UNIVERSITÉ DE MONTPELLIER

Les forêts aléatoires : Mondrian Forests

HMMA 308 : Apprentissage Statistique

Walid KANDOUCI

Table des matières

1 Introduction

Les forêts aléatoire reste jusqu'à aujourd'hui l'un des outils de machine learning les plus populaire vue la précision la robustesse qu'elles nous offres sur de vrai jeux de données, nous allons voir dans ce projet une nouvelle classe des forêts aléatoires : **'les Forêts de Mondrian"** (MF) en raison de l'arborescence sous-jacente de chaque classifieur de l'ensemble est un **processus de Mondrian**. Nous allons voir dans ce projet les propriétés de ces forêts aléatoires de Mondrian, ainsi que quelque applications à l'aide de Python.

2 Approche

Soit $N : (x_1, y_1), \ldots, (x_N, y_N) \in \mathbb{R}^D \times \mathcal{Y}$ nos données d'apprentissage (train), l'objectif ici est de prédire $y \in \mathcal{Y}$ pour les points $x \in \mathbb{R}^D$. On va se focaliser sur la classification multi-classes ou $\mathcal{Y} := \{1, \ldots, K\}$, cependant, il est possible d'étendre la méthodologie à d'autres tâches d'apprentissage supervisé telles que la régression. Soit $X_{1:n} := (x_1, \ldots, x_n), Y_{1:n} := (y_1, \ldots, y_n)$, et $\mathcal{D}_{1,n} := (X_{1:n}, Y_{1:n})$.

Les Mondrian forests sont construits comme les forets aléatoires : Soit $\mathcal{D}_{1:N}$ nos données d'apprentissage, on tire un échantillon $T_1, \ldots, T_M$ indépendants des arbres Mondrian (que l'on va décrire dans la section 2). La prédiction fait par chaque arbre de Mondrian T_m est une distribution $p_{T_m}(y \mid x, \mathcal{D}_{1:N})$ de y pour x. La prédiction fait par les forets de Mondrian est $\frac{1}{M}\sum_{m=1}^{M} p_{T_m}(y \mid x, \mathcal{D}_{1:N})$ de l'arbre de prédiction. Quand $M \to \infty$, cette moyenne vers $\mathbb{E}_{T\sim \mathrm{MT}(\lambda, \mathcal{D}_{1:N})}[p_T(y \mid x, \mathcal{D}_{1:N})]$ où $MT(\lambda, \mathcal{D}_{1:N})$ est la distribution de l'arbre de Mondrian. Comme la limite de l'espérance ne dépends pas de M, on ne s'attendrait pas à voir un sur ajustement quand M augmente.

Dans le cadre de l'apprentissage en ligne, les exemples de formation sont présentés l'un après l'autre dans une séquences d'essais. les forêts de Mondrian excelle dans ce cadre la : à l'itération $N+1$, chaque arbre de Mondrian $T \sim \mathrm{MT}(\lambda, \mathcal{D}_{1:N})$ est mis à jour pour incorporer l'exemple labellisé suivant : (x_{N+1}, y_{N+1}) par échantillonnage d'un arbre étendu $T'\prime$ de $\mathrm{MTx}(\lambda, T, \mathcal{D}_{N+1})$. En utilisant les propriétés du processus de Mondrian, on peut choisir la probabilité de MTx avec $T' = T$ sur $\mathcal{D}_{1:N}$ et T' est répartit en fonction de $\mathrm{MT}(\lambda, \mathcal{D}_{1:N+1})$:

$$
\begin{aligned}
T &\sim \mathrm{MT}(\lambda, \mathcal{D}_{1:N}) \\
T' \mid T, \mathcal{D}_{1:N+1} &\sim \mathrm{MTx}(\lambda, T, \mathcal{D}_{N+1}) \\
T' &\sim \mathrm{MT}(, \mathcal{D}_{1:N+1})
\end{aligned}
$$

Par conséquent, la distribution des arbres de Mondrian entraînés sur un ensemble de données de manière incrémentée est identique à celui des arbres Mondrian entraînés sur le même jeu de données de manière groupées , indépendant de l'ordre dans lequel les points de données ont été observés. À notre connaissance, aucun des forêts aléatoires en ligne existantes ont cette propriété.

De plus, nous peut tirer un échantillon de $\mathrm{MTx}(\lambda, T, \mathcal{D}_{N+1})$, la complexité évolue avec la profondeur de l'arbre, qui est typiquement logarithmique en N de manière efficace.

En traitant le paramètre en ligne comme une séquence de groupes de plus en plus importants, le problème est généralement un problème de calcule, les forets de Mondrian nous permettent de réaliser efficacement cette approche.

Nous allons définir avec plus de détaille dans la suite :

- La distribution des arbres de Mondrian $\mathrm{MT}(\lambda, \mathcal{D}_{1:N})$
- l'indexe de distribution $p_T(y \mid x, \mathcal{D}_{1:N})$
- La distribution mise a jour $\mathrm{MTx}(\lambda, T, \mathcal{D}_{N+1})$

3 Arbres de Mondrian

Nous allons poser l'hypothèse que ici l'arbre de décision dans $\mathbb{R}^D$ sera hiérarchique, partition binaire de $\mathbb{R}^D$ et une règle pour prédire le label des points de test à partir des données d'apprentissage. La structure de l'arbre de décision est fini, enraciné, strictement binaire T.

Un ensemble fini de noeuds est défini tel que :

- chaque noeud j a exactement un nœud parent, sauf pour un nœud racine distingué ϵ qui n'a pas de parents
- chaque noeud j est le parent d'exactement zéro ou deux noeuds enfants : (le noeud de gauche "$\mathrm{left}(j)$" et le noeud de droite "$\mathrm{right}(j)$")

On note les feuilles de T (les noeuds sans enfants) ("$leaves(T)$") chaque noeud de l'arbre $j \in T$ est associé avec un bloc $B_j \subset \mathbb{R}^D$ de l'ensemble de départ comme suit : Aux racines, on a $B_\epsilon = \mathbb{R}^D$, tandis que chaque nœud interne $j \in T \backslash \text{leaves}(T)$ avec deux enfants qui représentent une division du bloc des parent en deux moitiés, avec $\delta_j \in \{1, \ldots, D\}$ qui indique la dimension du split et ξ_j qui représente la localisation du split, en particulier :

$$B_{\text{left }(j)} := \left\{x \in B_j : x_{\delta_j} \leq \xi_j\right\}$$
$$B_{\text{right }(j)} :=| \left\{x \in B_j : x_{\delta_j} > \xi_j\right\}$$

On appel (T, δ, ξ) un arbre de décision. On note que les blocs associaient avec les feuilles de l'arbre forme une partition de $\mathbb{R}^D$, on peut écrire $B_j = (\ell_{j1}, u_{j1}] \times \ldots \times (\ell_{jD}, u_{jD}]$ ou ℓ_{jd} et u_{jd} représentent les bornes inférieurs et supérieurs du bloc rectangulaire B_j le long de la dimension d. On pose $\ell_j = \{\ell_{j1}, \ell_{j2}, \ldots, \ell_{jD}\}$ et $\mathbf{u}_j = \{u_{j1}, u_{j2}, \ldots, u_{jD}\}$.

On note aussi :

- $\text{parent}(j)$: le parent du noeud j
- $N(j)$: l'indice de nos données d'apprentissage au point j ($N(j) = \{n \in \{1, \ldots, N\} : \boldsymbol{x}_n \in B_j\}$)
- $\mathcal{D}_{N(j)} = \left\{\boldsymbol{X}_{N(j)}, Y_{N(j)}\right\}$: les caractéristiques et les étiquettes des points de données d'apprentissage au noeud j
- ℓ^x_{jd} et u^x_{jd} : les bornes inférieurs et supérieurs de nos données d'apprentissage au noeud j le long de la dimension d
- $B^x_j = \left(\ell^x_{j1}, u^x_{j1}\right] \times \ldots \times \left(\ell^x_{jD}, u^x_{jD}\right] \subseteq B_j$: le plus petit rectangle qui entoure les données d'apprentissage au noeud j

3.1 Distribution des processus Mondrian sur les arbres de décisions

Les processus de Mondrian sont des familles $\{\mathcal{M}_t : t \in [0, \infty)\}$ de partitions binaire hiérarchique aléatoires de $\mathbb{R}^D$ tel que $\mathcal{M}_t$ est un raffinement de $\mathcal{M}_S$ avec $t > S$. Les processus de Mondrian sont des candidat pour la structure de partition d'arbres de décision aléatoires mais sont, en général, des structures infinies que nous ne pouvons pas représenter tout à la fois, et c'est pour cela que l'on va se focaliser essentiellement que de la partition sur un ensemble fini de données observées, On introduit des arbres de Mondrian, qui sont des restrictions des processus de Mondrian à un ensemble fini de points.

Un arbre de Mondrian T peut être représenté par (T, δ, ξ, τ) ou (T, δ, ξ) est un arbre de décision et $\tau = \{\tau_j\}_{j \in T}$ associe un temps de division (split) $\tau_j \geq 0$ avec chaque noeud j. le temps de division augmente avec la profondeur ($\tau_j > \tau_{parent(j)}$) (on note $\tau_{parent(\epsilon)} = 0$)

Sachant que le paramètre λ représente la durée de vie et $\mathcal{D}_{1:n}$ nos données d'apprentissage, l'algorithme suivant décrit le processus génératif pour l'échantillonnage des arbres de Mondrian $\text{MT}(\lambda, \mathcal{D}_{1:n})$:

Algorithm 1 $\text{SampleMondrianTree}(\lambda, \mathcal{D}_{1:n})$

Initialisation: $T = \emptyset, \text{leaves}(T) = \emptyset, \delta = \emptyset, \xi = \emptyset, \tau = \emptyset, N(\epsilon) = \{1, 2, \ldots, n\}$
$\text{SampleMondrianBlock}(\epsilon, \mathcal{D}_{N(\epsilon)}, \lambda)$

— Le processus commence avec le noeud de départ ϵ et se répète dans l'arborescence.

Algorithm 2 $\text{SampleMondrianBlock}(j, \mathcal{D}_{N_{(j)}}, \lambda)$

Add j to T
For all d, set $\ell^x_{jd} = \min(\boldsymbol{X}_{N(j),d}), u^x_{jd} = \max(\boldsymbol{X}_{N(j),d})$
Sample E from exponential distribution with rate $\sum_d (u^x_{jd} - \ell^x_{jd})$
if $\tau_{\text{parent }(j)} + E < \lambda$ **then**
  Set $\tau_j = \tau_{\text{parent }(j)} + E$
  Sample split dimension δ_j, choosing d with probability proportional to $u^x_{jd} - \ell^x_{jd}$
  Sample split location ξ_j uniformly from interval $[\ell^x_{j\delta_j}, u^x_{j\delta_j}]$
  Set $N(\text{left}(j)) = \left\{n \in N(j) : \boldsymbol{X}_{n,\delta_j} \leq \xi_j\right\}$ and $N(\text{ right }(j)) = \left\{n \in| \ N(j) : \boldsymbol{X}_{n,\delta_j} > \xi_j\right\}$
  SampleMondrianBlock $\left(\text{left}(j), \mathcal{D}_{N(\text{left}(j))}, \lambda\right)$
  SampleMondrianBlock $\left(\text{ right }(j), \mathcal{D}_{N(\text{ right }(j))}, \lambda\right)$
else Set $\tau_j = \lambda$ and add j to leaves (T)

— On calcule d'abord le ℓ_ϵ^x et $\mathbf{u}_\epsilon^x$ (les bornes de B_ϵ^x), On prend un échantillon E qui suit une loi exponentielle de paramètre $\sum_d \left(u_{jd}^x - \ell_{jd}^x\right)$. Puisque $\tau_{parent(\epsilon)} = 0$, on a $E + \tau_{parent(\epsilon)} = E$. Si $E \geq \lambda$, le temps de divisions n'est pas dans la durée de vie λ ; c'est pour cela que ϵ sera la feuille du noeud et le processus s'arrêtera. Sinon ϵ est un noeud interne et on prend un échantillon du split $(\delta_\epsilon, \xi_\epsilon)$ d'une loi uniforme dans B_ϵ^x et le processus se répète ensuite le long des enfants gauche et droite.

Les arbres de Mondrian diffèrent des arbres de décision comme suit :

- les divisions sont échantillonnées indépendamment des $Y_{N(j)}$
- chaque noeud j est associé avec un temps de division τ_j
- λ contrôle le nombre totale des divisions
- la division représenté par un noeud interne j ne tient que dans B_j^x et non pas B_j

Considérons la famille de distribution $\mathrm{MT}(\lambda, F)$ où F s'étend sur tous les ensembles finis possibles de points de données. En raison du fait que ces distributions sont dérivées de celle d'un processus de Mondrian dans $\mathbb{R}^D$ restreint à F, $\mathrm{MT}(\lambda, F)$ sera projeté . Intuitivement, si on prend un échantillon de l'arbre Mondrian T de $\mathrm{MT}(\lambda, F)$ et si on limite T à un sous ensemble $F' \subseteq F$ alors l'arbre restreint T' a une distribution $\mathrm{MT}(\lambda, F')$. Plus important, la projectivité nous donne un moyen d'étendre un arbre de Mondrian sur un ensemble de données $\mathcal{D}_{1:N}$ à un ensemble de données plus grand $\mathcal{D}_{1:N+1}$.

On utilise cette propriété pour faire pousser progressivement un arbre Mondrian : On initialise l'arbre the Mondrian sur nos données observé, en observent de nouveaux données $\mathcal{D}_{N+1}$, on étant l'arbre de Mondrian en échantillonnant à partir de la distribution conditionnelle d'un arbre Mondrian sur $\mathcal{D}_{\infty:\mathcal{N}+\infty}$ avec comme restriction de $\mathcal{D}_{1:N}$: $\mathrm{MTx}\left(\lambda, T, \mathcal{D}_{N+1}\right)$.

Ainsi, un processus Mondrian de $\mathbb{R}^D$ n'est représenté que là où nous avons observé des données.

4 Label distribution :Model, Prior hiérarchique et posterior prédictif

Pour un arbre $T = (T, \delta, \xi, \tau)$, $\mathcal{D}_{1:N}$ un jeu de données et point test x. $\mathrm{leaf}(x)$ l'unique feuille du noeud j ($j \in \mathrm{leaves}(T)$) tel que $x \in B_j$.

Intuitivement, nous voulons la distribution prédictive des labels en x être une version lissée de la distribution empirique des labels pour les points dans $B_{leaf(x)}$ et en $B_{j'}$ pour le noeud j'. On arrive a ce lissage grâce à l'approche hiérarchique bayesienne : chaque noeud est associé avec la distribution d'un label choisit d'après la quel des distribution du label d'un noeud est similaire a selle de son parent. Le predictive $p_T\left(y \mid x, \mathcal{D}_{1:N}\right)$ est obtenue avec margninalisation.

On assume que le label de chaque bloc est indépendent du X étant donné la structure arborescente. Pour chaque $j \in T$, G_j : la distribution des labels au noeud j, et soit $G = \{G_j : j \in T\}$ l'ensemble des labels distributions à tout les noeud de l'arbre. Soit T et G les distributions prédictive du label en x est $p(y \mid x, T, \mathcal{G}) = G_{\mathrm{leaf}(x)}$ (le label distribution au noeud $\mathrm{leaf}(x)$).

On modélise G_j pour $j \in T$ comme une hiérarchie d'un processus normalisé stable (NSP). Le prior NSP est une distribution sur un des distributions (cas spécial du prior du processus de Pitman-Yor (PYP) ; le paramètre de concentration est zero). $d \in (0, 1)$ un paramètre de que l'on appel "discount" contrôle les variation autour la base de distribution, si $G_j \sim \mathrm{NSP}(d, H)$ alors : $\mathbb{E}\left[G_{jk}\right] = H_k$ et $\mathrm{Var}\left[G_{jk}\right] = (1-d)H_k\left(1 - H_k\right)$. On utilise le prior d'un HNSP (NSP hiérarchique) sur G_j comme suit :

$$G_\epsilon \mid H \sim \mathrm{NSP}\left(d_\epsilon, H\right)$$
$$G_j \mid G_{\text{parent }(j)} \sim \mathrm{NSP}\left(d_j, G_{\text{parent }(j)}\right)$$

Soit nos données $\mathcal{D}_{1:N}$, $p_T\left(y \mid x, \mathcal{D}_{1:N}\right)$ est obtenue en integrant sur G_j :

$$p_T\left(y \mid x, \mathcal{D}_{1:N}\right) = \mathbb{E}_{\mathcal{G} \sim p_T(\mathcal{G} \mid \mathcal{D}_{1:N})}\left[G_{\mathrm{leaf}(\boldsymbol{x}), y}\right] = \bar{G}_{\mathrm{leaf}(\boldsymbol{x}), y}$$

Où le posterior :

$$p_T\left(\mathcal{G} \mid \mathcal{D}_{1:N}\right) \propto p_T(\mathcal{G}) \prod_{n=1}^{N} G_{\mathrm{leaf}(\boldsymbol{x}_n), y_n}$$

Le calcule de la moyenne du posterior $\bar{G}_{\text{leaf }(x)}$ est un cas spécial du posterior du HPYP.

5 Apprentissage en ligne et prédiction

Nous allons maintent décrir une fammille $\mathrm{MTx}\left(\lambda, T, \mathcal{D}_{N+1}\right)$ qui est utilisé pour ajouter de manière incrémentielle un point de données $\mathcal{D}_{N+1}$ à un arbre T. Cette mise a jour est basé sur l'algorithme conditionnelle de Mondrian, spécialisé à un ensemble fini de points. En général, une ou plusieurs des trois opérations suivantes peuvent être exécutées lors de l'introduction d'un nouveau point de données :

- introduction d'une nouvelle division"au-dessus" d'une division existante
- extension d'un fractionnement existant au bloc mis à jours
- fractionnement d'un nœud feuille existant en deux enfants

À notre connaissance, les arbres de décision en ligne existants n'utilisent que la troisième opération, et les deux premières opérations sont uniques aux arbres de Mondrian.

l'algorithme suivant décrit cela avec plus de détails :

Algorithm 3 EXTENDMONDRIANTREE($T, \lambda, \mathcal{D}$)

Input : Tree $T = (\mathrm{T}, \delta, \xi, \tau)$,new training instance $\mathcal{D} = (x, y)$
ExtendMondrianBlock $(T, \lambda, \epsilon, \mathcal{D})$

Algorithm 4 EXTENDMONDRIANBLOCK($T, \lambda, j, \mathcal{D}$)

Set $\mathbf{e}^{\ell} = \max\left(\ell_j^x - x, 0\right)$ and $\mathbf{e}^{u} = \max\left(x - \mathbf{u}_j^x, 0\right)$
$\mathbf{e}^{\ell} = \mathbf{e}^{u} = \mathbf{0}_D$ if $\boldsymbol{x} \in B_j^x$
Sample E from exponential distribution with rate $\sum_d \left(e_d^{\ell} + e_d^{u}\right)$
if $\tau_{\text{parent }(j)} + E < \tau_j$ then D introduce new parent for node j
Sample split dimension δ, choosing d with probability proportional to $e_d^{\ell} + e_d^{u}$
Sample split location ξ uniformly from interval $\left[u_{j,\delta}^x, x_\delta\right]$ if $x_\delta > u_{j,\delta}^x$ else $\left[x_\delta, \ell_{j,\delta}^x\right]$
Insert a new node $\tilde{j}$ just above node j in the tree, and a new leaf j'', sibling to j, where
$\delta_{\tilde{j}} = \delta, \xi_{\tilde{j}} = \xi, \tau_{\tilde{j}} = \tau_{\text{parent }(j)} + E, \ell_{\tilde{j}}^x = \min\left(\ell_j^x, \boldsymbol{x}\right), \mathbf{u}_{\tilde{j}}^x = \max\left(\mathbf{u}_j^x, \boldsymbol{x}\right)$
$j'' = \mathrm{left}(\tilde{j})$ iff $x_{\delta_{\tilde{j}}} \leq \xi_{\tilde{j}}$
SampleMondrianBlock $(j'', \mathcal{D}, \lambda)$
else Update $\ell_j^x \leftarrow \min\left(\ell_j^x, x\right), \mathbf{u}_j^x \leftarrow \max\left(\mathbf{u}_j^x, x\right)$
update extent of node j
if $j \notin$ leaves (T) then
return if j is a leaf node, else recurse down the tree
if $x_{\delta_j} \leq \xi_j$ then
child $(j) = \mathrm{left}(j)$ else child $(j) = \mathrm{right}(j)$
ExtendMondrianBlock $(T, \lambda, \mathrm{child}(j), \mathcal{D})$
recurse on child containing $\mathcal{D}$

6 Application

6.1 Partie 1 : Générateur Mondrian

On définit ici la fonction *random_mondrian* qui génère un graphe type "Mondrian" et qui nous fournit la variable "budget. Il génère le cas général (non conditionnel) :

On commence par définir certaines fonction nécessaire pour faire fonctionner notre générateur : (code complet dans l'annexe)

```
def random_mondrians(box, budget, given_mondrian=None, color=None,
                rows=1, columns=1, figsize=(15, 15)):
   if given_mondrian is None:
      given_mondrian = Mondrian(None, budget)

   if rows == 1 and columns == 1:
      fig, ax = plt.subplots(figsize=figsize)
      plot_coloured_mondrian(grow_mondrian(box, budget, given_mondrian),
                        ax, color=color, given_mondrian=given_mondrian)
   else:
      fig, ax = plt.subplots(rows, columns, figsize=figsize)

      for row in range(rows):
         for col in range(columns):
            if not given_mondrian.is_empty() and row == 0 and col ==0:
               plot_coloured_mondrian(given_mondrian,
                                 ax[row, col], color=color)
            else:
               plot_coloured_mondrian(grow_mondrian(box, budget,
                                 given_mondrian),
                                 ax[row, col], color=color,
                                 given_mondrian=given_mondrian)
```

```
random_mondrians(box, 1, rows=5, columns=5, color='true_mondrian')
```

et qui nous donne le résultat suivant :

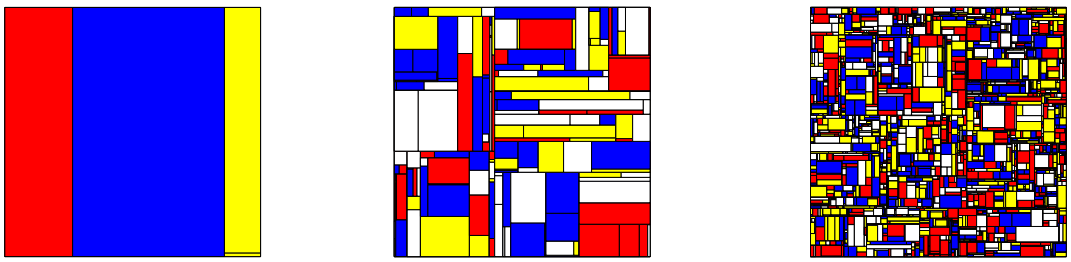


FIGURE 1 – Exemple générateur aléatoire Mondrian (budget=2,10,50)

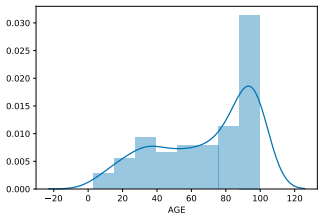
Ici on peut voir un fait intéressent a propos de ce générateur est que sur chaque coupe (division) les deux nouveaux blocs qui en résultent sont conditionnellement indépendants l'un de l'autre.

6.2 Forêt Mondrian sur nos données DIGITS

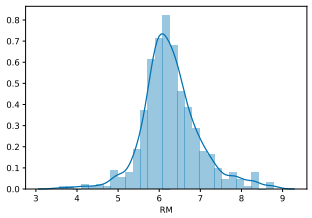
Dans cette partie, nous allons utilisé le jeu de données **"Boston Housing"**, il comprend le prix des maisons dans divers endroits de Boston. Outre le prix, l'ensemble de données fournit également des informations telles que la criminalité (CRIM), les zones d'activités non commerciales dans la ville (INDUS), l'âge des personnes qui possèdent la maison (AGE), et il existe de nombreux autres attributs.

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
0.02731	00.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
0.02729	00.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
0.03237	00.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
0.06905	00.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

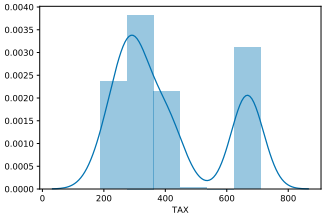
- **CRIM :** taux de criminalité par habitant par ville
- **ZN :** proportion de terrains résidentiels zonés pour les lots de plus de $25000m^2$
- **INDUS :** proportion d'acres non commerciales par ville
- **CHAS :** Variable fictive de Charles River (1 si la zone délimite la rivière, 0 sinon)
- **NOX :** concentration d'oxydes nitriques (parties par 10 millions)
- **RM :** nombre moyen de pièces par logement
- **AGE :** proportion de logements occupés par leur propriétaire construits avant 1940
- **DIS :** distances pondérées jusqu'à cinq centres d'emploi de Boston
- **RAD :** indice d'accessibilité aux autoroutes (radiales)
- **TAX :** taux d'imposition foncière de la valeur totale par tranche de 10000 $
- **PTRATIO :** ratio élèves-enseignant par ville
- **B :** $1000(Bk - 0,63)^2$ où Bk est la proportion d'afro américains par ville
- **LSTAT :** % statut inférieur de la population
- **MEDV :** Valeur médiane des logements occupés par leur propriétaire en milliers de dollars



(a) Logement occupé construit < 1940



(b) Nombre pièces



(c) Taux d'imposition

FIGURE 2 – Représentation graphique de quelques variables

A l'aide du package "$skgarden$" il nous ait possible d'importer les fonction "$MondrianForestCLassifier$", "$MondrianForestRegressor$", "$MondrianTreeClassifier$" et "$MondrianTreeRegressor$"

7 Conclusion

Nous avons introduit les forêts de Mondrian, une nouvelle classe de forêts aléatoires qui est une notion nouvelle et récente. Nous avons aussi dans ce projet créer une fonction génératrice qui nous permet de simuler et de mieux voir la méthode de fonctionnement des forêts de Mondrian.

Références

[1] Mondrian Forests : Efficient Online Random Forests
Balaji Lakshminarayanan (University College London), Daniel M. Roy (University of Cambridge),Yee Whye Teh(Department of Statistics University of Oxford)

Code Python

Mondrian Générateur

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.path as mpath
import matplotlib.lines as mlines
import matplotlib.patches as mpatches
from matplotlib.collections import PatchCollection

%matplotlib inline

def dimensions(box, size=None):
   if box is None:
      if size is not None:
         return np.zeros(size)
      else:
         raise ValueError('If box is not provided, you must specify size.')

   return np.diff(box, axis=1).flatten()

def linear_dimension(box):
   if box is None:
      return 0
   return dimensions(box).sum()

def interval_difference(outer_interval, inner_interval):
   lower_outer, upper_outer = outer_interval
   lower_inner, upper_inner = inner_interval
   return [lower_outer, lower_inner], [upper_inner, upper_outer]

def sample_interval_difference(outer_interval, inner_interval):
   intervals = interval_difference(outer_interval, inner_interval)
   dimensions = [np.diff(intervals[0])[0], np.diff(intervals[1])[0]]
   chosen_interval_index = np.random.choice(range(len(intervals)),
   p=dimensions/np.sum(dimensions))
   chosen_interval = intervals[chosen_interval_index]
   return np.random.uniform(low=chosen_interval[0],
   high=chosen_interval[1], size=1)[0], chosen_interval_index

def random_axis(dimensions):
   return np.random.choice(range(len(dimensions)),
                     p=dimensions/np.sum(dimensions))

def random_cut(box, axis):
   return np.random.uniform(low=self.box[axis][0],
                      high=self.box[axis][1],
                      size=1)[0]

def cost_next_cut(linear_dimension):
   return np.random.exponential(scale=1.0/linear_dimension, size=1)[0]

def new_cut_proposal(self):
   cost = self.cost_next_cut()
   axis = self.random_axis()
   cut_point = self.random_cut(axis)
```

```
    return cost, axis, cut_point

def cut_boxes(box, cut_axis, cut_point):
    left = box.copy()
    right = box.copy()
    low, high = box[cut_axis]

    if cut_point <= low or cut_point >= high:
        raise ValueError('Point is not in interval.')

    left[cut_axis] = [low, cut_point]
    right[cut_axis] = [cut_point, high]
    return left, right

class Mondrian(object):
    def __init__(self, box, budget):
        self.box = box
        self.budget = budget
        self.cut_point = None
        self.cut_axis = None
        self.cut_budget = None
        self.left = None
        self.right = None

    def extended_by(self, box):
        if self.box is None:
            return True
        return all((box[:, 0] <= self.box[:, 0]) & (box[:, 1] >= self.box[:, 1]))

    def contains(self, point):
        if self.box is None:
            return False
        return all(box[:,0] <= point) & (box[:,1] >= point)

    def has_cut(self):
        return self.cut_axis is not None

    def is_empty(self):
        return self.box is None

def grow_mondrian(box, budget, given_mondrian=None):
    if given_mondrian is None:
        given_mondrian = Mondrian(None, budget)

    if not given_mondrian.extended_by(box):
        raise ValueError('Incompatible boxes: given mondrian
        box must be contained in new box.')

    mondrian = Mondrian(box, budget)

    cost = cost_next_cut(linear_dimension(box) -
    linear_dimension(given_mondrian.box))

    next_budget = budget - cost

    given_mondrian_next_budget = given_mondrian.cut_budget if
    given_mondrian.has_cut() else 0
```

```
    if next_budget < given_mondrian_next_budget:
        if given_mondrian.has_cut():
            mondrian.cut_axis = given_mondrian.cut_axis
            mondrian.cut_point = given_mondrian.cut_point
            mondrian.cut_budget = given_mondrian_next_budget

            left, right = cut_boxes(box, mondrian.cut_axis,
            mondrian.cut_point)

            mondrian.left = grow_mondrian(left,
            given_mondrian_next_budget, given_mondrian.left)
            mondrian.right = grow_mondrian(right,
            given_mondrian_next_budget, given_mondrian.right)
    else:
        dimensions_outer = dimensions(box)
        dimensions_inner = dimensions(given_mondrian.box, size=len(box))
        mondrian.cut_axis = random_axis(dimensions_outer - dimensions_inner)
        outer_interval = box[mondrian.cut_axis]

        if given_mondrian.is_empty():
            inner_interval = [outer_interval[0], outer_interval[0]]
        else:
            inner_interval = given_mondrian.box[mondrian.cut_axis]

        mondrian.cut_point, cut_side = sample_interval_difference
        (outer_interval, inner_interval)
        mondrian.cut_budget = next_budget

        left, right = cut_boxes(box, mondrian.cut_axis, mondrian.cut_point)

        if cut_side: # entire given_mondrian to the left
            mondrian.left = grow_mondrian(left, next_budget, given_mondrian)
            mondrian.right = grow_mondrian(right, next_budget,
            Mondrian(None, next_budget))
        else: # all given_mondrian to the right
            mondrian.left = grow_mondrian(left, next_budget,
            Mondrian(None, next_budget))
            mondrian.right = grow_mondrian(right,
            next_budget, given_mondrian)

    return mondrian

def get_random_color():
    return np.random.choice(['blue', 'red', 'yellow', 'white'], 1)[0]

def box_2d(box, color=None, alpha=None):
    low_x, high_x = box[0]
    low_y, high_y = box[1]
    width = high_x - low_x
    height = high_y - low_y

    if color is None:
        color = 'white'
    if alpha is None:
        alpha = 1

    lower_left_corner = np.array([low_x, low_y])
```

```
    return mpatches.Rectangle(lower_left_corner, width, height,
                          color=color, ec='black', linewidth=2,
                          alpha=alpha)

def boxes(m, box_collection, color=None):
    random_color = False

    if color == 'true_mondrian':
        random_color = True
        color = get_random_color()

    box_collection.append(box_2d(m.box, color))

    if m.has_cut():
        color = 'true_mondrian' if random_color else color
        boxes(m.left, box_collection, color)
        boxes(m.right, box_collection, color)

def plot_coloured_mondrian(m, ax, color=None, given_mondrian=None):
    if given_mondrian is None:
        given_mondrian = Mondrian(None, budget)

    box_collection = []
    boxes(m, box_collection, color)

    if not given_mondrian.is_empty():
        box_collection.append(box_2d(given_mondrian.box,
                              color='black', alpha=0.1))

    collection = PatchCollection(box_collection, match_original=True)
    ax.add_collection(collection)

    ax.axis('off')
    ax.autoscale()
    #plt.show()

def random_mondrians(box, budget, given_mondrian=None,
color=None, rows=1, columns=1, figsize=(15, 15)):
    if given_mondrian is None:
        given_mondrian = Mondrian(None, budget)

    if rows == 1 and columns == 1:
        fig, ax = plt.subplots(figsize=figsize)
        plot_coloured_mondrian(grow_mondrian(box, budget, given_mondrian),
                           ax, color=color, given_mondrian=given_mondrian)
    else:
        fig, ax = plt.subplots(rows, columns, figsize=figsize)

        for row in range(rows):
            for col in range(columns):
                if not given_mondrian.is_empty() and row == 0 and col ==0:
                    plot_coloured_mondrian(given_mondrian,
                    ax[row, col], color=color)
                else:
                    plot_coloured_mondrian(grow_mondrian(box, budget,
                    given_mondrian),
                                       ax[row, col], color=color,
```

```
                                    given_mondrian=given_mondrian)

def growing_mondrians(initial_box, budget,
rows=1, columns=1, figsize=(15, 15)):
   if rows == 1 and columns == 1:
      fig, ax = plt.subplots(figsize=figsize)
      plot_coloured_mondrian(grow_mondrian(box, budget),
                        ax, color=None)
   else:
      fig, ax = plt.subplots(rows, columns, figsize=figsize)

      given_mondrian = Mondrian(None, budget)
      box = initial_box
      for row in range(rows):
         for col in range(columns):
            mondrian = grow_mondrian(box, budget, given_mondrian)
            plot_coloured_mondrian(mondrian, ax[row, col], color=None,
            given_mondrian=given_mondrian)
            given_mondrian = mondrian
            box = 2 * box

random_mondrians(box, 1, rows=3, columns=3, color='true_mondrian')
plt.savefig("MondrianExmpl.pdf")
```

Mondrian Forest sur données "Boston Housing"

```
%matplotlib inline

import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.genmod
from statsmodels.tools.sm_exceptions import ConvergenceWarning
import matplotlib.pyplot as plt
from math import sqrt
import seaborn as sns
sns.set_palette("colorblind")
import sklearn.datasets
from sklearn.model_selection import train_test_split
import skgarden
from skgarden import MondrianForestClassifier
from skgarden import MondrianForestRegressor
from skgarden import MondrianTreeClassifier
from skgarden import MondrianTreeRegressor
from sklearn.linear_model import LinearRegression
import sklearn

from sklearn.datasets import load_boston

boston = load_boston()

df_x=pd.DataFrame(boston.data, columns=boston.feature_names)
df_y=pd.DataFrame(boston.target)

df_x.head(5)
```

```
sns.distplot(df_x.AGE)
plt.savefig("Age.pdf")

sns.distplot(df_x.RM)
plt.savefig("RM.pdf")

sns.distplot(df_x.TAX)
plt.savefig("TAX.pdf")

print(boston.DESCR)

# Mondrian Tree Regressor
mtr = MondrianForestRegressor()
mtr.fit(X, y)
y_mean, y_std = mtr.predict(X, return_std=True)
mtr.fit(X, y)
```