Rapport Technique

Implémentation de l'Algorithme CART

Classification and Regression Trees de Breiman (1984)

Amllal Amine - Boughnam Houda - Khald Adam

17 décembre 2025

Résumé

Ce rapport présente une analyse complète de l'algorithme CART (Classification and Regression Trees) développé par Leo Breiman et ses collaborateurs en 1984. L'algorithme CART représente une contribution majeure au domaine de l'apprentissage automatique et des statistiques. Ce document couvre les fondements théoriques, les aspects mathématiques, l'implémentation pratique ainsi que l'analyse détaillée de l'implémentation open-source disponible sur le repository GitHub `Adamkhald/breiman-cart`. Le rapport inclut des pseudo-codes détaillés, des formulations mathématiques rigoureuses et des exemples d'application pratiques.

Table des matières

1 Introduction

1.1 Contexte Historique

L'algorithme CART (Classification and Regression Trees) a été introduit en 1984 par Leo Breiman, Jerome Friedman, Richard Olshen et Charles Stone dans leur ouvrage fondateur *Classification and Regression Trees*. Cette méthode révolutionnaire a transformé l'analyse de données en proposant une approche non-paramétrique pour la classification et la régression.

Leo Breiman, statisticien de renom à l'Université de Californie à Berkeley, a également apporté des contributions majeures avec le *bagging* (Bootstrap Aggregating) en 1996 et les *Random Forests* en 2001, qui sont des extensions directes de CART.

1.2 Objectifs du Rapport

Ce rapport vise à :
— Présenter les fondements théoriques de l'algorithme CART
— Développer les formulations mathématiques rigoureuses
— Fournir des pseudo-codes détaillés pour l'implémentation
— Analyser en profondeur l'implémentation du repository GitHub `Adamkhald/breiman-cart`
— Discuter des applications pratiques et des considérations d'implémentation
— Comparer avec d'autres implémentations notamment scikit-learn

1.3 Structure du Repository GitHub

Le repository `Adamkhald/breiman-cart` a été récemment restructuré (Version 0.2.0) pour offrir une API plus conviviale et une meilleure expérience utilisateur :

— **breiman_cart.py** : API unifiée principale (2000+ lignes)
  — `BRCRegression` : Classe pour tâches de régression
  — `BRCClassification` : Classe pour tâches de classification
  — `BRCInference` : Classe pour inférence avec modèles entraînés
— **breiman_cart/** : Package interne modulaire
  — `__init__.py` : Initialisation du package
  — `node.py` : Structure de nœuds avec support catégoriel
  — `splitter.py` : Logique de division (Gini, MSE, sous-ensembles)
  — `tree.py` : Implémentation interne des arbres
  — `pruning.py` : Élagage par coût-complexité
— **tests/** : Suite de tests unitaires
— **docs/** : Documentation théorique (theory.tex)
— **demo.py** : Script de démonstration
— **quickstart.py** : Guide de démarrage rapide
— **test_new_api.py** : Tests pour la nouvelle API
— **fix_syntax.py** : Utilitaire de correction syntaxique
— **requirements.txt** : Dépendances (NumPy, Pandas, pickle, json)

2 Fondements Théoriques

2.1 Principe Général

CART est un algorithme de partitionnement récursif qui construit un arbre de décision binaire. Le principe fondamental repose sur :

1. **Partitionnement récursif** : Division successive de l'espace des variables explicatives en régions homogènes
2. **Modélisation locale** : Estimation de modèles simples (constantes) dans chaque région de la partition
3. **Structure binaire stricte** : Chaque nœud génère exactement deux branches (caractéristique distinctive de CART)
4. **Approche gloutonne** : Sélection locale optimale à chaque division sans rétroaction

2.2 Définitions Mathématiques

2.2.1 Espace de Données

Soit un ensemble de n observations :

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n} \tag{1}$$

où :
— $\mathbf{x}_i \in \mathbb{R}^p$: vecteur de p variables explicatives (features)
— $y_i \in \mathcal{Y}$: variable réponse (target)
  — Classification : $\mathcal{Y} = \{1, 2, \ldots, K\}$ (classes discrètes)
  — Régression : $\mathcal{Y} = \mathbb{R}$ (valeur continue)

2.2.2 Structure d'Arbre

Un arbre T est défini par :
— $\mathcal{N}(T)$: ensemble des nœuds internes (nœuds de décision)
— $\widetilde{T}$: ensemble des nœuds terminaux (feuilles)
— $|\widetilde{T}|$: nombre de feuilles (mesure de complexité de l'arbre)
— Pour chaque nœud interne : une règle de division (j, s) où j est l'indice de la variable et s le seuil
— Pour chaque feuille : une prédiction constante

2.3 Règles de Division

2.3.1 Variables Continues

Pour une variable continue X_j, une division au point s crée deux régions :

$$R_L(j, s) = \{\mathbf{x} : x_j \leq s\} \quad \text{(région gauche)} \tag{2}$$

$$R_R(j, s) = \{\mathbf{x} : x_j > s\} \quad \text{(région droite)} \tag{3}$$

Recherche du seuil optimal : Pour une variable continue, les points de division candidats sont les points médians entre deux valeurs consécutives distinctes :

$$s_k = \frac{x_{(k)} + x_{(k+1)}}{2}, \quad k = 1, \ldots, m-1 \tag{4}$$

où $x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(m)}$ sont les valeurs distinctes triées.

2.3.2 Variables Catégorielles

Pour une variable catégorielle X_j avec catégories $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$, la division utilise des sous-ensembles :

$$R_L(j, S) = \{\mathbf{x} : x_j \in S\} \tag{5}$$
$$R_R(j, S) = \{\mathbf{x} : x_j \in \mathcal{C} \setminus S\} \tag{6}$$

où $S \subset \mathcal{C}$.

Note importante : Le nombre de divisions possibles pour m catégories est $2^{m-1} - 1$. Pour la classification binaire, Breiman a montré qu'on peut réduire cette complexité en ordonnant les catégories selon leur proportion de classe positive.

3 Critères de Division

3.1 Classification : Indice de Gini

L'impureté de Gini pour un nœud t est définie par :

$$I_{\text{Gini}}(t) = \sum_{k=1}^{K} p_k(t) \cdot (1 - p_k(t)) = 1 - \sum_{k=1}^{K} p_k^2(t) \tag{7}$$

où $p_k(t)$ est la proportion d'observations de classe k dans le nœud t :

$$p_k(t) = \frac{1}{n_t} \sum_{i \in t} \mathbb{1}(y_i = k) \tag{8}$$

Interprétation : L'indice de Gini mesure la probabilité de mal classer un élément choisi aléatoirement si on lui attribue une classe aléatoire selon la distribution des classes dans le nœud.

Propriétés :
— $I_{\text{Gini}}(t) = 0$ si le nœud est pur (toutes les observations de même classe)
— $I_{\text{Gini}}(t)$ est maximale pour une distribution uniforme
— Pour $K = 2$: $I_{\text{Gini}}(t) = 2p(1-p)$ où p est la proportion de la classe positive

La réduction d'impureté pour une division est :

$$\Delta I(s, t) = I(t) - \frac{n_L}{n_t} I(t_L) - \frac{n_R}{n_t} I(t_R) \tag{9}$$

3.2 Classification : Entropie

L'entropie de Shannon peut également être utilisée :

$$I_{\text{Entropy}}(t) = -\sum_{k=1}^{K} p_k(t) \log_2 p_k(t) \tag{10}$$

Comparaison Gini vs Entropie :
— Les deux critères produisent généralement des arbres similaires
— L'entropie est légèrement plus coûteuse à calculer (logarithmes)
— Gini favorise les divisions qui isolent la classe majoritaire
— Entropie favorise les divisions équilibrées

3.3 Régression : Erreur Quadratique Moyenne

Pour la régression, l'objectif est de minimiser la somme des carrés résiduelle (RSS) :

$$\text{RSS}(T) = \sum_{j=1}^{|\widetilde{T}|} \sum_{i \in R_j} (y_i - \hat{y}_j)^2 \tag{11}$$

où $\hat{y}_j$ est la prédiction dans la région R_j :

$$\hat{y}_j = \frac{1}{n_j} \sum_{i \in R_j} y_i \quad \text{(moyenne des valeurs cibles)} \tag{12}$$

La réduction de variance pour une division est :

$$\Delta\text{Var}(s, t) = \text{Var}(t) - \frac{n_L}{n_t}\text{Var}(t_L) - \frac{n_R}{n_t}\text{Var}(t_R) \tag{13}$$

avec :

$$\text{Var}(t) = \frac{1}{n_t} \sum_{i \in t} (y_i - \bar{y}_t)^2 \tag{14}$$

4 Algorithmes et Pseudo-Codes

4.1 Algorithme Principal de Construction

Algorithm 1 Construction d'Arbre CART

```
procedure BuildCART(𝒟, params)
    T ← CreateRootNode(𝒟)
    queue ← [T.root]
    while queue ≠ ∅ do
        t ← queue.pop()
        if StoppingCriterion(t, params) then
            MakeLeaf(t)
            continue
        end if
        (j*, s*) ← FindBestSplit(t)
        if (j*, s*) = None then
            MakeLeaf(t)
            continue
        end if
        (t_L, t_R) ← Split(t, j*, s*)
        queue.append(t_L)
        queue.append(t_R)
    end while
    return T
end procedure
```

4.2 Recherche de la Meilleure Division

Algorithm 2 Recherche de la Meilleure Division

```
1: procedure FindBestSplit(t)
2:     best_gain ← −∞
3:     best_split ← None
4:     for j = 1 to p do                                  ▷ Pour chaque variable
5:         if X_j est continue then
6:             candidates ← GetUniqueSortedValues(X_j, t)
7:             for s ∈ candidates do
8:                 gain ← ComputeGain(t, j, s)
9:                 if gain > best_gain then
10:                    best_gain ← gain
11:                    best_split ← (j, s)
12:                end if
13:            end for
14:        else                                           ▷ Variable catégorielle
15:            subsets ← GenerateSubsets(X_j)
16:            for S ∈ subsets do
17:                gain ← ComputeGain(t, j, S)
18:                if gain > best_gain then
19:                    best_gain ← gain
20:                    best_split ← (j, S)
21:                end if
22:            end for
23:        end if
24:    end for
25:    return best_split
26: end procedure
```

4.3 Calcul du Gain (Classification)

Algorithm 3 Calcul du Gain d'Information (Gini)

procedure ComputeGain(t, j, s)
    $(t_L, t_R) \leftarrow$ PartitionNode(t, j, s)
    $n_t \leftarrow$ —samples in t—
    $n_L \leftarrow$ —samples in t_L—
    $n_R \leftarrow$ —samples in t_R—
    $I_t \leftarrow$ GiniImpurity(t)
    $I_L \leftarrow$ GiniImpurity(t_L)
    $I_R \leftarrow$ GiniImpurity(t_R)
    gain $\leftarrow I_t - \frac{n_L}{n_t} I_L - \frac{n_R}{n_t} I_R$
    return gain
end procedure
procedure GiniImpurity(t)
    gini $\leftarrow 1$
    for $k = 1$ **to** K **do**
        $p_k \leftarrow \frac{1}{n_t} \sum_{i \in t} \mathbb{1}(y_i = k)$
        gini $\leftarrow$ gini $- p_k^2$
    end for
    return gini
end procedure

4.4 Calcul du Gain (Régression)

Algorithm 4 Calcul du Gain pour Régression

procedure ComputeRegressionGain(t, j, s)
    $(t_L, t_R) \leftarrow$ PartitionNode(t, j, s)
    $n_t \leftarrow$ —samples in t—
    $n_L \leftarrow$ —samples in t_L—
    $n_R \leftarrow$ —samples in t_R—
    $\text{Var}_t \leftarrow$ Variance(t)
    $\text{Var}_L \leftarrow$ Variance(t_L)
    $\text{Var}_R \leftarrow$ Variance(t_R)
    gain $\leftarrow \text{Var}_t - \frac{n_L}{n_t} \text{Var}_L - \frac{n_R}{n_t} \text{Var}_R$
    return gain
end procedure
procedure Variance(t)
    $\bar{y} \leftarrow \frac{1}{n_t} \sum_{i \in t} y_i$
    var $\leftarrow \frac{1}{n_t} \sum_{i \in t} (y_i - \bar{y})^2$
    return var
end procedure

5 Élagage (Pruning)

5.1 Principe de l'Élagage par Coût-Complexité

L'élagage vise à réduire le sur-apprentissage en simplifiant l'arbre. Le critère de coût-complexité est défini par :

$$R_\alpha(T) = R(T) + \alpha|\widetilde{T}| \tag{15}$$

où :
— $R(T)$: erreur de l'arbre sur l'ensemble d'apprentissage
— $\alpha \geq 0$: paramètre de complexité (hyperparamètre de régularisation)
— $|\widetilde{T}|$: nombre de feuilles (pénalisation de la complexité)
Intuition :
— $\alpha = 0$: arbre complet (sur-apprentissage possible)
— $\alpha \to \infty$: arbre réduit à la racine (sous-apprentissage)
— α optimal : compromis biais-variance optimal

5.2 Séquence d'Élagage

Pour chaque valeur de α, on trouve l'arbre optimal :

$$\hat{T}_\alpha = \arg\min_{T \subseteq T_{\max}} R_\alpha(T) \tag{16}$$

Le théorème de Breiman garantit l'existence d'une séquence finie d'arbres imbriqués :

$$T_{\max} \succ T_1 \succ T_2 \succ \cdots \succ T_M = \{\text{root}\} \tag{17}$$

5.3 Algorithme d'Élagage

Algorithm 5 Élagage par Coût-Complexité

procedure PRUNETREE(T, $\mathcal{D}_{\text{val}}$, y_{val})
  $\alpha_{\text{sequence}} \leftarrow$ ComputeAlphaSequence(T)
  trees $\leftarrow$ []
  for $\alpha \in \alpha_{\text{sequence}}$ **do**
    $T_\alpha \leftarrow$ PruneAtAlpha(T, α)
    score $\leftarrow$ Evaluate(T_α, $\mathcal{D}_{\text{val}}$, y_{val})
    trees.append((T_α, score))
  end for
  $T^* \leftarrow$ SelectBestTree(trees)
  return T^*
end procedure
procedure COMPUTEALPHASEQUENCE(T)
  $\alpha_{\text{seq}} \leftarrow$ []
  $T_{\text{current}} \leftarrow T$
  while T_{current} has more than 1 leaf **do**
    $\alpha_{\min} \leftarrow \infty$
    for $t \in$ internal nodes of T_{current} **do**
      $R_t \leftarrow$ Error(t as leaf)
      $R_{T_t} \leftarrow$ Error(subtree at t)
      $|\widetilde{T}_t| \leftarrow$ Number of leaves in subtree at t
      $\alpha_t \leftarrow \frac{R_t - R_{T_t}}{|\widetilde{T}_t| - 1}$
      if $\alpha_t < \alpha_{\min}$ **then**
        $\alpha_{\min} \leftarrow \alpha_t$
        $t_{\text{prune}} \leftarrow t$
      end if
    end for
    α_{seq}.append($\alpha_{\min}$)
    Prune subtree at t_{prune}
  end while
  return α_{seq}
end procedure

6 Critères d'Arrêt

6.1 Paramètres de Contrôle

Les critères d'arrêt principaux incluent :

1. **Profondeur maximale** : $\text{depth}(t) \geq \text{max\_depth}$
2. **Nombre minimal d'échantillons pour diviser** : $n_t < \text{min\_samples\_split}$
3. **Amélioration minimale** : $\Delta I < \text{min\_impurity\_decrease}$
4. **Nombre minimal d'échantillons par feuille** : $\min(n_L, n_R) < \text{min\_samples\_leaf}$
5. **Pureté du nœud** : $I(t) = 0$ (classification)

6. **Homogénéité des features** : Toutes les observations ont les mêmes valeurs de features

6.2 Algorithme de Vérification

Algorithm 6 Vérification des Critères d'Arrêt

```
procedure StoppingCriterion(t, params)
    if depth(t) ≥ params.max_depth then
        return true
    end if
    if n_t < params.min_samples_split then
        return true
    end if
    if I(t) = 0 then                                   ▷ Nœud pur
        return true
    end if
    if AllSamplesSameFeatureValues(t) then
        return true
    end if
    return false
end procedure
```

7 Prédiction

7.1 Algorithme de Prédiction

Algorithm 7 Prédiction sur Nouvelles Données

```
procedure PREDICT(T, x)
    t ← T.root
    while t is not a leaf do
        (j, s) ← GetSplitInfo(t)
        if x_j est continue then
            if x_j ≤ s then
                t ← t.left
            else
                t ← t.right
            end if
        else                                          ▷ Catégorielle
            if x_j ∈ s then                           ▷ s est un ensemble de catégories
                t ← t.left
            else
                t ← t.right
            end if
        end if
    end while
    return t.prediction
end procedure
procedure BATCHPREDICT(T, X)
    predictions ← []
    for x ∈ X do
        ŷ ← Predict(T, x)
        predictions.append(ŷ)
    end for
    return predictions
end procedure
```

8 Analyse Détaillée de l'Implémentation GitHub

8.1 Architecture du Code

Le repository `Adamkhald/breiman-cart` présente une implémentation modulaire de qualité production qui respecte fidèlement la méthodologie originale de 1984.

8.1.1 Structure des Fichiers

Fichier/Dossier	Description

breiman_cart.py	API unifiée principale (v0.2.0, 2000+ lignes). Expose BRCRegression, BRCClassification et BRCInference pour usage simplifié.
breiman_cart/__init__.py	Initialisation du package. Gère les imports et expose l'API publique.
breiman_cart/node.py	Structure de données pour les nœuds avec support natif des catégories. Implémente la classe Node avec attributs pour divisions, prédictions et métadonnées.
breiman_cart/splitter.py	Logique de division incluant calcul de Gini, MSE et génération de sous-ensembles pour variables catégorielles.
breiman_cart/tree.py	Implémentation interne des arbres CART. Classes de base pour construction et traversée.
breiman_cart/pruning.py	Implémentation complète de l'élagage par coût-complexité avec calcul de la séquence alpha.
tests/	Suite de tests unitaires pour valider chaque composant et l'API.
docs/theory.tex	Documentation mathématique détaillée de la théorie CART.
demo.py	Script de démonstration avec exemples complets d'utilisation.
quickstart.py	Guide de démarrage rapide montrant l'utilisation de base.
test_new_api.py	Tests spécifiques pour la nouvelle API (v0.2.0).
fix_syntax.py	Utilitaire pour validation et correction syntaxique.
requirements.txt	Dépendances : numpy, pandas, pickle, json, logging, warnings, itertools.
pyproject.toml	Configuration du package Python (build, métadonnées).
MANIFEST.in	Fichiers à inclure dans la distribution.
LICENSE	Licence MIT.
ReadMe.md	Documentation principale du projet.

8.2 Nouvelle API Unifiée (Version 0.2.0)

La version 0.2.0 introduit une refonte majeure de l'API avec trois classes principales pour une expérience utilisateur améliorée :

8.2.1 BRCRegression - Régression

Classe dédiée aux tâches de régression avec interface simplifiée :

Listing 1 – Utilisation de BRCRegression

```
from breiman_cart import BRCRegression
import pandas as pd
import numpy as np

# Donn es de r gression
X = pd.DataFrame({
```

```
    'sqft': [1200, 1800, 1500, 2200, 1900],
    'neighborhood': ['A', 'B', 'A', 'C', 'B'],
    'age_years': [10, 5, 15, 2, 7]
})
y = np.array([300000, 450000, 350000, 550000, 420000])

# Cr ation et entra nement du mod le
model = BRCRegression(
    max_depth=10,
    min_samples_split=20,
    min_samples_leaf=10
)
model.fit(X, y, categorical_features=['neighborhood'])

# Pr dictions
predictions = model.predict(X)

# Sauvegarde du mod le
model.save('regression_model.pkl')
```

8.2.2 BRCClassification - Classification

Classe optimisée pour les tâches de classification :

Listing 2 – Utilisation de BRCClassification

```
from breiman_cart import BRCClassification

# Donn es de classification
X = pd.DataFrame({
    'age': [25, 45, 35, 50, 23, 40, 55, 30],
    'city': ['NYC', 'LA', 'NYC', 'SF', 'LA', 'SF', 'NYC', 'LA'],
    'income': [50000, 80000, 60000, 95000, 45000, 75000, 100000,
       55000]
})
y = np.array([0, 1, 0, 1, 0, 1, 1, 0])

# Cr ation du classificateur
model = BRCClassification(
    max_depth=5,
    min_samples_split=2,
    criterion='gini'  # ou 'entropy'
)
model.fit(X, y, categorical_features=['city'])

# Pr dictions
predictions = model.predict(X)
probabilities = model.predict_proba(X)

#   valuation
accuracy = model.score(X, y)
print(f"Accuracy: {accuracy:.3f}")
```

8.2.3 BRCInference - Inférence avec Modèles Sauvegardés

Classe pour charger et utiliser des modèles pré-entraînés :

Listing 3 – Utilisation de BRCInference

```
from breiman_cart import BRCInference

# Chargement d'un mod le sauvegard
model = BRCInference.load('regression_model.pkl')

# Pr dictions sur nouvelles donn es
new_data = pd.DataFrame({
    'sqft': [1600, 2000],
    'neighborhood': ['A', 'B'],
    'age_years': [8, 3]
})

predictions = model.predict(new_data)
print(f"Prix␣pr dits:␣{predictions}")

# Export du mod le en diff rents formats
model.export_json('model.json')
model.export_summary('model_summary.txt')
```

8.3 Caractéristiques Principales

8.3.1 Gestion Native des Variables Catégorielles

Innovation majeure : L'implémentation traite véritablement les variables catégorielles par division de sous-ensembles (subset splitting), contrairement à scikit-learn qui utilise l'encodage one-hot.

Avantages :

- Fidélité à l'algorithme original de Breiman
- Réduction de la dimensionnalité (pas d'explosion du nombre de variables)
- Meilleure interprétabilité des divisions
- Respect de la sémantique des variables catégorielles

Listing 4 – Exemple d'utilisation avec la nouvelle API v0.2.0

```
from breiman_cart import BRCClassification
import pandas as pd
import numpy as np

# Donn es avec variable cat gorielle 'city'
X = pd.DataFrame({
    'age': [25, 45, 35, 50, 23],
    'city': ['NYC', 'LA', 'NYC', 'SF', 'LA']
})
y = np.array([0, 1, 0, 1, 0])

# Construction du classificateur avec nouvelle API
model = BRCClassification(
```

```
    max_depth=5,
    min_samples_split=2,
    criterion='gini'
)

# Entra nement (sp cifier cat gorielles dans fit())
model.fit(X, y, categorical_features=['city'])

# Pr dictions
predictions = model.predict(X)
print("Pr dictions:", predictions)

# Probabilit s
probabilities = model.predict_proba(X)
print("Probabilit s:", probabilities)

# Sauvegarde du mod le
model.save('classifier.pkl')
```

Avantages de la nouvelle API :
- Interface plus intuitive et cohérente
- Noms de classes explicites (BRCClassification vs CARTClassifier)
- Gestion simplifiée des modèles sauvegardés avec BRCInference
- Support natif de sauvegarde/chargement (pickle et JSON)
- Meilleure séparation des responsabilités
- Documentation inline améliorée

8.3.2 Élagage par Coût-Complexité

L'implémentation complète de l'élagage suit rigoureusement la méthodologie originale de Breiman avec :
- Calcul de la séquence complète des valeurs α
- Génération de tous les sous-arbres optimaux
- Validation sur ensemble de validation séparé
- Sélection du meilleur arbre selon la performance de validation

Listing 5 – Élagage de l'arbre

```
# Division des donn es
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42
)

# Entra nement sur ensemble d'apprentissage
tree.fit(X_train, y_train)

#  lagage  avec ensemble de validation
tree.prune(X_val, y_val)

```

```
# Évaluation sur ensemble de test
accuracy = tree.score(X_test, y_test)
```

8.3.3 Type Safety

Le code utilise des annotations de type Python (type hints) pour :
— Améliorer la lisibilité
— Faciliter la maintenance
— Permettre la détection d'erreurs statique
— Documenter implicitement les interfaces

8.4 Exemple de Régression

Listing 6 – Utilisation pour régression avec nouvelle API

```
from breiman_cart import BRCRegression

# Données pour prédiction de prix immobilier
X = pd.DataFrame({
    'sqft': [1200, 1800, 1500, 2200, 1900],
    'neighborhood': ['A', 'B', 'A', 'C', 'B'],
    'age_years': [10, 5, 15, 2, 7]
})
y = np.array([300000, 450000, 350000, 550000, 420000])

# Construction du régresseur
model = BRCRegression(
    max_depth=4,
    min_samples_split=2,
    min_impurity_decrease=0.01
)
model.fit(X, y, categorical_features=['neighborhood'])

# Prédictions
predictions = model.predict(X)
print("Prix prédits:", predictions)

# Métriques
mse = np.mean((y - predictions)**2)
rmse = np.sqrt(mse)
r2 = model.score(X, y)
print(f"RMSE: {rmse:.2f}")
print(f"R²  : {r2:.3f}")

# Export et sauvegarde
model.save('house_price_model.pkl')
model.export_json('model_config.json')
```

8.5 Gestion des Modèles et Persistance

La version 0.2.0 introduit des fonctionnalités robustes de sauvegarde et chargement des modèles :

8.5.1 Sauvegarde et Chargement avec Pickle

Listing 7 – Persistance des modèles

```
from breiman_cart import BRCClassification, BRCInference

# Entra nement et sauvegarde
model = BRCClassification(max_depth=5)
model.fit(X_train, y_train, categorical_features=['city'])
model.save('my_model.pkl')

# Chargement et utilisation
loaded_model = BRCInference.load('my_model.pkl')
predictions = loaded_model.predict(X_test)
```

8.5.2 Export en Format JSON

Listing 8 – Export JSON pour interopérabilité

```
# Export de la configuration compl te
model.export_json('model_config.json')

# Le fichier JSON contient:
# - Structure de l'arbre
# - Hyperparam tres
# - Statistiques d'entra nement
# - Variables cat gorielles
# - M tadonn es du mod le
```

8.5.3 Génération de Résumés

Listing 9 – Génération de résumé lisible

```
# G n ration d'un r sum  textuel
model.export_summary('model_summary.txt')

# Contient:
# - Nombre de n uds et feuilles
# - Profondeur de l'arbre
# - Variables importantes
# - Performance du mod le
# - R gles de d cision principales
```

8.6 Logging et Débogage

Le package intègre un système de logging complet pour faciliter le débogage et la surveillance :

Listing 10 – Configuration du logging

```
import logging

# Configuration du niveau de log
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)

# Le mod le log automatiquement:
# -  tapes  de construction de l'arbre
# - Divisions choisies    chaque n ud
# - M triques d'impuret
# - Warnings pour donn es probl matiques
# - Erreurs et exceptions

model = BRCClassification(max_depth=5, verbose=True)
model.fit(X, y)  # Affiche des logs d taill s
```

8.7 Différences avec Scikit-Learn

Aspect	breiman-cart (v0.2.0)	scikit-learn
API Principale	BRCClassification, BRCRegression, BRCInference	DecisionTreeClassifier, DecisionTreeRegressor
Variables catégorielles	Division par sous-ensembles native	Encodage one-hot requis
Élagage	Séquence complète de coût-complexité	Élagage simplifié via ccp_alpha
Sauvegarde	Méthodes save()/load() intégrées + JSON	Nécessite joblib/pickle externe
Logging	Système de logging intégré	Logging minimal
Transparence	Code Python pur, éducatif	Code optimisé en Cython
Dépendances	NumPy/Pandas + stdlib	Scipy, joblib, etc.
Performance	Plus lent mais pédagogique	Très rapide
Objectif	Éducation et recherche	Production
Fidélité Breiman	Implémentation fidèle 1984	Optimisations modernes
Export	JSON, pickle, résumés	pickle/joblib

8.8 Installation et Utilisation

8.8.1 Installation

Listing 11 – Installation du package

```
# Installation des d pendances
pip install numpy pandas

# Clonage du repository
git clone https://github.com/Adamkhald/breiman-cart.git
cd breiman-cart

# Installation en mode d veloppement (optionnel)
pip install -e .
```

8.8.2 Tests

Listing 12 – Exécution des tests

```
# Installation de pytest
pip install pytest

# Ex cution de tous les tests
python -m pytest tests/

# Tests avec couverture
pip install pytest-cov
python -m pytest tests/ --cov=src
```

8.9 Complexité Temporelle

8.9.1 Construction de l'Arbre

— **Meilleur cas** : $O(np \log n)$ (arbre équilibré)
— **Pire cas** : $O(n^2 p)$ (arbre dégénéré linéaire)
— **Cas moyen** : $O(np \log n)$

Détail par niveau :
— Nombre de niveaux : $O(\log n)$ (équilibré) à $O(n)$ (dégénéré)
— Par niveau : $O(np)$ pour évaluer toutes les divisions possibles
— Tri initial des variables continues : $O(np \log n)$

Variables catégorielles :
— Cas général : $O(2^m)$ divisions à évaluer
— Classification binaire optimisée : $O(m \log m)$

8.9.2 Élagage

— Calcul de la séquence α : $O(|\text{nodes}|^2)$
— Validation de chaque arbre dans la séquence : $O(|\alpha_{\text{seq}}| \cdot n_{\text{val}} \cdot \text{depth})$
— Total : $O(|\text{nodes}|^2 + |\text{nodes}| \cdot n_{\text{val}} \cdot \log n)$

8.9.3 Prédiction

— Une prédiction : $O(\text{depth})$ où $\text{depth} \leq \log n$ (équilibré)
— Batch de m prédictions : $O(m \cdot \text{depth})$

8.10 Complexité Spatiale

— Stockage de l'arbre : $O(|\text{nodes}|) = O(2^{\text{depth}} - 1)$
— Données triées (si pré-calcul) : $O(np)$
— Indices pour chaque nœud : $O(n \cdot \text{depth})$
— Total : $O(np + n \cdot \text{depth} + 2^{\text{depth}})$

8.11 Optimisations Possibles

1. **Tri pré-calculé** : Trier chaque variable une seule fois au début
2. **Statistiques incrémentales** : Mise à jour efficace des moyennes et variances
3. **Parallélisation** : Évaluation parallèle des variables
4. **Early stopping** : Arrêt anticipé si gain insuffisant

9 Applications et Cas d'Usage

9.1 Domaines d'Application

9.1.1 Médecine et Santé

Exemple : Prédiction du risque cardiovasculaire
— **Variables** : âge, pression artérielle, cholestérol, IMC, antécédents familiaux, habitudes (tabac, exercice)
— **Avantage CART** : Règles de décision interprétables pour les cliniciens
— **Exemple de règle** : "Si âge ¿ 55 ET pression ¿ 140 ET cholestérol ¿ 240 ALORS risque élevé"

Autres applications médicales :
— Diagnostic de maladies
— Stratification des patients
— Prédiction de réponse aux traitements
— Identification de facteurs de risque

9.1.2 Finance

Exemple : Évaluation du risque de crédit
— **Variables** : revenu, historique de crédit, ratio d'endettement, durée emploi, historique bancaire
— **Avantage CART** : Règles transparentes pour conformité réglementaire (explicabilité requise)
— **Résultat** : Score de crédit binaire ou probabilité de défaut

Autres applications financières :
— Détection de fraude
— Prédiction de churn client
— Évaluation d'investissements
— Pricing d'assurances

9.1.3 Marketing et E-commerce

Exemple : Segmentation de clientèle
— **Variables** : démographiques (âge, revenu), comportementales (fréquence achat, panier moyen), géographiques
— **Avantage CART** : Segments actionnables et interprétables
— **Utilisation** : Campagnes marketing ciblées, recommandations personnalisées

10 Avantages et Limitations

10.1 Avantages de CART

1. **Interprétabilité exceptionnelle**
   — Structure visuelle claire (arbre)
   — Règles de décision compréhensibles par des non-experts
   — Facilite l'adhésion des parties prenantes
   — Important pour domaines réglementés (finance, santé)
2. **Pas d'hypothèses distributionnelles**
   — Méthode non-paramétrique
   — Pas d'hypothèse sur la distribution des données
   — Robuste aux violations d'hypothèses
3. **Gestion native des types de données**
   — Variables continues sans transformation
   — Variables catégorielles sans encodage
   — Variables ordinales respectées
   — Mix de types dans le même modèle
4. **Détection automatique des interactions**
   — Captures automatiques des interactions complexes
   — Pas besoin de spécification manuelle
   — Hiérarchie naturelle des interactions
5. **Robustesse aux valeurs aberrantes**
   — Divisions basées sur les rangs/ordres
   — Peu sensible aux valeurs extrêmes
   — Pas de normalisation nécessaire
6. **Sélection automatique de variables**
   — Seules les variables informatives utilisées
   — Gestion naturelle de la haute dimensionnalité
   — Feature importance intrinsèque
7. **Gestion des valeurs manquantes**
   — Méthodes de surrogate splits
   — Pas d'imputation nécessaire
   — Utilisation optimale des données disponibles

10.2 Limitations de CART

1. **Instabilité élevée**
   — Petites variations $\Rightarrow$ arbres très différents

   - Problème de généralisation
   - Solution : ensembles d'arbres (Random Forests, Boosting)
2. **Sur-apprentissage facile**
   - Arbres profonds mémorisent les données
   - Forte variance du modèle
   - Solution : élagage rigoureux, validation croisée
3. **Biais de sélection de variables**
   - Favorise les variables avec plus de points de division
   - Variables continues avantagées vs catégorielles
   - Impact sur l'importance des variables
4. **Frontières de décision rectangulaires**
   - Limitation pour relations obliques
   - Approximation grossière de frontières lisses
   - Nécessite beaucoup de divisions pour frontières diagonales
5. **Difficulté d'extrapolation**
   - Prédictions constantes dans chaque région
   - Pas d'extrapolation au-delà des données d'entraînement
   - Problématique pour séries temporelles
6. **Sensibilité au déséquilibre des classes**
   - Performance dégradée sur classes minoritaires
   - Solution : pondération des classes, stratification

$((\mathrm{y}_test == 0)(y_pred_test == 1))[0]print(f"defauxpositifs : len(fp_indices)")iflen(fp_indices) > 0 : print("Exemplesdefauxpositifs : ")print(X_test.iloc[fp_indices[: 5]])$

Faux négatifs $\mathrm{fn}_indices = np.where((y_test == 1)(y_pred_test == 0))[0]print(f"defauxnégatifs : len(fn_indices)")iflen(fn_indices) > 0 : print("Exemplesdefauxnégatifs : ")print(X_test.iloc[fn_indice 5]])$

11 Conclusion

11.1 Contributions de CART

L'algorithme CART de Breiman et al. (1984) a apporté des contributions majeures et durables à l'apprentissage automatique :

1. **Méthodologie unifiée** : Cadre cohérent pour classification et régression
2. **Fondements théoriques solides** : Base mathématique rigoureuse avec preuves de consistance
3. **Praticité exceptionnelle** : Implémentation efficace et interprétable
4. **Influence durable** : Fondation pour Random Forests, Gradient Boosting, XGBoost
5. **Adoption industrielle** : Utilisation généralisée dans de nombreux domaines

11.2 Apports du Repository GitHub

L'implémentation `Adamkhald/breiman-cart` se distingue par :

— **Fidélité historique** : Respect scrupuleux de la méthodologie de 1984

— **Gestion native des catégories** : Implémentation correcte du subset splitting
— **Élagage complet** : Séquence alpha complète selon Breiman
— **Code pédagogique** : Clarté et lisibilité pour l'apprentissage
— **Minimalisme** : Dépendances réduites (NumPy/Pandas uniquement)

11.3 Perspectives Futures

Les directions de recherche et développement actuelles incluent :
— **Deep Learning et arbres** :
  — Neural Decision Trees
  — Soft Decision Trees différentiables
  — Tree-structured neural networks
— **Arbres causaux** :
  — Inférence causale via structures arborescentes
  — Causal CART pour hétérogénéité d'effets
  — Policy learning trees
— **Équité et explicabilité** :
  — Garanties formelles d'équité
  — Fair CART avec contraintes
  — Explications contrefactuelles
— **Scalabilité** :
  — Algorithmes distribués (MapReduce, Spark)
  — Approximations pour très grandes données
  — Online learning pour streaming

11.4 Recommandations Pratiques

Pour une utilisation efficace de CART en production :

1. **Toujours élaguer** : Utiliser un ensemble de validation pour l'élagage
2. **Validation rigoureuse** : Validation croisée systématique
3. **Ensembles pour production** : Random Forests ou Boosting pour robustesse
4. **Monitoring de stabilité** : Vérifier sur échantillons bootstrap
5. **Interprétation prudente** : Distinguer corrélation et causalité
6. **Feature engineering** : Malgré l'automatisme, les transformations aident
7. **Déséquilibre des classes** : Utiliser class_weight ou stratification
8. **Valeurs manquantes** : Implementer surrogate splits si nécessaire

11.5 Impact et Héritage

CART reste aujourd'hui :
— Un des algorithmes les plus utilisés en pratique
— La base de nombreux algorithmes modernes performants
— Un outil d'exploration de données incontournable
— Un standard d'interprétabilité en ML
— Une méthode d'enseignement fondamentale

Remerciements

Ce rapport a été élaboré en analysant l'implémentation open-source disponible sur le repository GitHub `Adamkhald/breiman-cart` et les fondements théoriques de l'article original de Breiman et al. (1984). Nous remercions les contributeurs du repository pour leur travail rigoureux et pédagogique.

Références

[1] Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA.

[2] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2), 123–140.

[3] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.

[4] Breiman, L. (2001). Statistical modeling : The two cultures. *Statistical Science*, 16(3), 199–215.

[5] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning : Data Mining, Inference, and Prediction* (2nd ed.). Springer.

[6] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*. Springer.

[7] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81–106.

[8] Quinlan, J. R. (1993). *C4.5 : Programs for Machine Learning*. Morgan Kaufmann.

[9] Therneau, T. M., & Atkinson, E. J. (2000). *An Introduction to Recursive Partitioning Using the RPART Routines*. Technical Report 61, Mayo Clinic.

[10] Loh, W. Y. (2011). Classification and regression trees. *Wiley Interdisciplinary Reviews : Data Mining and Knowledge Discovery*, 1(1), 14–23.

[11] Strobl, C., Boulesteix, A. L., Zeileis, A., & Hothorn, T. (2007). Bias in random forest variable importance measures : Illustrations, sources and a solution. *BMC Bioinformatics*, 8(1), 25.

[12] Athey, S., & Imbens, G. (2016). Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences*, 113(27), 7353–7360.

[13] Repository GitHub : `https://github.com/Adamkhald/breiman-cart`

A Glossaire des Notations

Notation	Description
$\mathcal{D}$	Ensemble de données d'apprentissage
n	Nombre total d'observations
p	Nombre de variables explicatives (features)
$\mathbf{x}_i$	Vecteur de variables pour l'observation i
y_i	Variable réponse pour l'observation i
T	Arbre de décision complet
$\mathcal{N}(T)$	Ensemble des nœuds internes de l'arbre
$\widetilde{T}$	Ensemble des nœuds terminaux (feuilles)

$\lvert\widetilde{T}\rvert$	Nombre de feuilles (complexité de l'arbre)
R_j	Région correspondant à la feuille j
n_t	Nombre d'observations dans le nœud t
$I(t)$	Impureté du nœud t
α	Paramètre de coût-complexité pour l'élagage
$R_\alpha(T)$	Critère de coût-complexité
R_L, R_R	Régions gauche et droite après division
n_L, n_R	Nombre d'observations dans régions gauche/droite
$p_k(t)$	Proportion de classe k dans le nœud t
$\hat{y}_j$	Prédiction pour la région j
K	Nombre de classes (classification)
m	Nombre de catégories (variable catégorielle)

B Formules Récapitulatives

B.1 Critères d'Impureté

Gini (Classification) :

$$I_{\text{Gini}}(t) = 1 - \sum_{k=1}^{K} p_k^2(t) \tag{18}$$

Entropie (Classification) :

$$I_{\text{Entropy}}(t) = -\sum_{k=1}^{K} p_k(t) \log_2 p_k(t) \tag{19}$$

Variance (Régression) :

$$I_{\text{Var}}(t) = \frac{1}{n_t} \sum_{i \in t} (y_i - \bar{y}_t)^2 \tag{20}$$

B.2 Gain de Division

$$\Delta I(s,t) = I(t) - \frac{n_L}{n_t} I(t_L) - \frac{n_R}{n_t} I(t_R) \tag{21}$$

B.3 Coût-Complexité

$$R_\alpha(T) = R(T) + \alpha |\widetilde{T}| \tag{22}$$

B.4 Alpha Critique

$$\alpha_t = \frac{R_t - R_{T_t}}{|\widetilde{T}_t| - 1} \tag{23}$$

B.5 Décomposition Biais-Variance

$$\mathbb{E}[(y - \hat{f}(\mathbf{x}))^2] = \mathrm{Var}(\hat{f}(\mathbf{x})) + [\mathrm{Bias}(\hat{f}(\mathbf{x}))]^2 + \sigma^2 \tag{24}$$