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 $X_1, \ldots, X_p$ et la réponse Y.
- **Idée générale:** Un arbre de décision *partitionne* l'espace des prédicteurs $X_1 \times \cdots \times X_p$ en K régions distinctes $R_1, \ldots, R_K$ (*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.

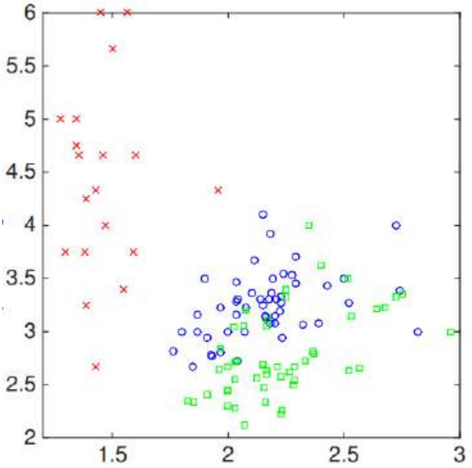
Introduction

- Les arbres de régression peuvent modéliser une relation *non-linéaire* entre les prédicteurs $X_1, \ldots, X_p$ et la réponse Y.
- **Idée générale:** Un arbre de décision *partitionne* l'espace des prédicteurs $X_1 \times \cdots \times X_p$ en K régions distinctes $R_1, \ldots, R_K$ (*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.

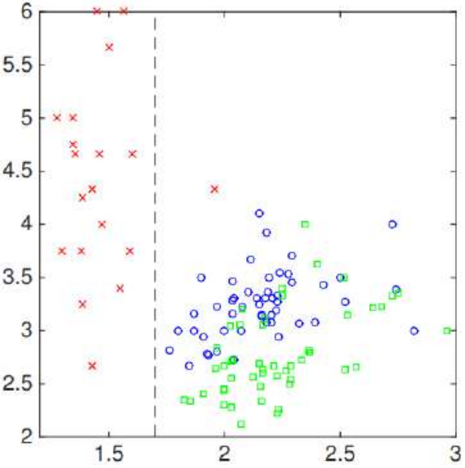
Introduction

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

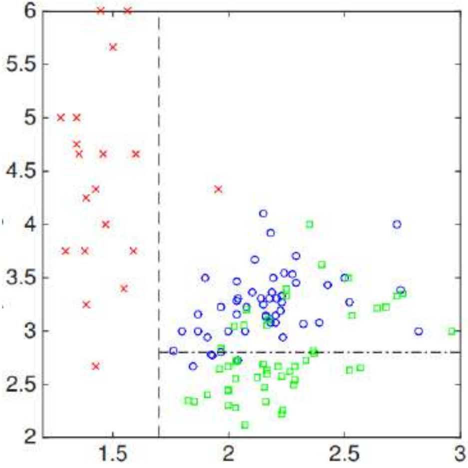
Introduction



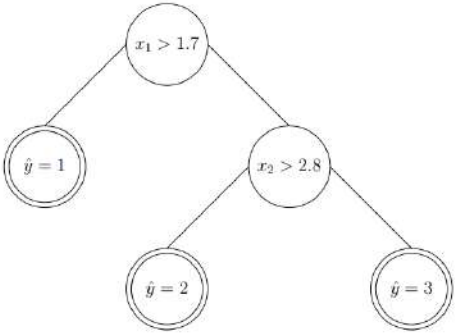
Introduction



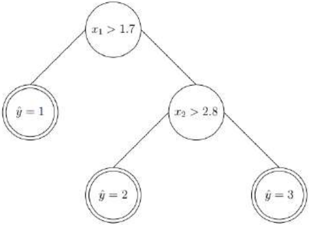
Introduction

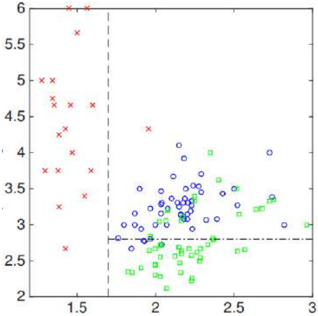


Introduction



Introduction





Arbres de régression (Regression Trees)

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

Arbres de régression (Regression Trees)

- Les **arbres de régression (régression 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 $X_1 \times \cdots \times X_p$.
- Comment construire cette partition?

Arbres de régression (Regression Trees)

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

Arbres de régression

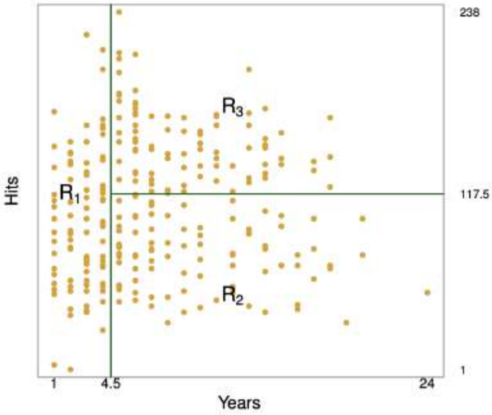
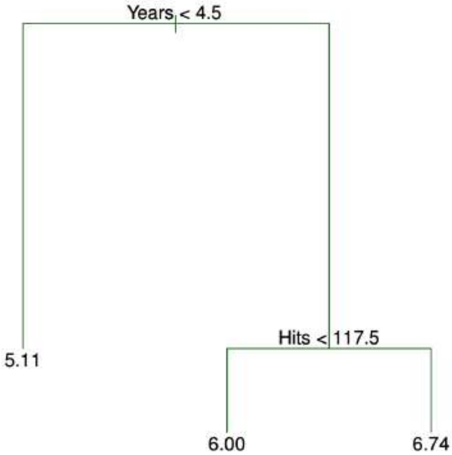


FIGURE 8.2. *The three-region partition for the* Hitters *data set from the regression tree illustrated in Figure 8.1.*

Arbres de régression



Arbres de régression

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

$$RSS(R_1, \ldots, R_K) = \sum_{k=1}^{K} \sum_{\{i : \boldsymbol{x_i} \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

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

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

Arbres de régression

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

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

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

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

Arbres de régression

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

$$RSS(R_1, \ldots, R_K) = \sum_{k=1}^{K} \sum_{\{i : \boldsymbol{x_i} \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

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

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

Arbres de régression

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

$$RSS(R_1, \ldots, R_K) = \sum_{k=1}^{K} \sum_{\{i : \boldsymbol{x_i} \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

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

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

Arbres de régression

- Computationnellement, on ne peut pas tester toutes les partitions $R_1, \ldots, R_K$ possibles (le problème de minimisation de la RSS est NP-hard).
- Pour $N = 51$ points et $K = 6$ cellules, on a $\binom{51-1}{6-1} > 2K$ partitions possibles (résultat combinatoire).
- Pour construire la partition $R_1, \ldots, R_K$, 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 $R_1, \ldots, R_K$ possibles (le problème de minimisation de la RSS est NP-hard).
- Pour $N = 51$ points et $K = 6$ cellules, on a $\binom{51-1}{6-1} > 2K$ partitions possibles (résultat combinatoire).
- Pour construire la partition $R_1, \ldots, R_K$, 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 $R_1, \ldots, R_K$ possibles (le problème de minimisation de la RSS est NP-hard).
- Pour $N = 51$ points et $K = 6$ cellules, on a $\binom{51-1}{6-1} > 2K$ partitions possibles (résultat combinatoire).
- Pour construire la partition $R_1, \ldots, R_K$, 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 $R_1, \ldots, R_K$ (*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 $R_1, \ldots, R_K$.

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 $R_1, \ldots, R_K$ (*cellules*, *boxes*) au sein desquelles les data se ressemblent.
- Les exemples $\boldsymbol{x_i}$ 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 $R_1, \ldots, R_K$.

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 $R_1, \ldots, R_K$ (*cellules*, *boxes*) au sein desquelles les data se ressemblent.
- Les exemples $\boldsymbol{x_i}$ 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 $R_1, \ldots, R_K$.

Arbres de régression

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

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

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

- Ainsi, la minimisation des variances individuelles des régions $R_1, \ldots, R_K$ correspond bien à la minimisation de la

$$RSS(R_1, \ldots, R_K) = \sum_{k=1}^{K} \sum_{\{i : \boldsymbol{x_i} \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

Arbres de régression

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

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

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

- Ainsi, la minimisation des variances individuelles des régions $R_1, \ldots, R_K$ correspond bien à la minimisation de la

$$RSS(R_1, \ldots, R_K) = \sum_{k=1}^{K} \sum_{\{i : \boldsymbol{x_i} \in R_k\}} (y_i - \bar{y}_{R_k})^2$$

Arbres de régression

- 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 dit *bon* s'il engendre une réduction de variance, i.e.,

$$\mathrm{Var}(R_1) + \mathrm{Var}(R_2) < \mathrm{Var}(R) \text{ ssi}$$

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

- 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

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

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

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

$$\mathrm{Var}(R_1) + \mathrm{Var}(R_2) < \mathrm{Var}(R) \quad \text{ssi}$$

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

- 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

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

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

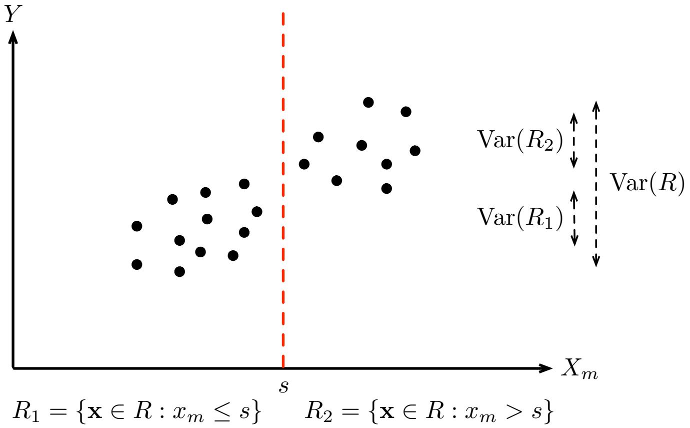
- Un split (m, s) de R en R_1 et R_2 est dit *bon* s'il engendre une réduction de variance, i.e.,

$$\mathrm{Var}(R_1) + \mathrm{Var}(R_2) < \mathrm{Var}(R) \text{ ssi}$$

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

- 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 $\mathrm{VarRed}(R, R_1, R_2)$.

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 $\mathrm{VarRed}(R, R_1, R_2)$.

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 $\mathrm{VarRed}(R, R_1, R_2)$.

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 best_split
```

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 best_split
```

Arbres de régression

Algorithm 1: best_split(dataset)

best_var_red $= -\infty$
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 best_split

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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_left], 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_left], 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_left], 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_left], 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_left], 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_left], 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_left], 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_left], 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_left], 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 $\boldsymbol{x}$ 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:

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

Arbres de régression

- Une fois l'arbre de décision construit (par Recursive Binary Splitting), la *prédiction* $\hat{y}$ associée à la data $\boldsymbol{x}$ 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 $\boldsymbol{x}$;
- On prend comme prédiction $\hat{y}$ la moyenne des targets de cette région R_k:

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

Arbres de régression

- Une fois l'arbre de décision construit (par Recursive Binary Splitting), la *prédiction* $\hat{y}$ associée à la data $\boldsymbol{x}$ 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 $\boldsymbol{x}$;
- On prend comme prédiction $\hat{y}$ la moyenne des targets de cette région R_k:

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

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 $X_1 \times \cdots \times X_p$.
- Le principe est très similaire à celui d'un arbre de régression.

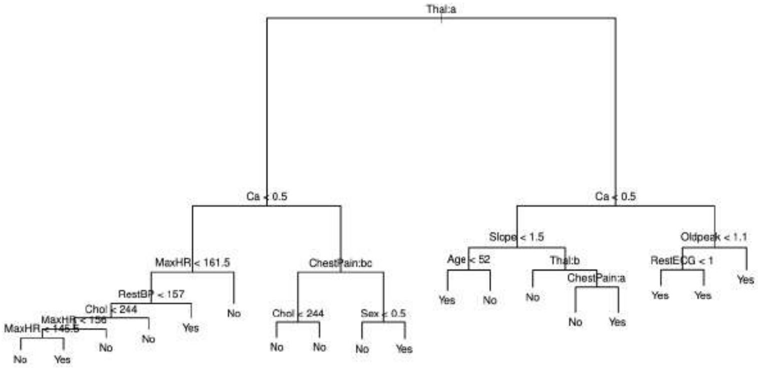
Arbres de classification (Classification Trees)

- 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* $R_1, \ldots, R_K$ de l'espace des prédicteurs $X_1 \times \cdots \times X_p$.
- 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 $X_1 \times \cdots \times X_p$.
- Le principe est très similaire à celui d'un arbre de régression.

Arbres de classification



Arbres de classification

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

Arbres de classification

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

Arbres de classification

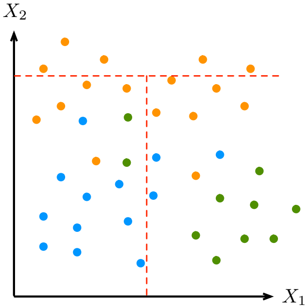
- Soit le train set $S_{\text{train}} = \{(\boldsymbol{x_i}, y_i) \in \mathbb{R}^p \times \mathcal{C} : i = 1, \ldots, N\}$.
- On cherche la partition $R_1, \ldots, R_K \subseteq X_1 \times \cdots \times X_p$ qui minimise une certaine fonction de loss.
- Cette fois, on ne considère pas la somme des carrés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

Arbres de classification

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

Arbres de classification

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



Arbres de classification

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

$$\begin{aligned} \mathrm{Gini}(R_k) &= \sum_{c=1}^{C} \hat{p}_{kc}(1-\hat{p}_{kc}) \\ \mathrm{Ent}(R_k) &= -\sum_{c=1}^{C} \hat{p}_{kc}\log(\hat{p}_{kc}) \end{aligned}$$

où $\hat{p}_{kc}$ est la proportion des y_i égaux à c dans la région R_k.

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

Arbres de classification

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

$$\begin{aligned} \mathrm{Gini}(R_k) &= \sum_{c=1}^{C} \hat{p}_{kc}(1 - \hat{p}_{kc}) \\ \mathrm{Ent}(R_k) &= -\sum_{c=1}^{C} \hat{p}_{kc} \log(\hat{p}_{kc}) \end{aligned}$$

  où $\hat{p}_{kc}$ est la proportion des y_i égaux à c dans la région R_k.

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

Arbres de classification

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

$$
\begin{aligned}
\mathrm{Gini}(R_1, \ldots, R_K) &= \sum_{k=1}^{K} \sum_{c=1}^{C} \hat{p}_{kc}(1 - \hat{p}_{kc}) \\
\mathrm{Entropy}(R_1, \ldots, R_K) &= -\sum_{k=1}^{K} \sum_{c=1}^{C} \hat{p}_{kc} \log(\hat{p}_{kc})
\end{aligned}
$$

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):

$$\mathrm{Ent}(R_1) + \mathrm{Ent}(R_2) < \mathrm{Ent}(R) \text{ ssi}$$

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

- 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):

$$\mathrm{Ent}(R_1) + \mathrm{Ent}(R_2) < \mathrm{Ent}(R) \text{ ssi}$$

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

- 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*.

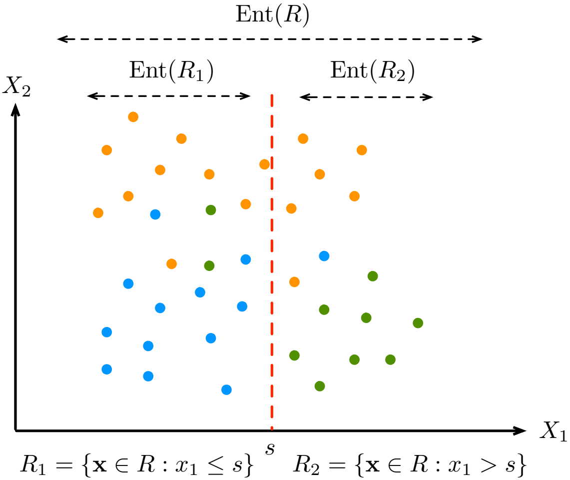
- Un split (m, s) de R en R_1 et R_2 est *bon* s'il engendre un *gain informationnel*, c'est-à-dire une réduction de l'entropie (ou de l'indice de Gini):

$$\text{Ent}(R_1) + \text{Ent}(R_2) < \text{Ent}(R) \text{ ssi}$$

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

- 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 $\mathrm{EntRed}(R, R_1, R_2)$.

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 $\mathrm{EntRed}(R, R_1, R_2)$.

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 $\mathrm{EntRed}(R, R_1, R_2)$.

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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 best_split
```

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_left], 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_left], 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_left], 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_left], 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_left], 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_left], 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_left], 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_left], 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_left], 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 $\boldsymbol{x}$ 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 $\boldsymbol{x}$;
- On prend comme prédiction $\hat{y}$ la target qui apparaît le plus souvent dans cette région R_k:

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

Arbres de classification

- Une fois l'arbre de décision construit (par Recursive Binary Splitting), la *prédiction* $\hat{y}$ associée à la data $\boldsymbol{x}$ 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 $\boldsymbol{x}$;
- 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 $\boldsymbol{x}$ 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 $\boldsymbol{x}$;
- 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 pallier 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 pallier 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 pallier 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 pallier 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.