Parallélisme du Random Forest

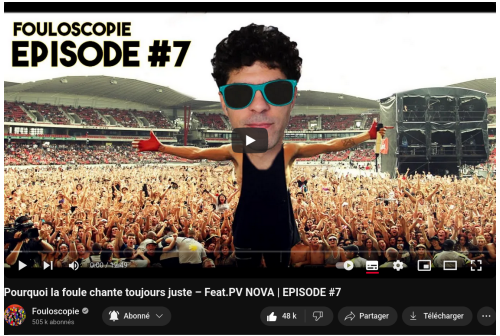
Machine Learning

Béryl HOUESSOU

Institut de Mathématiques et de Sciences Physiques
Université d'Abomey-Calavi, Bénin

1er juillet 2025

Bonjour à tous, mon thème de présentation porte sur le Parallélisme du Random Forest. Mais avant tout propos, permettez-moi cette question.



Pourquoi lors des concerts, la foule chante-t-elle toujours juste ?

Introduction

Pourquoi lors des concerts, la foule chante-t-elle toujours juste ?

Cette question peut sembler éloignée du machine learning, mais ce phénomène d'intelligence collective est au cœur des modèles ensemblistes.

De la foule au Random Forest

Dans un concert :

- Chaque chanteur fait des erreurs

Dans Random Forest :

- Chaque arbre fait des erreurs

De la foule au Random Forest
Dans un concert :
Chaque chanteur fait des erreurs
Dans Random Forest :
Chaque arbre fait des erreurs

Individuellement, chaque chanteur amateur fait des erreurs, mais collectivement la voix de la foule est toujours juste. Dans les modèles ensemblistes comme Random Forest, chaque arbre peut être vu comme un chanteur amateur, et la forêt comme la foule qui chante juste.

De la foule au Random Forest

Dans un concert :

- Chaque chanteur fait des erreurs
- Collectivement : voix juste

Dans Random Forest :

- Chaque arbre fait des erreurs
- Collectivement : prédiction plus ou moins précise

De la foule au Random Forest
Dans un concert :
Chaque chanteur fait des erreurs
Collectivement : voix juste
Dans Random Forest :
Chaque arbre fait des erreurs
Collectivement : prédiction plus ou moins précise

Individuellement, chaque chanteur amateur fait des erreurs, mais collectivement la voix de la foule est toujours juste. Dans les modèles ensemblistes comme Random Forest, chaque arbre peut être vu comme un chanteur amateur, et la forêt comme la foule qui chante juste.

De la foule au Random Forest

Dans un concert :

- Chaque chanteur fait des erreurs
- Collectivement : voix juste
- Correction mutuelle

Dans Random Forest :

- Chaque arbre fait des erreurs
- Collectivement : prédiction plus ou moins précise
- Agrégation des résultats

De la foule au Random Forest
Dans un concert :
Chaque chanteur fait des erreurs
Collectivement : voix juste
Correction mutuelle
Dans Random Forest :
Chaque arbre fait des erreurs
Collectivement : prédiction plus ou moins précise
Agrégation des résultats

Individuellement, chaque chanteur amateur fait des erreurs, mais collectivement la voix de la foule est toujours juste. Dans les modèles ensemblistes comme Random Forest, chaque arbre peut être vu comme un chanteur amateur, et la forêt comme la foule qui chante juste.

De la foule au Random Forest

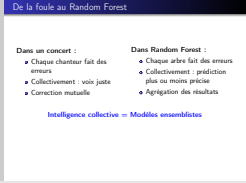
Dans un concert :

- Chaque chanteur fait des erreurs
- Collectivement : voix juste
- Correction mutuelle

Dans Random Forest :

- Chaque arbre fait des erreurs
- Collectivement : prédiction plus ou moins précise
- Agrégation des résultats

Intelligence collective = Modèles ensemblistes



Individuellement, chaque chanteur amateur fait des erreurs, mais collectivement la voix de la foule est toujours juste. Dans les modèles ensemblistes comme Random Forest, chaque arbre peut être vu comme un chanteur amateur, et la forêt comme la foule qui chante juste.

Qu'est-ce que le Random Forest ?

Définition

Algorithme d'apprentissage automatique ensembliste qui :

- Construit plusieurs arbres sur des **échantillons aléatoires**

Le Random Forest utilise une approche ensembliste pour améliorer la précision des prédictions. Cette méthode permet de réduire le risque de surapprentissage.

Qu'est-ce que le Random Forest ?

Définition

Algorithme d'apprentissage automatique ensembliste qui :

- Construit plusieurs arbres sur des **échantillons aléatoires**
- **Combine leurs résultats** pour une prédiction finale

Qu'est-ce que le Random Forest ?
Définition
Algorithme d'apprentissage automatique ensembliste qui :
- Construit plusieurs arbres sur des **échantillons aléatoires**
- **Combine leurs résultats** pour une prédiction finale

Le Random Forest utilise une approche ensembliste pour améliorer la précision des prédictions. Cette méthode permet de réduire le risque de surapprentissage.

Qu'est-ce que le Random Forest ?

Définition

Algorithme d'apprentissage automatique ensembliste qui :

- Construit plusieurs arbres sur des **échantillons aléatoires**
- **Combine leurs résultats** pour une prédiction finale
- Réduit le surapprentissage et améliore la robustesse

Qu'est-ce que le Random Forest ?
Définition
Algorithme d'apprentissage automatique ensembliste qui :
Construit plusieurs arbres sur des échantillons aléatoires
Combine leurs résultats pour une prédiction finale
Réduit le surapprentissage et améliore la robustesse

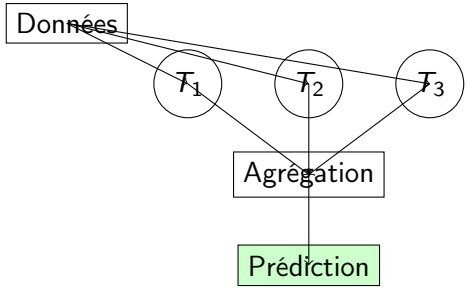
Le Random Forest utilise une approche ensembliste pour améliorer la précision des prédictions. Cette méthode permet de réduire le risque de surapprentissage.

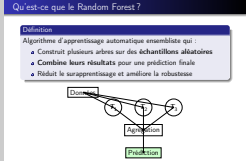
Qu'est-ce que le Random Forest ?

Définition

Algorithme d'apprentissage automatique ensembliste qui :

- Construit plusieurs arbres sur des **échantillons aléatoires**
- **Combine leurs résultats** pour une prédiction finale
- Réduit le surapprentissage et améliore la robustesse





Le Random Forest utilise une approche ensembliste pour améliorer la précision des prédictions. Cette méthode permet de réduire le risque de surapprentissage.

Algorithm 1 Random Forest

Require: Ensemble d'entraînement D, nombre d'arbres n
Ensure: Forêt de décision F

1: Initialiser $F \leftarrow \emptyset$
2: **for** $i = 1$ à n **do**
3: $D_i \leftarrow$ échantillon bootstrap de D
4: $T_i \leftarrow$ construire un arbre de décision sur D_i
5: Ajouter T_i à F
6: **end for**
7: **return** F

Algorithme Random Forest

Algorithm 1 Random Forest
Require: Ensemble d'entraînement D, nombre d'arbres n
Ensure: Forêt de décision F
1: Initialiser $F \leftarrow \emptyset$
2: **for** $i = 1$ à n **do**
3: $D_i \leftarrow$ échantillon bootstrap de D
4: $T_i \leftarrow$ construire un arbre de décision sur D_i
5: Ajouter T_i à F
6: **end for**
7: **return** F

Voici l'algorithme de base du Random Forest. Chaque arbre est construit sur un échantillon bootstrap différent.

Construction des arbres : Bagging

Bootstrap Aggregating

Principe : Créer plusieurs sous-ensembles par échantillonnage avec remise

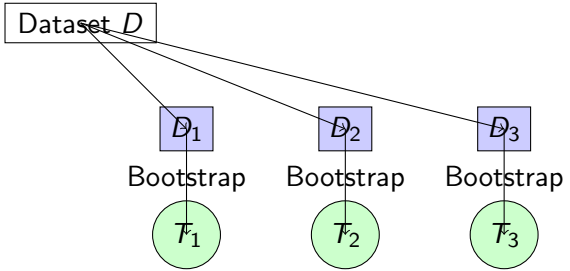
Construction des arbres : Bagging
Bootstrap Aggregating
Principe : Créer plusieurs sous-ensembles par échantillonnage avec remise

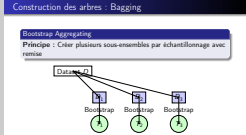
Le bagging consiste à créer plusieurs sous-ensembles de données à partir de l'ensemble d'entraînement initial. Cette approche permet d'amener de la diversité entre les arbres, ce qui réduit le surapprentissage et améliore la généralisation. Aussi les modèles d'arbres de décision étant déterministes c'est-à-dire qu'avec les mêmes données d'entraînement j'obtiens toujours les mêmes arbres, le bagging permet d'atténuer cet effet en introduisant de la variabilité.

Construction des arbres : Bagging

Bootstrap Aggregating

Principe : Créer plusieurs sous-ensembles par échantillonnage avec remise



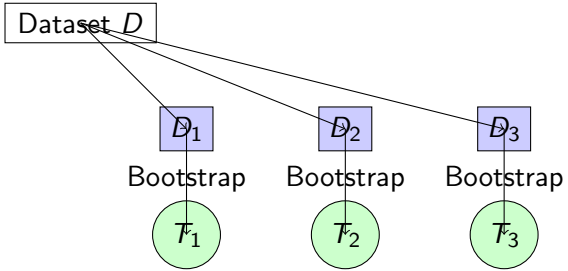


Le bagging consiste à créer plusieurs sous-ensembles de données à partir de l'ensemble d'entraînement initial. Cette approche permet d'amener de la diversité entre les arbres, ce qui réduit le surapprentissage et améliore la généralisation. Aussi les modèles d'arbres de décision étant déterministes c'est-à-dire qu'avec les mêmes données d'entraînement j'obtiens toujours les mêmes arbres, le bagging permet d'atténuer cet effet en introduisant de la variabilité.

Construction des arbres : Bagging

Bootstrap Aggregating

Principe : Créer plusieurs sous-ensembles par échantillonnage avec remise



Avantage : Rajouter de la diversité entre les arbres, ce qui réduit le surapprentissage et améliore la généralisation.

Construction des arbres : Bagging
Bootstrap Aggregating
Principe : Créer plusieurs sous-ensembles par échantillonnage avec remise
Dataset D
Bootstrap
Bootstrap
Bootstrap
Avantage : Rajouter de la diversité entre les arbres, ce qui réduit le surapprentissage et améliore la généralisation.

Le bagging consiste à créer plusieurs sous-ensembles de données à partir de l'ensemble d'entraînement initial. Cette approche permet d'amener de la diversité entre les arbres, ce qui réduit le surapprentissage et améliore la généralisation. Aussi les modèles d'arbres de décision étant déterministes c'est-à-dire qu'avec les mêmes données d'entraînement j'obtiens toujours les mêmes arbres, le bagging permet d'atténuer cet effet en introduisant de la variabilité.

Mesures de pureté

Entropie

Mesures de pureté
Entropie

Pour choisir les variables à chaque nœud, on utilise une mesure de qualité comme l'entropie ou l'indice de Gini. L'objectif est de maximiser la pureté des nœuds.

Mesures de pureté

Entropie

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Mesures de pureté
Entropie
$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$

Pour choisir les variables à chaque nœud, on utilise une mesure de qualité comme l'entropie ou l'indice de Gini. L'objectif est de maximiser la pureté des nœuds.

Mesures de pureté

Entropie

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

- Varie entre 0 et 1

Mesures de pureté
Entropie
$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$
Varie entre 0 et 1

Pour choisir les variables à chaque nœud, on utilise une mesure de qualité comme l'entropie ou l'indice de Gini. L'objectif est de maximiser la pureté des nœuds.

Mesures de pureté

Entropie

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

- Varie entre 0 et 1
- 0 = distribution homogène

Mesures de pureté
Entropie
$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$
Varie entre 0 et 1
0 = distribution homogène

Pour choisir les variables à chaque nœud, on utilise une mesure de qualité comme l'entropie ou l'indice de Gini. L'objectif est de maximiser la pureté des nœuds.

Mesures de pureté

Entropie

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

- Varie entre 0 et 1
- 0 = distribution homogène
- 1 = distribution hétérogène

Mesures de pureté
Entropie
$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$
- Varie entre 0 et 1
- 0 = distribution homogène
- 1 = distribution hétérogène

Pour choisir les variables à chaque nœud, on utilise une mesure de qualité comme l'entropie ou l'indice de Gini. L'objectif est de maximiser la pureté des nœuds.

Mesures de pureté

Entropie

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

- Varie entre 0 et 1
- 0 = distribution homogène
- 1 = distribution hétérogène

Indice de Gini

Mesures de pureté
Entropie
$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$
Varie entre 0 et 1
0 = distribution homogène
1 = distribution hétérogène
Indice de Gini

Pour choisir les variables à chaque nœud, on utilise une mesure de qualité comme l'entropie ou l'indice de Gini. L'objectif est de maximiser la pureté des nœuds.

Mesures de pureté

Entropie

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Indice de Gini

$$Gini(X) = 1 - \sum_{i=1}^{n} p_i^2$$

- Varie entre 0 et 1
- 0 = distribution homogène
- 1 = distribution hétérogène

Mesures de pureté
Entropie
$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$
Indice de Gini
$Gini(X) = 1 - \sum_{i=1}^{n} p_i^2$
- Varie entre 0 et 1
- 0 = distribution homogène
- 1 = distribution hétérogène

Pour choisir les variables à chaque nœud, on utilise une mesure de qualité comme l'entropie ou l'indice de Gini. L'objectif est de maximiser la pureté des nœuds.

Mesures de pureté

Entropie

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

- Varie entre 0 et 1
- 0 = distribution homogène
- 1 = distribution hétérogène

Indice de Gini

$$Gini(X) = 1 - \sum_{i=1}^{n} p_i^2$$

- Varie entre 0 et 0,5

Mesures de pureté
Entropie
$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$
Varie entre 0 et 1
0 = distribution homogène
1 = distribution hétérogène
Indice de Gini
$Gini(X) = 1 - \sum_{i=1}^{n} p_i^2$
Varie entre 0 et 0,5

Pour choisir les variables à chaque nœud, on utilise une mesure de qualité comme l'entropie ou l'indice de Gini. L'objectif est de maximiser la pureté des nœuds.

Mesures de pureté

Entropie

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

- Varie entre 0 et 1
- 0 = distribution homogène
- 1 = distribution hétérogène

Indice de Gini

$$Gini(X) = 1 - \sum_{i=1}^{n} p_i^2$$

- Varie entre 0 et 0,5
- 0 = pureté parfaite

Mesures de pureté
Entropie
$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$
Varie entre 0 et 1
0 = distribution homogène
1 = distribution hétérogène
Indice de Gini
$Gini(X) = 1 - \sum_{i=1}^{n} p_i^2$
Varie entre 0 et 0,5
0 = pureté parfaite

Pour choisir les variables à chaque nœud, on utilise une mesure de qualité comme l'entropie ou l'indice de Gini. L'objectif est de maximiser la pureté des nœuds.

Mesures de pureté

Entropie

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

- Varie entre 0 et 1
- 0 = distribution homogène
- 1 = distribution hétérogène

Indice de Gini

$$Gini(X) = 1 - \sum_{i=1}^{n} p_i^2$$

- Varie entre 0 et 0,5
- 0 = pureté parfaite
- 0,5 = désordre maximal

Mesures de pureté
Entropie
$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$
Varie entre 0 et 1
0 = distribution homogène
1 = distribution hétérogène
Indice de Gini
$Gini(X) = 1 - \sum_{i=1}^{n} p_i^2$
Varie entre 0 et 0,5
0 = pureté parfaite
0,5 = désordre maximal

Pour choisir les variables à chaque nœud, on utilise une mesure de qualité comme l'entropie ou l'indice de Gini. L'objectif est de maximiser la pureté des nœuds.

Construction d'un arbre

Algorithm 2 Construction d'un arbre de décision

Require: Ensemble D, profondeur max d, échantillons min m
Ensure: Arbre de décision T

1: **if** profondeur max atteinte OU $|D| < m$ **then**
2: **return** feuille (classe majoritaire/moyenne)
3: **end if**
4: Sélectionner sous-ensemble aléatoire de caractéristiques
5: **for** chaque caractéristique **do**
6: Calculer gain d'impureté pour chaque seuil
7: **end for**
8: Choisir meilleure caractéristique et seuil
9: Créer D_{gauche} et D_{droite}
10: $T_{gauche} \leftarrow$ récursion sur D_{gauche}
11: $T_{droite} \leftarrow$ récursion sur D_{droite}
12: **return** T

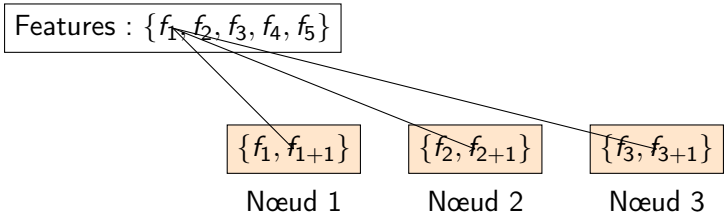
Construction d'un arbre
Algorithm 2 Construction d'un arbre de décision
Require: Ensemble D, profondeur max d, échantillons min m
Ensure: Arbre de décision T
1: if profondeur max atteinte OU $|D| < m$ then
2: return feuille (classe majoritaire/moyenne)
3: end if
4: Sélectionner sous-ensemble aléatoire de caractéristiques
5: for chaque caractéristique do
6: Calculer gain d'impureté pour chaque seuil
7: end for
8: Choisir meilleure caractéristique et seuil
9: Créer D_{gauche} et D_{droite}
10: $T_{gauche} \leftarrow$ récursion sur D_{gauche}
11: $T_{droite} \leftarrow$ récursion sur D_{droite}
12: return T

Voici l'algorithme de construction d'un arbre dans le contexte du Random Forest. Une petite piqure de rappel : Un arbre de décision est un modèle d'apprentissage supervisé qui se construit selon un processus récursif de partitionnement des données. On commence par un nœud racine représentant l'ensemble complet des données, puis on divise cet ensemble en sous-ensembles plus petits, de manière à rendre ces groupes aussi homogènes que possible par rapport à la variable cible. Suivant cette logique, il nous faut trouver des critères d'arrêt pour notre algo de construction d'arbre (étant récursif) donc ici nous avons la profondeur maximale et le nombre minimum d'échantillons par feuille. Ces deux critères sont importants pour éviter le surapprentissage, parce qu'on ne veut pas que l'arbre devienne trop complexe et s'adapte trop aux données d'entraînement, ce qui pourrait nuire à sa capacité de généralisation.

Sélection aléatoire de caractéristiques : Feature bagging

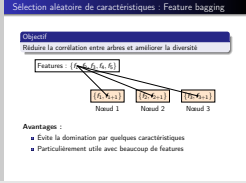
Objectif

Réduire la corrélation entre arbres et améliorer la diversité



Avantages :

- Évite la domination par quelques caractéristiques
- Particulièrement utile avec beaucoup de features



À chaque nœud de l'arbre, un sous-ensemble aléatoire de caractéristiques est sélectionné. Cela permet de réduire la corrélation entre les arbres et améliore la qualité globale des prédictions.

Agrégation des résultats

Classification

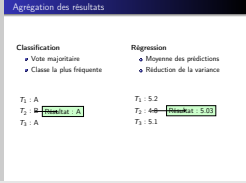
- Vote majoritaire
- Classe la plus fréquente

T_1 : A
T_2 : B — Résultat : A
T_3 : A

Régression

- Moyenne des prédictions
- Réduction de la variance

T_1 : 5.2
T_2 : 4.8 — Résultat : 5.03
T_3 : 5.1



L'agrégation est la dernière étape du Random Forest. Elle combine les prédictions de tous les arbres pour obtenir une prédiction finale plus robuste.

Avantages et inconvénients du Random Forest

Avantages

- **Robustesse :** Réduit le surapprentissage
- **Précision :** Souvent supérieure aux arbres individuels
- **Versatilité :** Fonctionne sur différents types de données
- **Gestion du bruit :** Résistant aux données aberrantes
- **Pas de normalisation :** Invariant aux transformations monotones

Avantages et inconvénients du Random Forest
Avantages
Robustesse : Réduit le surapprentissage
Précision : Souvent supérieure aux arbres individuels
Versatilité : Fonctionne sur différents types de données
Gestion du bruit : Résistant aux données aberrantes
Pas de normalisation : Invariant aux transformations monotones

Le Random Forest présente plusieurs avantages qui en font un algorithme populaire : robustesse, précision, versatilité. Toutefois, il n'est pas sans défauts, notamment en termes de coût computationnel. C'est là que la parallélisation entre en jeu.

Avantages et inconvénients du Random Forest

Avantages

- **Robustesse :** Réduit le surapprentissage
- **Précision :** Souvent supérieure aux arbres individuels
- **Versatilité :** Fonctionne sur différents types de données
- **Gestion du bruit :** Résistant aux données aberrantes
- **Pas de normalisation :** Invariant aux transformations monotones

Inconvénients

- **Coût computationnel :** Temps de calcul élevé
- **Mémoire :** Consommation importante
- **Interprétabilité :** Moins lisible qu'un arbre unique
- **Surapprentissage :** Possible avec trop d'arbres
- **Biais :** Peut favoriser les features avec plus de modalités

Le Random Forest présente plusieurs avantages qui en font un algorithme populaire : robustesse, précision, versatilité. Toutefois, il n'est pas sans défauts, notamment en termes de coût computationnel. C'est là que la parallélisation entre en jeu.

Avantages et inconvénients du Random Forest

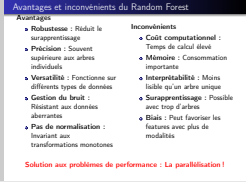
Avantages

- **Robustesse :** Réduit le surapprentissage
- **Précision :** Souvent supérieure aux arbres individuels
- **Versatilité :** Fonctionne sur différents types de données
- **Gestion du bruit :** Résistant aux données aberrantes
- **Pas de normalisation :** Invariant aux transformations monotones

Inconvénients

- **Coût computationnel :** Temps de calcul élevé
- **Mémoire :** Consommation importante
- **Interprétabilité :** Moins lisible qu'un arbre unique
- **Surapprentissage :** Possible avec trop d'arbres
- **Biais :** Peut favoriser les features avec plus de modalités

Solution aux problèmes de performance : La parallélisation !



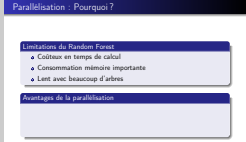
Le Random Forest présente plusieurs avantages qui en font un algorithme populaire : robustesse, précision, versatilité. Toutefois, il n'est pas sans défauts, notamment en termes de coût computationnel. C'est là que la parallélisation entre en jeu.

Parallélisation : Pourquoi ?

Limitations du Random Forest

- Coûteux en temps de calcul
- Consommation mémoire importante
- Lent avec beaucoup d'arbres

Avantages de la parallélisation



Le Random Forest n'est pas sans défauts. Il peut être coûteux en termes de temps de calcul et de mémoire. C'est là que la parallélisation entre en jeu.

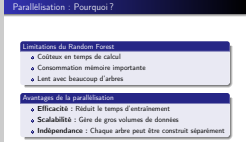
Parallélisation : Pourquoi ?

Limitations du Random Forest

- Coûteux en temps de calcul
- Consommation mémoire importante
- Lent avec beaucoup d'arbres

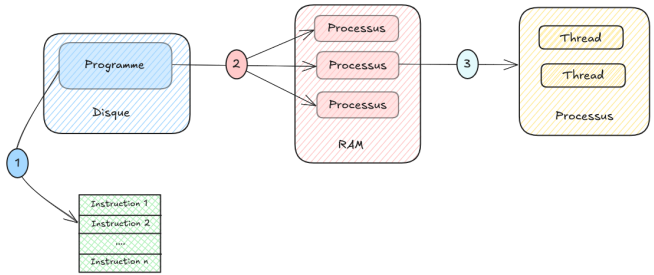
Avantages de la parallélisation

- **Efficacité** : Réduit le temps d'entraînement
- **Scalabilité** : Gère de gros volumes de données
- **Indépendance** : Chaque arbre peut être construit séparément

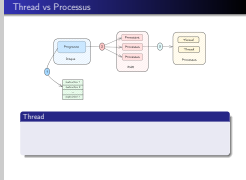


Le Random Forest n'est pas sans défauts. Il peut être coûteux en termes de temps de calcul et de mémoire. C'est là que la parallélisation entre en jeu.

Thread vs Processus

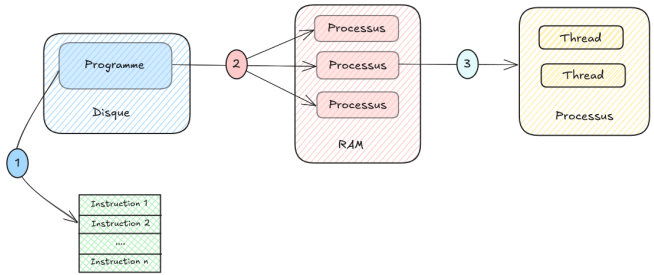


Thread



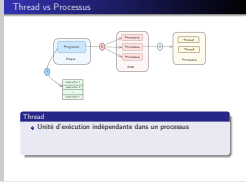
Un thread est une unité d'exécution indépendante au sein d'un processus. Un processus peut donc contenir plusieurs threads, chacun exécutant une tâche spécifique. UN processus étant un programme en cours d'exécution, il peut contenir plusieurs threads qui partagent la même mémoire et les mêmes ressources.

Thread vs Processus



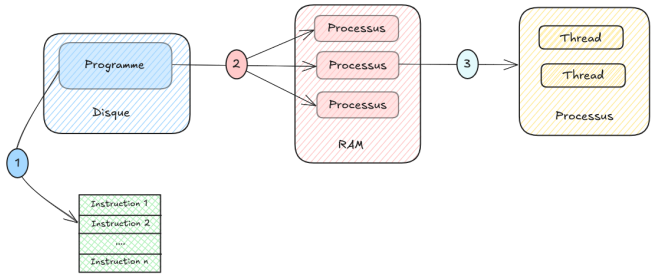
Thread

- Unité d'exécution indépendante dans un processus



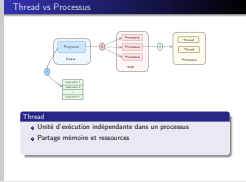
Un thread est une unité d'exécution indépendante au sein d'un processus. Un processus peut donc contenir plusieurs threads, chacun exécutant une tâche spécifique. UN processus étant un programme en cours d'exécution, il peut contenir plusieurs threads qui partagent la même mémoire et les mêmes ressources.

Thread vs Processus



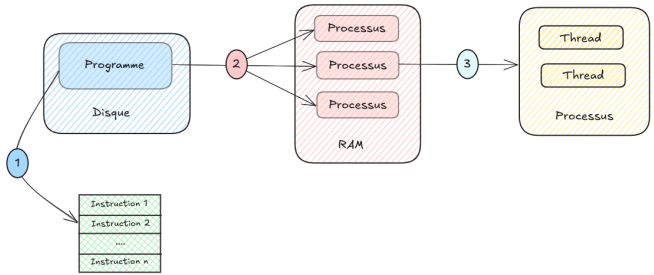
Thread

- Unité d'exécution indépendante dans un processus
- Partage mémoire et ressources



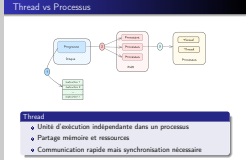
Un thread est une unité d'exécution indépendante au sein d'un processus. Un processus peut donc contenir plusieurs threads, chacun exécutant une tâche spécifique. UN processus étant un programme en cours d'exécution, il peut contenir plusieurs threads qui partagent la même mémoire et les mêmes ressources.

Thread vs Processus



Thread

- Unité d'exécution indépendante dans un processus
- Partage mémoire et ressources
- Communication rapide mais synchronisation nécessaire



Un thread est une unité d'exécution indépendante au sein d'un processus. Un processus peut donc contenir plusieurs threads, chacun exécutant une tâche spécifique. UN processus étant un programme en cours d'exécution, il peut contenir plusieurs threads qui partagent la même mémoire et les mêmes ressources.

Parallélisation avec std : :thread

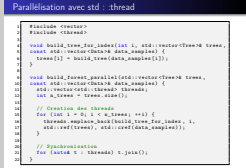
```cpp
#include <vector>
#include <thread>

void build_tree_for_index(int i, std::vector<Tree>& trees,
const std::vector<Data>& data_samples) {
  trees[i] = build_tree(data_samples[i]);
}

void build_forest_parallel(std::vector<Tree>& trees,
const std::vector<Data>& data_samples) {
  std::vector<std::thread> threads;
  int n_trees = trees.size();

  // Creation des threads
  for (int i = 0; i < n_trees; ++i) {
    threads.emplace_back(build_tree_for_index, i,
    std::ref(trees), std::cref(data_samples));
  }

  // Synchronisation
  for (auto& t : threads) t.join();
}
```

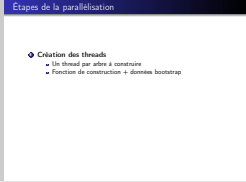



`threads.emplace_back(...)` ajoute un nouvel élément à la fin du vecteur `threads`. `emplace_back` construit un nouvel objet directement dans la mémoire du vecteur, évitant la création d'un objet temporaire. Les arguments passés à `emplace_back` sont utilisés pour construire le nouveau `std::thread`. Le premier, `build_tree_for_index`, est la fonction que ce nouveau thread exécutera. Les autres sont ceux qui seront passés à cette dernière lorsque le thread démarrera.

- `i` : l'index de l'arbre à construit dans le vecteur trees.
- `std::ref(trees)` : Le constructeur `std::thread` copie par défaut ses arguments. `std::ref` est un wrapper qui garantit que l'objet `trees` est passé **par référence**. Cela signifie que la fonction du thread peut accéder directement au conteneur `trees` d'origine et le modifier, plutôt qu'à une copie. Cela est essentiel car la tâche du thread consiste à ajouter un nouvel arbre à ce conteneur partagé.
- `std::cref(data_samples)` : similaire à `std::ref`, `std::cref` transmet l'objet `data_samples` **par référence constante**. L'aspect

Étapes de la parallélisation

1. **Création des threads**
   - Un thread par arbre à construire
   - Fonction de construction + données bootstrap



Voici comment procéder pour la parallélisation : création des threads, synchronisation et gestion des résultats. Il est important de s'assurer que chaque thread écrit dans une zone mémoire distincte.

Étapes de la parallélisation

1. **Création des threads**
   - Un thread par arbre à construire
   - Fonction de construction + données bootstrap
2. **Synchronisation**
   - Attendre tous les threads (`join()`)

Étapes de la parallélisation
Création des threads
Un thread par arbre à construire
Fonction de construction + données bootstrap
Synchronisation
Attendre tous les threads (join())

Voici comment procéder pour la parallélisation : création des threads, synchronisation et gestion des résultats. Il est important de s'assurer que chaque thread écrit dans une zone mémoire distincte.

Étapes de la parallélisation

1. **Création des threads**
   - Un thread par arbre à construire
   - Fonction de construction + données bootstrap
2. **Synchronisation**
   - Attendre tous les threads (`join()`)
3. **Gestion des résultats**
   - Chaque thread écrit dans une zone mémoire distincte
   - Pas de partage de données modifiables

Étapes de la parallélisation
Création des threads
Un thread par arbre à construire
Fonction de construction + données bootstrap
Synchronisation
Attendre tous les threads (join())
Gestion des résultats
Chaque thread écrit dans une zone mémoire distincte
Pas de partage de données modifiables

Voici comment procéder pour la parallélisation : création des threads, synchronisation et gestion des résultats. Il est important de s'assurer que chaque thread écrit dans une zone mémoire distincte.

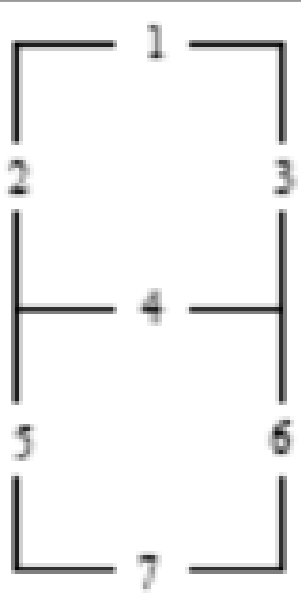
Exercice pratique : Reconnaissance des chiffres

Problème : Reconnaître les chiffres 0-9 à partir de 7 segments

1234567890

Représentation :

- Vecteur binaire de 7 éléments
- 1 = segment allumé, 0 = éteint
- Exemple : "8" = [1,1,1,1,1,1,1]



Exercice pratique : Reconnaissance des chiffres
Problème : Reconnaître les chiffres 0-9 à partir de 7 segments
1234567890
Représentation :
Vecteur binaire de 7 éléments
1 = segment allumé, 0 = éteint
Exemple : "8" = [1,1,1,1,1,1,1]

Dans cet exercice pratique, nous avons 10 chiffres représentés par des combinaisons de 7 segments lumineux. Chaque chiffre est un vecteur binaire de 7 dimensions.

Conclusion

Points clés

- **Random Forest** : Intelligence collective appliquée au ML
- **Robustesse** : Réduction du surapprentissage par agrégation
- **Parallélisation** : Accélération grâce à l'indépendance des arbres

Conclusion
Points clés
Random Forest : Intelligence collective appliquée au ML
Robustesse : Réduction du surapprentissage par agrégation
Parallélisation : Accélération grâce à l'indépendance des arbres

En conclusion, le Random Forest est un algorithme puissant qui utilise l'intelligence collective pour améliorer les prédictions. Sa parallélisation permet d'exploiter efficacement les ressources matérielles modernes.

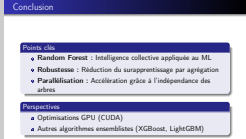
Conclusion

Points clés

- **Random Forest** : Intelligence collective appliquée au ML
- **Robustesse** : Réduction du surapprentissage par agrégation
- **Parallélisation** : Accélération grâce à l'indépendance des arbres

Perspectives

- Optimisations GPU (CUDA)
- Autres algorithmes ensemblistes (XGBoost, LightGBM)



En conclusion, le Random Forest est un algorithme puissant qui utilise l'intelligence collective pour améliorer les prédictions. Sa parallélisation permet d'exploiter efficacement les ressources matérielles modernes.

Merci, pour votre attention !

Merci, pour votre attention !