas.
 - On commence avec le dataset complet R .
 - On cherche son best split (m, s) .
 - Ce split partitionne R en R_1 (fils gauche) et R_2 (fils droit).
 - Récursivement, on cherche les best splits et les partitions associées pour les fils gauche et droit R_1 et R_2 de R .
 - On s'arrête lorsque la région R_1 ou R_2 est trop petite, ou lorsqu'une profondeur maximale est atteinte (condition d'arrêt).

ARBRES DE CLASSIFICATION

Recursive Binary Splitting

Algorithm 4: 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)
```

ARBRES DE CLASSIFICATION

Recursive Binary Splitting

Algorithm 4: 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)
```

ARBRES DE CLASSIFICATION

Recursive Binary Splitting

Algorithm 4: 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)
```

ARBRES DE CLASSIFICATION

Recursive Binary Splitting

Algorithm 4: 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)
```

ARBRES DE CLASSIFICATION

Recursive Binary Splitting

Algorithm 4: 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)
```

ARBRES DE CLASSIFICATION

Recursive Binary Splitting

Algorithm 4: 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)
```

ARBRES DE CLASSIFICATION

Recursive Binary Splitting

Algorithm 4: 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)
```

ARBRES DE CLASSIFICATION

Recursive Binary Splitting

Algorithm 4: 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)
```

ARBRES DE CLASSIFICATION

Recursive Binary Splitting

Algorithm 4: 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)
```

ARBRES DE CLASSIFICATION

- ▶ Une fois l'arbre de décision construit (par Recursive Binary Splitting), la *prédiction* \hat{y} associée à la data x est obtenue de la manière suivante:
- ▶ On cherche la région R_k de l'espace des features $X_1 \times \dots \times X_p$ à laquelle appartient x ;
- ▶ On prend comme prédiction \hat{y} la target qui apparaît le plus souvent dans cette région R_k :

$$\hat{y} = \arg \max_{c=1,\dots,C} \hat{p}_{kc}$$

ARBRES DE CLASSIFICATION

- ▶ Une fois l'arbre de décision construit (par Recursive Binary Splitting), la *prédiction* \hat{y} associée à la data x est obtenue de la manière suivante:
- ▶ On cherche la région R_k de l'espace des features $X_1 \times \cdots \times X_p$ à laquelle appartient x ;
- ▶ On prend comme prédiction \hat{y} la target qui apparaît le plus souvent dans cette région R_k :

$$\hat{y} = \arg \max_{c=1,\dots,C} \hat{p}_{kc}$$

ARBRES DE CLASSIFICATION

- ▶ Une fois l'arbre de décision construit (par Recursive Binary Splitting), la *prédiction* \hat{y} associée à la data x est obtenue de la manière suivante:
- ▶ On cherche la région R_k de l'espace des features $X_1 \times \cdots \times X_p$ à laquelle appartient x ;
- ▶ On prend comme prédiction \hat{y} la target qui apparaît le plus souvent dans cette région R_k :

$$\hat{y} = \arg \max_{c=1,\dots,C} \hat{p}_{kc}$$

CONCLUSION

- ▶ Les arbres de décision sont intuitifs, interprétables, et peuvent gérer des variables qualitatives (discrètes) sans introduire de “dummy variables”.
- ▶ Les arbres de décision sont bien meilleurs que les méthodes de régression linéaire lorsque la relation entre les prédicteurs et la réponse est non-linéaire.
- ▶ Les arbres de décision manquent de robustesse: un petit changement dans le dataset peut engendrer un grand changement dans l'arbre associé.
- ▶ Il existe des méthodes pour contrer cela: *bagging*, *boosting* et *random forests* (cf. chapitre suivant).

CONCLUSION

- ▶ Les arbres de décision sont intuitifs, interprétables, et peuvent gérer des variables qualitatives (discrètes) sans introduire de “dummy variables”.
- ▶ Les arbres de décision sont bien meilleurs que les méthodes de régression linéaire lorsque la relation entre les prédicteurs et la réponse est non-linéaire.
- ▶ Les arbres de décision manquent de robustesse: un petit changement dans le dataset peut engendrer un grand changement dans l'arbre associé.
- ▶ Il existe des méthodes pour contrer cela: *bagging*, *boosting* et *random forests* (cf. chapitre suivant).

CONCLUSION

- ▶ Les arbres de décision sont intuitifs, interprétables, et peuvent gérer des variables qualitatives (discrètes) sans introduire de “dummy variables”.
- ▶ Les arbres de décision sont bien meilleurs que les méthodes de régression linéaire lorsque la relation entre les prédicteurs et la réponse est non-linéaire.
- ▶ Les arbres de décision manquent de robustesse: un petit changement dans le dataset peut engendrer un grand changement dans l'arbre associé.
- ▶ Il existe des méthodes pour contrer cela: *bagging*, *boosting* et *random forests* (cf. chapitre suivant).

CONCLUSION

- ▶ Les arbres de décision sont intuitifs, interprétables, et peuvent gérer des variables qualitatives (discrètes) sans introduire de “dummy variables”.
- ▶ Les arbres de décision sont bien meilleurs que les méthodes de régression linéaire lorsque la relation entre les prédicteurs et la réponse est non-linéaire.
- ▶ Les arbres de décision manquent de robustesse: un petit changement dans le dataset peut engendrer un grand changement dans l'arbre associé.
- ▶ Il existe des méthodes pour contrer cela: *bagging*, *boosting* et *random forests* (cf. chapitre suivant).

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.