Achraf AZIZE
Ecole Polytechnique
achraf.azize@polytechnique.edu

26 août 2020

INTERPRÉTABILITÉ ET AUTOM

Déclaration d'intégrité relative au plagiat

Je soussigné : Achraf AZIZE certifie sur l'honneur :

1. Que les résultats décrits dans ce rapport sont l'aboutissement de mon travail.
2. Que je suis l'auteur de ce rapport.
3. Que je n'ai pas utilisé des sources ou résultats tiers sans clairement les citer et les référencer selon les règles bibliographiques préconisées. Je déclare que ce travail ne peut être suspecté de plagiat.

Date : 27 août 2020
Signature : Achraf AZIZE.

Résumé

Le Machine Learning automatisé, également appelé AutoML, est le processus d'automatisation des tâches fastidieuses et itératives de développement de modèles Machine Learning. Dans ce contexte, le DataLab du Crédit Agricole a développé un outil d'apprentissage automatisé, nommé MLBox, dans lequel une pipeline Machine Learning complète est proposée automatiquement pour résoudre un jeu de données téléchargé par le client. Le but de ce stage est d'enrichir cette solution MLBox sur deux niveaux : l'interprétabilité des modèles prédictifs, pour aider le client à mieux comprendre le modèle final, et l'intégration du deep learnig, pour dépasser l'étape d'extraction de features, qui nécessite une intervention humaine importante, et améliorer les performances générales de la MLBox.

A l'issue de ce stage, un package d'interprétabilité (Section 2.6.1) a été développé et intégré à la MLBox, un premier modèle Deep Learning avec quelques premières optimisations (Section 3.3.2) a aussi été intégré à la MLBox, et finalement le développement d'un script Python (Section 3.3.3) end-to-end permettant de trouver automatiquement le réseau de neurones avec performance optimal, sous quelques contraintes : un espace de recherche pré-défini, avec un budget en temps et en ressources de calculs. L'ensemble des développements effectués pendant le stage, ainsi que l'ensemble des notebooks permettant de reproduire les résultats présentés dans ce rapport sont disponible sur le git [13].

La section 2 traite la question de l'interprétabilité en définissant le problème, listant les différentes solutions étudiés dans l'état de l'art et exposant les résultats du package développé ainsi que son intégration à la MLBox. La section 3 retrace le chemin de conception suivi pour arriver à intégrer des réseaux de neurones automatiques à la MLBox, en commençant par définir le cas d'usage (données tabulaires, classification binaire, classes non balancées), précisant la définition de l'AutoML utilisée, et analysant les différents scénarios et développements permettant d'avoir une intégration automatique du deep learning dans la MLBox.

Je tiens à remercier Lishen SUN, ma tutrice qui m'a beaucoup aidé pendant ce stage, ainsi que Walid ERRAY, Emmeric TONNELIER et toute l'équipe analytique du DataLab, avec qui j'ai passé une très bonne expérience.

TABLE DES MATIÈRES

1 INTRODUCTION

1.1 AutoML : Démocratisation du Machine Learning

La dernière décennie a vu une explosion de la recherche et des applications en matière de Machine Learning ; vision par ordinateur, le traitement de la parole et les jeux. Toutefois, la conception d'algorithmes prometteurs pour un problème spécifique nécessite toujours un effort humain important. Les performances de nombreuses méthodes Machine Learning sont très sensibles à une pléthore de décisions de conception, ce qui constitue un obstacle considérable pour les nouveaux utilisateurs. Cela est particulièrement vrai dans le domaine en plein essor du deep learning, où les ingénieurs humains doivent sélectionner les bonnes architectures neurales, les procédures d'entraînement, les méthodes de régularisation et les hyperparamètres de toutes ces composants. Le domaine de l'apprentissage automatique (AutoML) vise à prendre ces décisions de manière objective, automatisée et en fonction des données : l'utilisateur fournit simplement des données, et le système AutoML détermine automatiquement l'approche la plus performante pour cette application particulière. Ainsi, AutoML vise à sortir l'humain de la boucle, et à rendre par conséquent les approches d'apprentissage automatique de pointe accessible au public ne disposant pas des ressources nécessaires pour connaître en détails les technologies qui le sous-tendent. Cela peut être considéré comme une démocratisation de l'apprentissage machine : avec AutoML, l'apprentissage machine personnalisé de pointe est à la portée de tous.

1.2 Crédit Agricole, DataLab et MLBox

Le Groupe Crédit Agricole est un réseau de banques coopératives et mutualistes au service du monde agricole, devenu depuis 1990 un groupe bancaire généraliste international. Créé en 2015 au sein du pôle Développement, Client et Innovation, le DataLab Groupe Crédit Agricole SA est un centre de compétence dédié aux sciences des données et à leurs applications dans le domaine bancaire. Son rôle est de créer des approches innovantes pour la valorisation de la donnée interne et externe. Dans le cadre de ses missions, des thématiques scientifiques à forte valeur ajoutée sont étudiées : apprentissage automatique AutoML, Analyse du Langage Naturel, Process Mining, Time Series Mining, Deep Learning, etc. Afin de répondre au besoin de ses clients, le DataLab Groupe Crédit Agricole a développé une solution interne AutoML, nommée la MLBox. Cependant, cette solution n'est pas totalement complète : Certes le nombre de choix et de décisions est moins important en comparaison avec un projet de Machine Learning classique, mais l'humain est encore très présent et important dans la boucle MLBox.

1.3 Objectifs du stage

Le but du stage est d'enrichir la MLBox sur deux niveaux :

En premier lieu : l'interprétabilité des modèles prédictifs. De nos jours, les modèles de 'Machine Learning' sont de plus en plus performant, capable d'apprendre directement depuis de grandes bases de données, avec une précision accrue. Cependant, l'introduction de biais incompris, et le manque d'interprétabilité de ces modèles ont générés des problématiques d'ordre éthique et juridique. Comme la MLBox est destinée à des clients non experts, celle-ci offre un outil d'interprétabilité basique pour aider ses clients à mieux comprendre le modèle prédictif final. Le premier objectif de ce stage est d'améliorer l'interprétabilité au sein de la MLBox. La section 2 traite plus en détails cette question.

En deuxième lieu : l'abscence de modèle deep learning dans la MLBox. Comme la MLBox est une solution destinée à des caisses régionales qui n'utilisent que des données tabulaires, la MLBox contient uniquement des modèles basées sur des arbres de décisions, algorithmes considérés 'champion' dans le domaine des données tabulaires. Ceci dit, le deep leaning est toujours une option interessante à explorer, surtout que c'est un processus end-to-end qui permet de dépasser la partie pre-processing et extraction de features : une partie qui prend beaucoup de temps et demande une intervention humaine considérable. La Section 3 étudie l'intérêt d'utiliser le deep learning dans le cadre d'usage de la MLBox, ainsi que l'ensemble des étapes, scénarios et choix effectués afin de l'intégrer complètement à la MLBox.

2 INTERPRÉTABILITÉ DES MODÈLES

2.1 Motivation

L'interprétabilité, i.e. la capacité d'expliquer ou de présenter des informations dans des termes humainement compréhensibles, est devenue une exigence minimale pour tout projet de 'Data Sciences'.

Cela est d'autant plus vrai que nous vivons dans une époque où les algorithmes prennent une place de plus en plus croissante dans notre vie quotidienne : attribution d'un crédit, recommandations, choix de nos trajets, etc. Cette multiplication d'algorithmes soulève néanmoins de nombreuses questions : Comment ont-ils été construits ? Comment fonctionnent-ils ? Comment expliquer leurs décisions ?

Les réponses à ces questions constituent un champ de recherche relativement récent, mais en pleine expansion dans le monde scientifique. Ces questions doivent être prises au sérieux par les entreprises désirant se doter de tels outils, au risque de voir se dégrader la relation avec leurs clients et l'adhésion des métiers vis-à-vis des projets accès sur les données.

2.2 Compromis Interprétabilité-Performance

En matière de méthodes et d'algorithmes d'apprentissage, les niveaux d'interprétabilité peuvent varier considérablement. Ainsi l'univers des modèles de 'Machine Learning' peut être scindé entre d'une part les modèles interprétables par nature (les régressions multilinéaires,les arbres et règles de décision, etc.) et les modèles dits «boites noires» (Random Forest, réseaux de neurones, etc.)

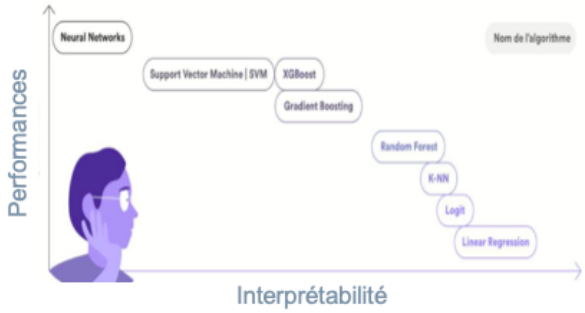


Figure 1. Illustration du Compromis Interprétabilité-Performance

Ceci représente un compromis auquel fait face tout projet de développement en lien avec le 'Machine Learning' : où se situe le curseur idéal entre performance et interprétabilité ? La modélisation est, en effet, généralement caractérisée par une relation inversement proportionnelle entre performance et interprétabilité.

Améliorer l'interprétabilité des modèles de Machine Learning est un des principaux leviers à disposition des équipes de 'Data Sciences' pour remplir les critères de réussite de développement d'un projet. Cela permet en effet de sortir de la dualité Interprétabilité vs Performance, qui pourrait jouer en défaveur des modèles potentiellement plus performants.

2.3 Interprétabilité Globle et Interprétabilité Locale

Comprendre et expliquer les modèles sont donc un des enjeux majeurs dans les projets reposant sur le 'Machine Learning'. Quelle est la démarche à suivre pour y arriver ? Quelles sont les solutions existantes ? Pour répondre à ces questions, deux catégories de techniques émergent : l'interprétabilité globale et l'interprétabilité locale.

L'interprétabilité globale cherche à identifier les variables les plus importantes du modèle notamment via une analyse minutieuse de la contribution de chaque variable sur les données de sortie du modèle. Quelles est leur contribution sur les performances du modèle ? Quelle est la relation entre chaque variable et la sortie du modèle ?. L'interprétabilité globale doit permettre à terme d'améliorer la compréhension du modèle par les experts métiers et donc de mieux s'approprier les résultats.

De son côté, l'interprétabilité locale cherche à décrypter le comportement du modèle à l'échelle d'un individu en identifiant l'impact et la contribution local de chaque variable. Cette méthode doit permettre d'améliorer la communication et la justification des résultats de l'algorithme à l'utilisateur final.

2.4 Formulations du problème de l'interprétabilité

Un examen de la littérature [1] permet d'identifier deux grandes catégories de sous problèmes de l'interprétation : **l'ingénierie inverse** ('Reverse Engineering) et **la conception des explications** ('Design of Explication').

Dans le premier cas, compte tenu d'un prédicteur 'boîte noire' déjà entraîné et de ses différentes décisions produites, le problème consiste à reconstruire une explication. L'ensemble de données original sur lequel la boîte noire s'est entraîné n'est généralement pas connu. Dans le second cas, à partir d'un ensemble de données d'entraînement, la tâche consiste à élaborer un modèle prédicteur interprétable, ainsi que ses explications. Avant d'approfondir et bien définir ces deux catégories, commençons par introduire le vocabulaire et les différentes notations nécessaires. On précise qu'on reprend dans cette section les même défnitions et formalismes utilisés par Guidotti et al. [1].

2.4.1 • Notations et vocabulaire

Un prédicteur, également appelé modèle prédicteur, est une fonction $b : X^m \to Y$, où X^m est l'espace des features (ou variables ou caractéristiques), m le nombre de features et Y l'espace cible. L'espace de features X peut correspondre à tout type basique de données : nombres réels, 'booleans', 'strings', comme il peut être complexe, combinant différent type basique.

Un prédicteur b est le résultat d'une fonction d'apprentissage L_b, tel que $L_b : (X^{n\times m} \times Y^n) \to (X^m \to Y)$, avec n le nombre d'exemples : L_b prend en entrée une base de données $D = \left\{(x_i, y_i)_{1\leq i\leq n}\right\}$, et retourne un prédicteur b, qu'on peut utiliser pour prédire la cible $\hat{y}, i.e., b(x) = \hat{y}$.

Dans le cadre de l'apprentissage supervisé, on utilise généralement une base de donnée d'entraînement D_{train} pour entraîner le learner $L_b(D_{train})$ qui permet de construire un prédicteur b. On utilise après une base de donnée de test D_{test} pour évaluer la performance de b. D_{train} et D_{test} sont tiré avec la même distribution.

Dans ce qui suit, nous indiquons avec b un prédicteur 'boîte noire' appartenant à l'ensemble pas interprétables. b est une boîte noire parce que le raisonnement qui sous-tend la fonction n'est pas compréhensible par les humains et le résultat rendu ne fournit aucun indice pour son choix. De même, nous indiquons par c un prédicteur compréhensible pour lequel est disponible une explication globale ou locale.

2.4.2 • Reverse Engineering

Le Reverse Engineering peut encore se raffiner en 3 sous catégories :

2.4.2.1 Explication de modèle Black Box

Définition 1.1 *Etant donné un prédicteur 'boîte noire' b et une base de données $D = \{X, Y\}$, le problème d'Explication de modèle Black Box consiste à trouver une fonction $f : (X^m \to Y) \times (X^{n\times m} \times Y^n) \to (X^m \to Y)$, qui prend en entrées b et D, et retourne $c_g = f(b, D)$, avec c_g un prédicteur fidèle à b, et globalement explicable, i.e. il existe une fonction d'explication globale $\epsilon_g : (X^m \to Y) \to \Sigma$ qui permet d'extraire de c_g un ensemble d'explication $E = \epsilon_g(c_g) \in \Sigma$ qui modélise d'une manière humainement compréhensible la logique interne du modèle c_g.*

L'idée est de trouver un prédicteur globalement explicable et fidèle au modèle 'boîte noire' i.e. permet de reproduire à une fidélité près les décisions du prédicteur 'boîte noire'. L'ensemble d'explication E peut être un arbre de décision ou un ensemble de décision, et par conséquent c_g peut être un classifieur arbre de décision, ou classifieur règle de décision.

2.4.2.2 Explication des résultats de la Black Box

Définition 1.2 *Etant donné un prédicteur 'boîte noire' b et une base de données $D = \{X, Y\}$, le problème d'Explication des résultats de la Black Box consiste à trouver une fonction $f : (X^m \to Y) \times (X^{n\times m} \times Y^n) \to (X^m \to Y)$, qui prend en entrées b et D, et retourne $c_l = f(b, D)$, avec c_l un prédicteur fidèle à b, et localement explicable, i.e. il existe une fonction d'explication locale $\epsilon_l : (X^m \to Y) \times (X^m \to Y) \times X^m \to \Sigma$ qui permet d'extraire à partir du modèle 'boîte noire' b, de c_l et d'un point de donnée $x \in X^m$ un ensemble d'explication $e = \epsilon_l(b, c_g, x) \in \Sigma$.*

La différence est que l'explication ici n'est valable que pour un point de donnée $x \in X^m$, et $e \in \Sigma$ peut être dans ce cas un chemin dans l'arbre de décision ou une association de règle de décisions précise qui explique la prédiction pour le point x.

2.4.2.3 Problème d'inspection Black Box

Définition 1.3 *Etant donné un prédicteur 'boîte noire' b et une base de données $D = \{X, Y\}$, le problème d'inspection Black Box consiste à trouver une fonction $f : (X^m \to Y) \times (X^{n\times m} \times Y^n) \to V$, qui prend en entrées b et D, et retourne une représentation visuelle du comportement du prédicteur b, $v = f(b, D) \in V$.*

Par exemple, la fonction f peut être une technique basée sur l'analyse de sensibilité, qui en observant les changement dans les prédictions en variant les entrées de b, retourne un ensemble de visuations (courbes de dépendances partielles, classement d'importance de variables).

2.4.3 • Conception d'explications

Définition 2 *Etant donné une base de données $D = \{X, Y\}$, le problème de Conception d'explications consiste à trouver une fonction d'apprentissage $L_c : (X^{n\times m} \times Y^n) \to (X^m \to Y)$, qui prend en entrée D, et retourne un prédicteur compréhensible (localement ou globalement) $c = L_c(D)$.*

Un exemple de fonction L_c peut être la fonction d'apprentissage d'un arbre de décision.

2.4.4 • Ouvrir la 'boîte noire'

Selon ces différentes définitions, quand on dit qu'une méthode est capable 'd'ouvrir une boîte noire', on refère à l'une des posssiblités : (i) on explique un modèle, (ii) on explique un résultat, (iii) on inspecte le modèle avec des visualisations, (iv) on fournit une solution interpretable.

2.5 Etat de l'art des solutions

Une analyse de l'état de l'art des solutions capable d'ouvrir la 'boîte noire' nous permet de les classifier selon les éléments suivant :

- le type du problème rencontré (expliquer un modèle, un résultat, inspecter un modèle, concevoir un modèle interpretable)
- le type d'explicateur utilisé (arbre de décision, règles de décisions, classement d'importance, courbes de dépendance partielle, etc)
- le type de 'boîte noire' que la solution permet d'ouvrir (réseau de neurones, xgboost, LGBM, SVM, etc)
- le type des données utilisées (tabulaire, image, série temporelle, etc)

Le tableau 1 de R.Guidotti et al. [1] contient une classification complète de toutes les solutions existantes dans l'état de l'art, selon ces même quatre critères. Dans le cadre de notre étude, on s'intéresse qu'aux méthodes d'interprétabilité agnostiques, i.e. capale d'expliquer tout les types de modèles 'boîte noire'. On ne s'interesse aussi qu'aux données tabulaires, car la version actuelle de la MLBox ne prend en charge que ce type de données.

Que va-t-on interpréter	Comment va-t-on l'interpréter	Solution de quel problème ?	Solutions
Modèle	Feature Importance	Problème d'inspection Black Box	PI* [13]
		Problème d'inspection Black Box	SHAP [2]
	Arbre de décision	Explication de modèle Black Box	TREPAN
	Dépendance partielle	Conception d'explications	GAM
Réultat	Contribution partielle	Explication des résultats de la Black Box	SHAP [2]
	Modèle Linéaire	Explication des résultats de la Black Box	LIME [3]
	Règles de décisions	Explication des résultats de la Black Box	Anchor [4]

Table 1. Catégorisation des différentes solutions étudiées et implémentées dans le package Interpreter

(* PI = Permutation Importance)
Pour chacune de ces solutions, on s'interesse à son principe, ses avantages et désavantages. L'implémentation et les différents résultats pratiques sont présentés à la section (2.6).

2.5.1 • Permutation Importance

Permutation Importance est une méthode d'interpretation permettant de mesurer l'importance d'une 'feature' en mesurant la modification de l'erreur de prédiction du modèle après que nous avons permuté les valeurs de cette 'feature' en question, ce qui brise la relation entre cette feature et le résultat réel. C'est une méthode agnostique, solution du Problème d'inspection Black Box, expliquant la 'boîte noire' avec un classement d'importance, et applicable aux données tabulaires.

2.5.1.1 Principe

Le concept est vraiment simple : Nous mesurons l'importance d'une feature en calculant la modification dans l'erreur de prédiction du modèle après la permutation de cette feature. Une feature est "importante" si la permutation de ses valeurs augmente l'erreur du modèle, car dans ce cas, le modèle s'est appuyé sur cette feature pour la prédiction. Une feature est "pas importante" si la permutation de ses valeurs laisse l'erreur du modèle inchangée, parce que dans ce cas le modèle l'a ignoré pour la prédiction. La mesure de l'importance de la feature de permutation a été d'abord

introduite pour les forêts aléatoires. Sur la base de cette idée, Fisher, Rudin et Dominici (2018) [6] ont proposé une version agnostique.

Algorithme 1 : L'algorithme de 'permutation importance' basé sur Fisher, Rudin et Dominici (2018)

Data : Modèle entrainé f, matrice de features X $\in R^{n \times m}$ (n le nombre d'exemples, m le nombre de features), vecteur cible y, une mesure d'erreur (loss) L(y,f(X)).

Result : Classement des features par Importance

Estimer l'erreur du modèle original $er^{orig} = L(y, f(X))$ (par exemple, erreur quadratique moyenne);

for *feature j = 1, ...,m* **do**

  Générer la matrice de features X^{perm} en permutant la feature j dans les données X ;

  Estimer l'erreur $er^{perm} = L(Y, f(X^{perm}))$ sur la base des prédictions des données permutées ;

  Calculer l'importance de la feature de permutation $FI^j = er^{orig}/er^{perm}$;

Trier les importances FI par ordre décroissant ;

2.5.1.2 Avantages

Permutation Importance fournit un aperçu global, hautement comprimé du comportement du modèle et inuitif.

La mesure de l'importance prend automatiquement en compte toutes les interactions avec les autres features. En permutant la feature, vous détruisez également les effets d'interaction avec d'autres features. Cela signifie que l'importance prend en compte à la fois l'effet de la feature principale et les effets d'interaction sur la performance du modèle.

Permutation Importance n'exige pas un réentraînement du modèle. D'autres méthodes suggèrent de supprimer une feature, de réentrainer le modèle et de comparer ensuite l'erreur du modèle. Comme le réentraînement d'un modèle machine learning peut prendre beaucoup de temps, seule la permutation d'une feature peut faire gagner beaucoup de temps.

2.5.1.3 Désavantages

D'autre part, l'importance dépend de comment on permute les features, ce qui ajoute un caractère aléatoire à la mesure. Lorsque la permutation est répétée, les résultats peuvent varier considérablement. Répéter plusieurs permutations et moyenner les mesures d'importance stabilise la mesure, mais augmente le temps de calcul.

Vous devez avoir accès au véritable 'labels'. Si quelqu'un vous fournit uniquement le modèle entrainé et les données non étiquetées, vous ne pouvez pas calculer l'importance.

Si les caractéristiques sont corrélées, l'importance de permutation peut être biaisée par des instances de données irréalistes : La permutation des features produit des cas de données peu probables lorsque deux ou plusieurs features sont corrélées. Comme pour la taille et le poids d'une personne, et que je mélange une des caractéristiques, je crée de nouvelles instances qui sont peu probables, voire physiquement impossibles (personne de 2 mètres pesant 30 kg par exemple), mais je les utilise pour mesurer l'importance.

2.5.2 • SHAP

SHAP est une méthode d'interprétation agnostique, qui permet d'estimer la contribution exacte de chaque feature dans la prédiction finale. La méthode d'explication SHAP calcule les 'Shapley Values' emprunté de la théorie des jeux coopératives. Les 'features' d'une instance de données agissent comme les acteurs d'une coalition. Les 'Shapley Values' nous indiquent comment répartir équitablement le "gain"(la prédiction) entre les "joueurs"(les features).

2.5.2.1 Principe

Une prédiction peut s'expliquer en supposant que chaque valeur d'une feature d'une instance est un "joueur", dans un jeu où la prédiction est le "gain". Les 'Shapley Values' nous indiquent comment répartir équitablement ce "gain"

entre les features. La 'Shapley Value' est la contribution marginale moyenne d'une 'feature value' à travers toutes les coalitions possibles.

Prenons un exemple concret pour mieux comprendre l'idée. Supposons que mon modèle entraîné de Random Forest me permet de predire le prix d'une maison à partir du nombre de chambre, la superficie et l'adresse. Et supposons que mon modèle prédit que pour 3 chambres, une superficie de $100m^2$ et au centre de Paris, le prix est de 150 000 euros. Sachant que le prix moyen d'une maison est de 100 000 euros à Paris (par exemple), le but est d'expliquer la difference (50 000) et la partager entre les features : par exemple que 10 000 euros vient du fait qu'on a 3 chambres, 5000 euros parce que c'est $100m^2$ et 35 000 euros parce que c'est au centre de Paris.

Pour un modèle quelconque, la 'Shapley Value' est définie à partir d'une fonction de valeur *val* qui quantifie la valeur de chaque ensemble de joueurs. La 'Shapley Value' d'une 'feature' est sa contribution au gain (prédiction), pondérée et additionnée sur toutes les combinaisons possibles de 'features' :

$$\phi_j(val) = \sum_{S \subseteq \{x_1,...,x_p\} \setminus \{x_j\}} \frac{|S|!(p-|S|-1)!}{p!}(val(S \cup \{x_j\}) - val(S))$$

avec $val(S)$ est la prédiction des valeurs pour l'ensemble de features S marginalisées par rapport aux 'features' qui ne sont pas dans l'ensemble S :

$$val(S) = \int \hat{f}(x_1, ..., x_p) d\mathbb{P}_{x \notin S} - E_X(\hat{f}(X))$$

La 'Shapley Values' est la seule méthode qui satisfait les propriétés d'efficacité, de symétrie, de facticité et d'additivité, qui ensemble peuvent être considérées comme une définition d'un 'gain' équitable.
Efficacité : $\sum_{j=1}^{p} \phi_j(\hat{f}) = \hat{f}(x) - E(\hat{f}(X))$

Symétrie : Les contributions de deux valeurs j et k devraient être les mêmes si elles contribuent de manière égale à toutes les coalitions possibles :

$$\text{Si } val(S \cup \{x_j\}) = val(S \cup \{x_k\}) \text{ pour tout } S \subseteq \{x_1, ..., x_p\} \setminus \{x_j, x_k\}$$

alors $\phi_j = \phi_k$

Factice : Une 'feature' j qui ne change pas la valeur prédite - quelle que soit la coalition des valeurs de 'features' à laquelle elle est ajoutée - devrait avoir une valeur de Shapley de 0.

$$\text{Si } val(S \cup \{x_j\}) = val(S) \text{ pour tout } S \subseteq \{x_1, ..., x_p\} \setminus \{x_j, x_k\}$$

alors $\phi_j = 0$

Additivité : Pour un jeu avec des gains combinés $val_1 + val_2$, les valeurs respectives de Shapley sont les suivantes :

$$\phi_1 + \phi_2$$

Supposons que vous aviez entraîné un 'Random Forest', ce qui signifie que la prédiction est une moyenne de plusieurs arbres de décision. La propriété d'additivité garantit que pour une 'feature', vous pouvez calculer la 'Shapley Value' pour chaque arbre individuellement, faire la moyenne et obtenir la valeur de Shapley pour cette 'feature' du 'Random Forest'.

Estimation des Shapley Values :

Toutes les coalitions possibles de 'features' doivent être évaluées avec et sans la j-ième 'feature', pour calculer la valeur exacte de Shapley. Avec plus de 'features', la solution exacte à ce problème devient problématique car le nombre de coalitions possibles augmente de manière exponentielle.
Strumbelj et al. [5] proposent une approximation avec l'échantillonnage de Monte-Carlo :

$$\hat{\phi_j} = \frac{1}{M} \sum_{i=1}^{m} (\hat{f}(x_{+j}^m) - \hat{f}(x_{-j}^m))$$

avec $x_{+j}^m)et x_{-j}^m)$ construit avec l'algorithme :

Algorithme 2 : Approximation des Shapley Values avec l'échantillonnage de Monte-Carlo

Data : un modèle entraîné f, une instance à expliquer x, le nombre de simulations M, j l'index de la feature pour laquelle on veut calculer SHAP, Matrice de data X

for *m = 1,...M* **do**
- On tire une instance z aléatoirement de X ;
- On chosit une permutation σ des features ;
- On applique σ à x et z : $x_\sigma = (x_1, ..., x_p)$ et $z_\sigma = (z_1, ..., z_p)$;
- On construit deux features :
  - $x_{+j} = (x_1, ..., x_{j-1}, x_j, z_{j+1}, ..., z_p)$
  - $x_{-j} = (x_1, ..., x_{j-1}, z_j, z_{j+1}, ..., z_p)$
- On calcule $\phi_j^m = \hat{f}(x_{+j}) - \hat{f}(x_{-j}))$;

La shap value de la 'feature' j est : $\phi_j(x) = \frac{1}{M} \sum_{i=1}^{m} (\phi_j^m)$

2.5.2.2 Avantages

La différence entre la prédiction et la prédiction moyenne est équitablement totalement expliquée par les 'Shapley Values' des 'features' de l'instance : la propriété d'efficacité des valeurs de Shapley. Cette propriété distingue la valeur de Shapley des autres méthodes. La 'Shapley Value' pourrait être la seule méthode permettant de fournir une explication complète. Dans les situations où la loi exige une explication – comme le "droit aux explications" de l'UE – la valeur de Shapley pourrait être la seule méthode conforme à la loi, car elle est basée sur une théorie solide et distribue les effets de manière équitable. Je ne suis pas juriste, ce qui ne reflète que mon intuition quant aux exigences. Il est aussi intéressant d'expliquer une prédiction comme un jeu joué par les valeurs des 'features'.

2.5.2.3 Désavantages

SHAP nécessite beaucoup de temps de calcul. Dans 99,9 % des problèmes du monde réel, seule la solution approximative est possible. Un calcul exact de la valeur de Shapley est coûteux car il y a 2^k coalitions possibles des valeurs des caractéristiques. Le nombre exponentiel des coalitions est traité par des coalitions d'échantillonnage et en limitant le nombre d'itérations M. La diminution de M réduit le temps de calcul, mais augmente la variance de la valeur de Shapley.

La valeur de Shapley peut être mal interprétée. La 'Shapley Value' d'une 'feature' **n'est pas** la différence de la valeur prédite après avoir retiré la caractéristique de l'apprentissage du modèle. L'interprétation de la valeur de Shapley est la suivante : Compte tenu de l'ensemble actuel des valeurs des 'features', la contribution d'une valeur d'une 'feature' à la différence entre la prédiction réelle et la prédiction moyenne est la valeur de Shapley estimée.

SHAP renvoie une 'valeur simple' par 'feature', mais pas de modèle de prédiction comme LIME ou d'autres modèles surrogate. Cela signifie que SHAP ne ne peut pas être utilisée pour faire des simulations sur les changements de prédiction pour des changements dans l'entrée.

Un autre inconvénient est que vous devez avoir accès aux données si vous voulez calculer la valeur de Shapley pour une nouvelle instance de données. L'accès à la fonction de prédiction n'est pas suffisant, car vous avez besoin

des données pour remplacer des parties de l'instance qui vous intéresse par des valeurs provenant d'instances de données tirées au hasard.

Et comme pour Permutation Importance, la méthode des valeurs de Shapley souffre de l'inclusion d'instances de données irréalistes lorsque les caractéristiques sont corrélées. Pour simuler qu'une valeur d'une 'feature' est manquante dans une coalition, nous marginalisons cette 'feature' en échantillonnant des valeurs de sa distribution marginale. Cela est possible tant que les caractéristiques sont indépendantes. Lorsque les caractéristiques sont dépendantes, nous pouvons alors échantillonner des valeurs de features qui n'ont pas de sens dans ce cas. Une solution pourrait être de procéder à l'échantillonnage en tenant en compte de la dépendance des caractéristiques.

2.5.3 • LIME

LIME (Local interpretable model-agnostic explanations) est un modèle 'surrogate' (modèle de substitution) utilisé pour expliquer les prédictions individuelles des modèles 'boîte noire'. Les modèles de substitution sont formés pour se rapprocher des prédictions du modèle sous-jacent de la boîte noire. Au lieu de former un modèle de substitution global, le LIME se concentre sur la formation de modèles de substitution locaux pour expliquer les prédictions individuelles.

2.5.3.1 Principe

L'idée est assez intuitive. Tout d'abord, oublions les données d'entraînement de la boîte noire. Imaginons qu'on a juste une boîte noire qu'on peut utiliser tant qu'on veut. L'objectif est de comprendre pourquoi la boîte a fait une certaine prédiction. LIME teste ce qui arrive aux prédictions lorsqu'on pertube localement l'instance qu'on veut expliquer. LIME génère un nouvel ensemble de données composé d'échantillons permutés et des prédictions correspondantes du modèle de la boîte noire. Sur ce nouvel ensemble de données, LIME forme ensuite un modèle interprétable, qui est pondéré par la proximité des instances échantillonnées par rapport à l'instance d'intérêt. Le modèle interprétable peut être n'importe quoi des modèles interprétables, par exemple Lasso ou un arbre de décision. Le modèle appris doit être une bonne approximation des prédictions du modèle boîte noire au niveau local, mais il ne doit pas nécessairement être une bonne approximation globale. Ce type de précision est également appelé fidélité locale.

La recette pour former des modèles de substitution locaux :

- Sélectionnez l'instance qui vous intéresse et pour laquelle vous voulez avoir une explication de sa prédiction.
- Perturbez votre ensemble de données et obtenez les prédictions de la boîte noire pour ces nouveaux points.
- Pondérez les nouveaux échantillons en fonction de leur proximité par rapport à l'instance d'intérêt.
- Formez un modèle pondéré et interprétable sur l'ensemble de données avec les variations.
- Expliquez la prédiction en interprétant le modèle local.

Mathématiquement, les modèles de substitution locaux avec contrainte d'interprétabilité peuvent être exprimés comme suit :

$$explanation(x) = argmin_{g \in G} L(f, g, \pi) + \Omega(g)$$

Le modèle d'explication pour x est le modèle g (par exemple le modèle de régression linéaire) qui minimise la perte L (par exemple l'erreur quadratique moyenne), qui mesure la proximité de l'explication par rapport à la prédiction du modèle original f (par exemple un modèle xgboost), alors que la complexité du modèle Ωg est maintenu à un niveau bas (par exemple, préférer moins de 'features'). G est la famille des explications possibles, par exemple tous les modèles de régression linéaire possibles. La mesure de proximité $\pi(x)$ définit l'étendue du voisinage

autour de l'instance x que nous considérons pour l'explication. En pratique, le LIME n'optimise que la partie perte. L'utilisateur doit déterminer la complexité, par exemple en sélectionnant le nombre maximum de features que le modèle de régression linéaire peut utiliser.

2.5.3.2 Avantages

La modélisation des prédictions sous forme de somme pondérée rend transparente la manière dont les prédictions sont produites. Et grâce au Lasso, nous pouvons nous assurer que le nombre de 'features' utilisées reste faible.

De nombreuses personnes utilisent des modèles de régression linéaire. Il existe un niveau élevé d'expérience et d'expertise collectives, y compris du matériel pédagogique sur les modèles de régression linéaire et des mises en œuvre de logiciels. La régression linéaire peut être trouvée dans R, Python, Java, Julia, Scala, Javascript. Mathématiquement, il est simple d'estimer les poids et vous avez la garantie de trouver les poids optimaux (étant donné que toutes les hypothèses du modèle de régression linéaire sont respectées par les données). Avec les poids, on peut obtenir des intervalles de confiance, des tests et une théorie statistique solide. Il existe également de nombreuses extensions du modèle de régression linéaire (voir le chapitre sur GAM).

2.5.3.3 Désavantages

Les modèles de régression linéaire ne peuvent représenter que des relations linéaires, c'est-à-dire une somme pondérée de 'features' d'entrée. Chaque non-linéarité ou interaction doit être fabriquée à la main et donnée explicitement au modèle en tant que 'features' d'entrée.

Les modèles linéaires ne sont souvent pas non plus très performants en matière de prédiction, car les relations qui peuvent être apprises sont très limitées et simplifient généralement la complexité de la réalité.

L'interprétation d'un poids peut ne pas être très intuitive car elle dépend de toutes les autres features. Une feature présentant une corrélation positive élevée avec le résultat y et peut quand même obtenir un poids négatif dans le modèle linéaire, car, étant donné d'autres features corrélées, elle est négativement corrélée avec y dans l'espace à haute dimension.

2.5.4 • Anchor

Anchor explique les prédictions individuelles de tout modèle de classification 'boîte noire', en trouvant une règle de décision qui "ancre" suffisamment la prédiction.

2.5.4.1 Principe

Comme son prédécesseur, l'approche d'Anchor déploie une stratégie basée sur les perturbations afin de générer des explications locales pour les prédictions des modèles d'apprentissage 'boîte noire'.

Anchors utilise des techniques d'apprentissage par renforcement, combiné avec un algorithme de recherche de graphes afin de réduire au minimum le nombre d'appels du modèle (et donc la durée d'exécution requise). Ribeiro, Singh et Guestrin ont proposé l'algorithme en 2018, les mêmes chercheurs qui ont introduit l'algorithme LIME. Toutefois, au lieu des modèles de substitution utilisés par LIME, les explications qui en résultent sont exprimées sous la forme de règles IF-THEN faciles à comprendre, appelées ancres.

Un ancre A est formellement défini comme suit :

$$\mathbb{E}_{D_x(z|A)}\left[1_{f(z)=f(x)}\right] \geq \tau, A(x) = 1$$

Où :

- x représente l'instance à expliquer

- A est un ensemble de prédicats, c'est-à-dire de règles ou ancres de telle sorte que les 'features' de x les verifie.
- f le modèle 'boîte noire'
- $D_x(.|A)$ indique la distribution du voisinage de x, correspondant à A
- $0 \leq \tau \leq 1$ précise un seuil de précision. Seules les règles qui atteignent une fidélité locale d'au moins τ sont considérés comme un résultat valable.

Bien que la description mathématique des ancres puisse sembler claire et simple, la construction de règles est pratiquement irréalisable. Il faudrait évaluer $1_{f(z)=f(x)}$ pour tout $z \in D_x(.|A)$ ce qui n'est pas possible dans des espaces d'entrée continus ou larges. C'est pourquoi les auteurs proposent d'introduire le paramètre δ pour créer une définition probabiliste. De cette façon, les échantillons sont tirés jusqu'à ce qu'il y ait une confiance statistique concernant leur précision. La définition probabiliste se lit comme suit :

$$P(prec(A) \geqslant \tau) \geqslant 1 - \delta \text{ avec } prec(A) = \mathbb{E}_{D_x(z|A)}$$

Les deux définitions précédentes sont combinées et étendues par la notion de couverture. Sa raison d'être consiste à trouver des règles qui s'appliquent à une partie de préférence importante de l'espace d'entrée du modèle. La couverture est formellement définie comme une probabilité d'ancrage qui s'applique à ses voisins, c'est-à-dire à son espace de perturbation :

$$cov(A) = \mathbb{E}_{D_{(z)}}[A(z)]$$

Ainsi, la procédure vise à obtenir une règle qui a la couverture la plus grande parmi toutes les règles éligibles (toutes celles qui satisfont le seuil de précision donné par la définition probabiliste).

L'approche pour trouver ancres utilise quatre composantes principales, comme le montre la figure ci-dessous.

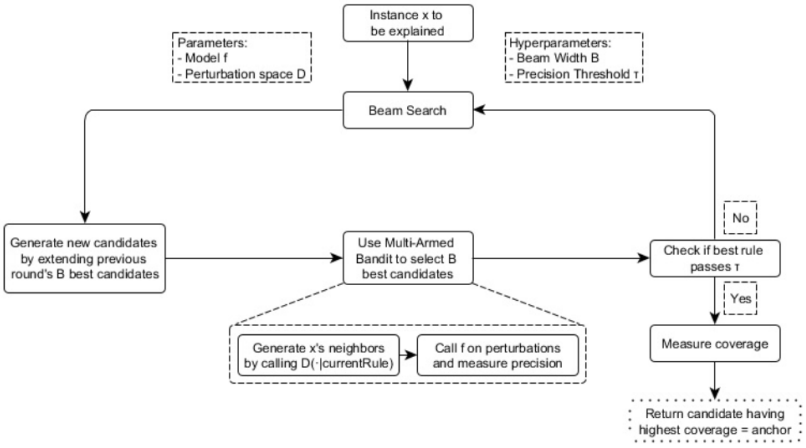


Figure 2. Les composantes de l'algorithme d'Anchor et leurs interrelations (simplifié)

2.5.4.2 Avantages

L'approche par ancrage offre de multiples avantages par rapport au LIME. Premièrement, la sortie de l'algorithme est plus facile à comprendre, car les règles de décision sont faciles à interpréter. De plus, les ancres peuvent même indiquer une mesure d'importance en incluant la notion de couverture.

Deuxièmement, l'approche des ancres fonctionne lorsque les prédictions du modèle sont non linéaires ou complexes dans le voisinage d'une instance. Comme l'approche déploie des techniques d'apprentissage par renforcement au lieu d'adapter des modèles de substitution, il est moins probable qu'elle sous-adapte le modèle.

En outre, l'algorithme est agnostique par rapport au modèle et donc applicable à n'importe quel modèle.

2.5.4.3 Désavantages

L'algorithme souffre d'un 'setup' hautement configurable et percutant, comme la plupart des explications basées sur les perturbations. Non seulement les hyperparamètres tels que la largeur du faisceau δ ou le seuil de précision δ doivent être réglés pour donner des résultats significatifs, mais la fonction de perturbation doit également être explicitement conçue pour un domaine d'utilisation.

En outre, de nombreux scénarios nécessitent une discrétisation, car sinon les résultats sont trop spécifiques, ont une faible couverture et ne contribuent pas à la compréhension du modèle. Si la discrétisation peut être utile, elle peut également brouiller les limites de la décision si elle est utilisée de manière négligente et avoir ainsi l'effet inverse. Comme il n'existe pas de meilleure technique de discrétisation, les utilisateurs doivent être conscients des données avant de décider de la manière de les discrétiser.

La construction d'ancres nécessite de nombreux appels au modèle 'boîte noire', comme tous les explicatifs basés sur les perturbations. Bien que l'algorithme déploie des algorithmes pour minimiser le nombre d'appels, sa durée d'exécution dépend encore beaucoup des performances du modèle et est donc très variable.

Enfin, la notion de couverture est indéfinie dans certains domaines. Par exemple, il n'existe pas de définition évidente ou universelle de la manière dont les superpixels d'une image se comparent à ceux d'autres images.

2.5.5 • GAM

Les modèles linéaires ont de nombreuses hypothèses qui malheureusement sont souvent violées dans la réalité : La cible y sachant les attributs peut avoir une distribution non gaussienne, les attributs peuvent interagir et la relation entre eux et le résultat peut être non linéaire. La bonne nouvelle est que GAM ('Generalized Additive Models') permet de dépasser ces limites.

2.5.5.1 Principe

Le modèle GAM est :

$$g(\mathbb{E}_Y(y|x)) = \beta_0 + f_1(x_1) + f_2(x_2) + ... + f_p(x_p)$$

La formule est similaire aux modèles linéaires, à quelques différemces :

- une distribution de probabilité de la famille exponentielle qui définit E_Y
- une fonction de lien g, pour chaque distribution de la famille exponentielle existe une fonction canonique de lien g (l'identité pour les gaussiennes, ln pour Poisson et logit pour Bernoulli)
- le terme linéaire est remplacé par une fonction plus souple f_i. Le cœur d'un GAM reste une somme d'effets des features, mais vous avez la possibilité d'autoriser des relations non linéaires entre certaines caractéristiques et la sortie. En pratique, on utilise les fonction de 'splines'.

- les x_i peuvent être des features ou produit de features (pour modéliser les interractions).

2.5.5.2 Avantages

Toutes ces extensions du modèle linéaire sont un peu un univers en soi. Quels que soient les problèmes que vous rencontrez avec les modèles linéaires, vous trouverez probablement une extension qui y remédiera.

En plus de faire des prédictions, vous pouvez utiliser les modèles pour faire des inférences, tirer des conclusions sur les données, étant donné que les hypothèses du modèle ne sont pas violées. Vous obtenez des intervalles de confiance pour les poids, les tests de signification, les intervalles de prédiction et bien plus encore.

Les logiciels statistiques ont généralement de très bonnes interfaces pour s'adapter aux GAM et à des modèles linéaires plus spécifiques.

En supposant que les modèles linéaires sont hautement interprétables mais souvent inadaptés à la réalité, les extensions décrites dans ce chapitre offrent un bon moyen de réaliser une transition en douceur vers des modèles plus flexibles, tout en préservant une partie de l'interprétabilité.

2.5.5.3 Désavantages

La plupart des modifications du modèle linéaire rendent le modèle moins interprétable. Toute fonction de lien g (dans un GAM) qui n'est pas la fonction d'identité complique l'interprétation ; les interactions compliquent également l'interprétation ; les effets des features non linéaires sont soit moins intuitifs (comme la transformation logarithmique), soit ne peuvent plus être résumés par un seul nombre (par exemple les fonctions spline).

Les performances des ensembles basés sur les arbres, comme la forêt aléatoire ou le renforcement des arbres par gradient, sont dans de nombreux cas meilleures que celles des modèles linéaires les plus sophistiqués. Il s'agit en partie de ma propre expérience et en partie d'observations des modèles gagnants sur des plateformes comme kaggle.com

2.6 Résultats et implémentations

2.6.1 • Le package 'Interpreter'

Toutes les solutions décrites dans la section 2.3 sont disponible dans un seul package développé pendant ce stage, et disponible sur le git [13] :

Une structure commune à été adopté dans le développement de toutes ces méthodes. Pour utiliser l'une des solutions de l'interpret toolbox, il faut suivre trois étapes :

Etape 1 : Importer la méthode à utiliser (SHAP dans l'exemple ci-dessous)

```
from interpret_toolbox.explainers import ShapExplainer
```

Etape 2 : Initier une instance de la méthode utilisé. Tous les modèles ont besoin en entrée d'un modèle entrainée, d'une base de données d'entrées, le nom des features, le nom des classes et le mode d'utilisation

```
exp_shap = ShapExplainer(model = f, data = X_train, feature_names = feature_names, mode = '
    classification' )
```

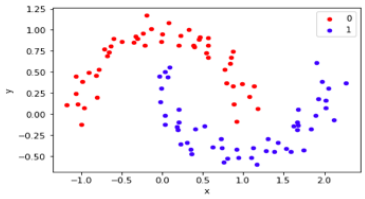
Etape 3 : Appeler les différentes méthodes valable pour chacune des méthodes

```
exp_shap.force_plot()
```

Les résultats de la troisième étape sont différentes selon la méthode d'interprétabilité utilisée. Afin de comparer les différents outputs de ces méthodes, on utilise une base de données générée artificiellement, pour avoir un contrôle direct sur l'importance de chaque attribut, et avoir plus de perspective sur l'interprétabilité des résultat. Le notebook permettant de reproduire tous ces résultats est présent sur le git [13].

On commence par générer un 'blob' sur le plan avec deux features 'x' et 'y' (les features importantes), voir la figure 3.a) ci-dessous. Après, on ajoute deux features aléatoires 'z' et 'w'. Ceci étant un problème de classification, le

label est la couleur utilisée ci dessous. On entraîne après un modèle Random Forest, qui va représenter notre modèle 'boîte noire' par la suite.



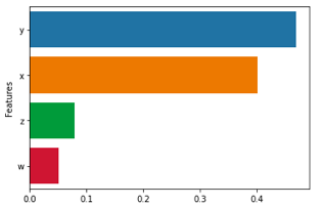
(a) Le 'blob' généré par x et y

	x	y	z	w	label
0	1.582023	-0.445815	-3.891267	0.795026	1
1	0.066045	0.439207	0.154370	-0.374438	1
2	0.736631	-0.398963	-0.964266	0.134048	1
3	-1.056928	0.242456	1.651524	1.202055	0
4	-0.802162	0.202718	-1.793286	0.284748	0

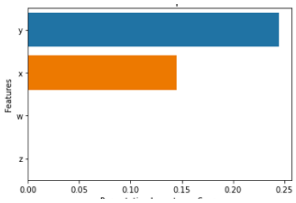
(b) La base donnée finale

Figure 3. Génération de la base de données utilisée pour le test du package Interpreter

On résume toutes les sorties du package dans la figure suivante :

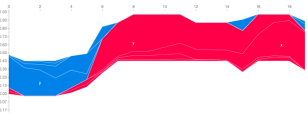


(a) PI appliqué sur Dtrain

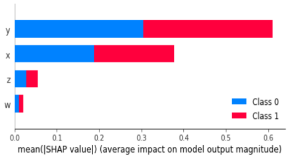


(b) PI appliqué sur Dtest

(c) SHAP force Plot appliqué à l'instance 0 du dataset



(d) SHAP force plot all

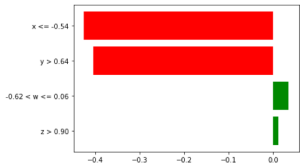


(e) Classement des features SHAP

	0	1
0	x <= -0.54	-0.425231
1	y > 0.64	-0.404011
2	-0.62 < w <= 0.06	0.034879
3	z > 0.90	0.011798

```
Intercept 0.7288584682961458
Prediction_local [-0.0537067]
Right: 0.0
```

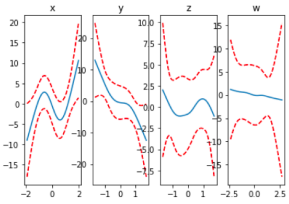
(f) Coefficients linéaires de LIME, instance 0 du dataset



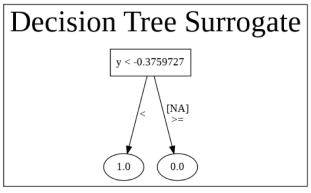
(g) Classement LIME, instance 1 du dataset

```
Prediction: 0.00
Anchor: x <= -0.54 AND y > 0.64
Precision: 1.00
Coverage: 0.11
```

(h) Anchor décision rules, instance 1 du dataset



(i) Fonction de dépendance partielle GAM



(j) Arbre de décision TREPAN

Figure 4. L'ensemble des résultats et méthodes en sortie du package Interpreter, expliqués en détails ci-dessous

Figures 4.(a) et 4.(b)
En appelant la méthode de Permutation Importance, on a un classement de l'importance des features. On retrouve

bien le classement souhaité des features ('y' et 'x' les plus importantes), cependant, selon si on utilise D_{train} ou D_{test}, les résultats sont différents.

Une interprétation intuitive qu'on pourrait conclure de cette expérience est qu'utiliser D_{train} en entrée de Permutation Importance permet d'avoir une idée sur l'importance des features mais par rapport à la prédiction du modèle (dans notre cas, on voit que le modèle a 'overfit' un peu sur les deux features 'w' et 'z' alors qu'elles étaient aléatoires), par contre en utiisant D_{test}, Permutation Importance permet d'avoir l'importance des 'features' dans la relation réelle entre la cible et les attributs.

Figures 4.(c), 4.(d) et 4.(e)
La librairie SHAP permet d'avoir plusieurs courbes intéressantes.

La première (4.c) étant 'force plot'. Celle ci permet de représenter sur une figure, de manière intuitive, les 'Shapeley Values' comme des forces permettant d'expliquer la différence entre la prédiction d'une instance donnée (model output value) et la moyenne de toutes les prédictions (base value). Ici, comme la tâche est une classification, les figures représentées dans les courbes sont des probabilités d'appartenir à la classe 1. On peut voir que par exemple pour l'instance 0 de la base de données (Figure 4.(c)) que cette probabilité est de 0.9, et est supérieur de la base value (0.43) : une différence de 0.47. La feature 'y' est responsable de 0.32, puis 'x' est responsable de 0.13, et 'w' de 0.02.

Une rotation de 90 degrés des 'force plot' de toutes les instances de la base de données, puis en rassemblant (par similarité) permet d'avoir la figure 4.d. Une autre possibilité avec SHAP est d'avoir un classement de features, ceci en classant pour chaque features la moyenne de ses 'Shapeley Values' sur toutes les instances (figure 4.e).

Figures 4.(f) et 4.(g)
LIME permet d'approximer localement le modèle (Random Forest) par un modèle linéaire (Lasso). En approximant ainsi localement à côté de l'instance 1 du dataset, la figure 4.f résume cette approximation. Sur le tableau, on a sur la colonne 1 la région de l'espace où l'approximation est valide, et pour chaque feature, le coefficient de cette feature dans l'approximation linéaire. Le terme constant (intercept) est donné après le tableau avec une comparaison entre la prédiction locale du modèle linéaire en comparaison avec le modèle boîte noire..En classent ces coefficients, on peut aussi avoir un classement d'importance des featues local pour l'instance 0, et là aussi on a un classement bien intuitif.

Figure 4.(h)
Anchor permet d'expliquer la prédiction d'une instance en trouvant la règle de décision qui maximise le 'coverage' de l'ancre à une précision fixe (treshold, fixé ici å 0.96) .

En sortie, on trouve la prédicition (classe 0), la règle qui l'explique ($x < 0.54$ et $y > 0.64$), le coverage (pourcentage de la base de données où cette règle suffit pour expliquer la prédiction) (0.11) et la précision de cette règle (1). On voit bien que sur cet exemple Anchor a bien compris que x et y suffise pour expliquer la prédiction de l'instance 0

Figure 4.(i)
GAM est un modèle qui permet de généraliser et dépasser les limitations des modèles linéaires, tout en restant interprétable .

Dans ce cas, en entraînant un modèle GAM sur les bases de données d'entrée, on peut avoir des courbes de dépendance partielle en dessinant les courbes $x_i, f(x_i)$. Pour chaque courbe, les abscisses représentent les valeurs des features, et les ordonnées leur effet sur la prédiction. Comme il s'agit ici d'une classification, les ordonnées représentent l'effet sur le score de classification (log(p/1-p) avec p la probabilité d'appartenir à la classe 1), et comme c'est un modèle linéaire généralisé, on peut aussi calculé un intervalle de confiance autour de l'effet prédit (dessiné en rouge sur les courbes). On voit que 'x' et 'y' ont des effets sur le score, selon les différentes positions, car la relation entre x,y et la cible est non linéaire ('blob'). D'autre part, 'z' et 'w' n'ont presque pas d'effet sur le score de classification.

Figure 4.(j)
Trepan permet d'approximer globalement le modèle 'boîte noire' par un arbre de décsion à une fidélité près. En sortie du toolbox, on trouve une représentation visuelle de l'arbre, ainsi que la fidélité de cette approximation. Dans cet exemple, diviser le plan selon y permet déjà d'avoir une bonne explication du modèle. Ceci dit, la fidélité n'est pas très importante : 0.78. La visualisation affichée est une simplification du modèle.

2.6.2 • Intégration dans la MLBox

Le package 'Interpreter' s'insère dans la chaîne MLBox d'une manière à pouvoir expliquer le meilleur modèle trouvé par la MLBox, ainsi que les prédicitions de ce modèle.

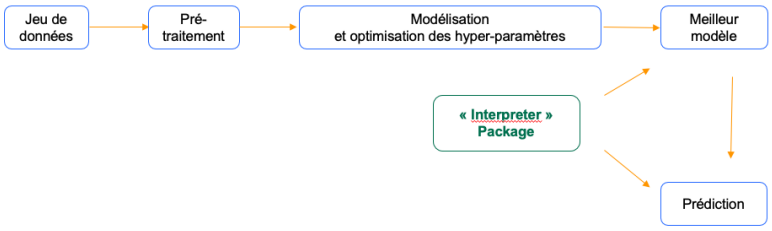


Figure 5. Intégration du package Interpreter dans la chaîne MLBox

Comme la MLBox a été conçue pour deux utilisations différentes :
- En tant que librairie, qu'on peut importer et utiliser sur un notebook
- Via l'UI, sur un site Web

L'intégration du package a aussi été effectué sur deux niveux, comme représenté ci-dessous :

.

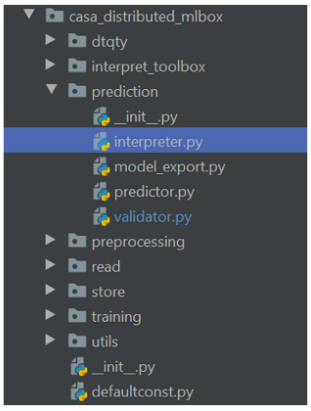


Figure 6. Intégration du package Interpreter dans le code source de la MLBox

-Interpret_toolbox : pour l'utilisation sur notebook, peut être importer et utilisé comme précisé dans la section précédente (2.6.1)

-prediction/interpreter.py : pour l'utilisation sur site Web. Importe aussi les méthodes de interpret_toolbox, mais pour faire le lien avec le UI, les sorties du toolbox ont été reformaté en format dictionnaire. Ces dictionnaires vont être repris par l'équipe 'front end' responsable du site Web, afin de les afficher d'une manière similaire aux résultats de la librairie. Cette partie a demandé un immense travail de développement (pour retrouver toutes les informations à mettre dans les dictionnaires) et de conception (pour trouver le bon format de dictionnaires qui résume bien toutes les informations présentes dans les courbes, pour qu'elles puissent être reprises et redessinées sur le site WEB).

.

Le package 'Interpreter' ainsi que son intégration dans la MLBox a été développer en respectant les différentes règles de bon développement, et a été testé pour différentes tâches (classification,

régression), pour différent modèles non-deep (Random Forest, XGBoost, SVM etc) et pour différentes bases de données (présentées à la section 3.3.1.2). Pour les modèles deep learning, des librairies (comme SHAP et LIME) proposent des versions spéciales pour les réseaux de neurones, qui n'ont pas été intégrés sur le toolbox pour des problèmes de compatibilité. Ceci pourra être un première axe de développement en perspective.

3 VERS DES RÉSEAUX DE NEURONES AUTOMATISÉS

3.1 Deep Learning et données tabulaires

Les réseaux de neurones profonds ont permis des percées dans divers domaines qui ont longtemps été considérés comme un défi. Deux exemples notables sont la vision par ordinateur et le traitement du langage naturel. Il ressort clairement de ces percées que les réseaux neuronaux profonds ont pris le dessus sur le domaine des données non structurées.

Pourtant, il semble qu'il n'existe pas d'équivalent d'apprentissage approfondi pour les données tabulaires. A moins de travailler dans une entreprise qui s'occupe d'applications de traitement du signal ou de traitement du langage naturel, la plupart des données existent dans des bases de données relationnelles et des feuilles de calcul Excel. Cet état de fait n'est pas près de disparaître !

Les données provenant de bases de données relationnelles et de tableurs sont des exemples de données structurées. Les données structurées sont organisées sous forme de tableau pour permettre des opérations efficaces sur les colonnes du tableau, telles que la recherche et les jointures. Typiquement un schéma défini pour les données : chaque attribut est nommé, et son type de valeur est spécifié, qui pourrait être une chaîne ou un entier par exemple. Un identifiant unique est également attribué dans chaque schéma. Habituellement, pour effectuer une tâche, les données doivent être tirées de plusieurs bases de données, chacune possédant son propre schéma.

Il ne fait aucun doute que les champions actuels de l'apprentissage à partir de tableaux sont les algorithmes basés sur des arbres de décision. Les arbres à gradient renforcé sont souvent l'outil de choix, comme en témoigne leur popularité dans les concours Kaggle, XGBoost, CatBoost et LightGBM étant les favoris de la foule.

Cependant, utiliser des réseaux de neurones pour les données tabulaires peut s'avérer très utile.

— Il pourrait donner de meilleurs performances, en particulier pour les très grands ensembles de données. (plus de 2000 points selon Klambauer et al. [7]). D'autres techniques comme les embedding des données catégorielles peuvent avoir un bon effet sur les performances (trouver la meilleure représentation des features catégoriques qui conserve la notion de similarité, emprunté de la NLP)

— Le Deep Learning débloque la possibilité d'entraîner les systèmes de bout en bout avec la descente de gradient, de sorte que d'autres types de données (image, texte, série temporelle) peuvent être branchées sans modifier l'ensemble de la pipeline.

— Il est plus facile d'utiliser les modèles deep learning en mode en ligne (avec des points de données arrivant un par un, "en continu" plutôt que tous en même temps), car la plupart des algorithmes basés sur des arbres ont besoin d'un accès global aux données pour déterminer les points de séparation.

Ceci dit, un inconvénient du deep est rapidement notable : Les modèles deep learning sont souvent complexes et reposent sur une vaste optimisation des hyperparamètres, ce qui pose beaucoup moins de problèmes pour les algorithmes basés sur des arbres de décision qui fonctionnent souvent assez bien sans aucun réglage des paramètres (cf Table 3). C'est pour cette raison que la question d'optimisation des hyperparamètres du deep est traité en priorité dans la section suivante.

3.2 Recherche d'architectures neuronales

Le succès de l'apprentissage approfondi dans les tâches perceptuelles est largement dû à l'automatisation du processus d'ingénierie des features : les extracteurs de features hiérarchiques sont appris de bout en bout à par-

tir de données plutôt que conçus manuellement. Ce succès s'est toutefois accompagné d'une demande croissante en matière d'ingénierie de l'architecture, où des architectures neurales de plus en plus complexes sont conçues manuellement. La recherche d'architecture neuronale (NAS), processus d'automatisation de l'ingénierie de l'architecture, est donc une étape logique dans l'automatisation de l'apprentissage machine. La NAS peut être considérée comme un sous-domaine d'AutoML et recoupe largement l'optimisation des hyperparamètres et le méta-apprentissage.

3.2.1 • Définitions

Commençons par définir l'AutoML. Plus précisément, le problème de l'apprentissage automatique (sans intervention humaine), produisant des prédictions sur un ensembles de tests avec un certain budget. Formellement, ce problème AutoML peut être énoncé comme suit (Feurer et a.l. [2015]) :

Définition 3.1 (AutoML Problem). *Pouri* $= 1, ..., n + m$, *soit* $x_i \in \mathbb{R}^d$ *un vecteur de features, et* $y_i \in Y$ *la valeur cible correspondante. Etant donné un dataset d'entraînement* $D_{train} = \{(x_1, y_1), ..., (x_n, y_n)\}$ *et un dataset de test* $D_{test} = \{(x_{n+1}, y_{n+1}), ..., (x_{n+m}, y_{n+m})\}$ *tirés de la même distribution, ainsi qu'un budget de resssource b et une métrique de loss* $L(.,.)$*, le problem AutoML est celui de produire (automatiquement) les pr édictions de test* $\hat{y}_{n+1}, ..., \hat{y}_{n+m}$ *qui minimise sous le budget b la loss* $\frac{1}{m}\sum_{j=1}^{m} L(\hat{y}_{n+j}, y_{n+j})$ En pratique, le budget b comprent des limites de ressources computationnels (CPU/GPU), ou/et le temps d'exécution et l'utilisation de la mémoire.

Mais comme tout modèle prédictif est détérminé à partir d'un nombre d'hyperparamètres, il est aussi possible de voire le problème d'AutoML comme un problème CASH :

Définition 3.2 (CASH Problem). *Soit* $A = \left\{A^{(1)}, ..., A^{(R)}\right\}$ *un ensemble d'algorithmes, et que les hyperpamètres de* $A^{(j)}$ *sont dans le domaine* Λ_j*. Soit* $D_{train} = \{(x_1, y_1), ..., (x_n, y_n)\}$ *un dataset d'entraîenement, divisé en K cross-validation folds* $\left\{D_{valid}^{(1)}, ..., D_{valid}^{(K)}\right\}$ *et* $\left\{D_{train}^{(1)}, ..., D_{train}^{(K)}\right\}$ *tel que* $D_{train}^{(i)} = D_{train} \setminus D_{valid}^{(i)}$ *pouri* $= 1, ..., K$*. Et soit* $L(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$ *la loss de l'algorithme* $A_\lambda^{(j)}$*, atteint sur le dataset de validation* $D_{valid}^{(i)}$ *quand cette algorithme est entra îné sur la base de données* $D_{train}^{(i)}$ *avec l'hyperparametre* λ*. Alors le problème CASH (Combined Algorithm Selection and Hyperparameter Optimisation) consiste à trouver l'algorithme et le hyperparamètre qui minimisent la loss :*

$$A^*, \lambda_* \in \underset{A^{(j)} \in A, \lambda \in \Lambda^{(j)}}{argmin} \frac{1}{K}\sum_{i=1}^{K} L(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$$

Dans tout ce qui suit on considèle le problème de l'AutoDL (ou NAS) comme sous problème de l'AutoML selon une formulation CASH, car elle permet de combiner à la fois la recherche d'architecture ainsi que celle des hyperparamètres.

Les problèmatiques de NAS diffèrent en compléxité, vu que l'espace de recherche est plus grand et compliqué à exprimer, estimer la performance d'un réseau de neurones peut prendre beaucoup de temps et de ressources. On traite ainsi les problèmes NAS selon trois dimensions : l'espace de recherche, la stratégie de recherche et la stratégie d'estimation des performances.

3.2.2 • Espace de Recherche

L'espace de recherche définit les architectures neurales qu'une approche NAS pourrait découvrir en principe.

Un espace de recherche relativement simple est l'espace des réseaux neuronaux structurés en chaîne. Une architecture de réseau neuronal structuré en chaîne A peut s'écrire comme une séquence de n couches, où la ième couche L_i reçoit son entrée de la couche i - 1 et sa sortie sert d'entrée pour la couche i + 1, c'est-à-dire $A = L_n o ... L_1 o L_0$. L'espace de recherche est ensuite paramétré par :(i) le nombre (maximum) de couches n (éventuellement non

limitées) ; (ii) le type d'opération que chaque couche peut exécuter, par exemple, regroupement, convolution, ou des types de couches plus avancés ; et (iii) les hyperparamètres associés à l'opération, par exemple, le nombre de filtres, la taille du noyau et les pas pour une couche convolutionnelle, ou simplement le nombre d'unités pour les réseaux entièrement connectés (Dense Layers). Notez que les paramètres de (iii) sont conditionnés sur (ii), par conséquent la paramétrisation de l'espace de recherche n'est pas de longueur fixe mais plutôt un espace conditionnel.

D'autres travaux récents sur la NAS, motivées par les architectures artisanales constituées de motifs répétés, proposent de rechercher de tels motifs, respectivement des cellules ou des blocs doublés, plutôt que des architectures entières. Ces types d'espace de recherche dépasse le cadre de ce travail.

3.2.3 • Algorithmes d'optimisation

De nombreuses stratégies de recherche différentes peuvent être utilisées pour explorer l'espace des architectures neuronales, notamment la recherche aléatoire, l'optimisation bayésienne, les méthodes évolutives, l'apprentissage par renforcement (RL).
On ne considère que l'optimisation bayésienne dans le cadre de ce travail.

3.2.3.1 Optimisation Bayesienne

L'optimisation bayésienne a célébré plusieurs succès dans le domaine des NAS depuis 2013, conduisant à des architectures de vision de pointe, des performances de pointe pour CIFAR-10 sans augmentation des données, et les premiers réseaux neuronaux à réglage automatique à gagner des ensembles de données en compétition contre des experts humains.

L'optimisation bayésienne est un cadre d'optimisation de pointe pour l'optimisation globale des fonctions coûteuses 'boîtes noires'. Le but est de minimiser cette fonction coûteuse en l'appelant un minimum possible de fois. Il est donc important d'apprendre de l'historique des évaluations à chaque fois pour trouver le meilleur candidat à évaluer après.

L'optimisation bayésienne est un algorithme itératif avec deux ingrédients clés : un modèle de substitution probabiliste et une fonction d'acquisition pour décider quel point évaluer ensuite. Dans chaque itération, le modèle de substitution est ajusté à toutes les observations de la fonction cible (coûteuse) effectuées jusqu'à présent. Ensuite, la fonction d'acquisition, qui utilise la distribution prédictive du modèle probabiliste, détermine l'utilité des différents points candidats, en échangeant l'exploration et l'exploitation. Par rapport à l'évaluation de la fonction de boîte noire, qui est coûteuse, la fonction d'acquisition est peu coûteuse à calculer et peut donc être optimisée en profondeur.

Modèle de Substitution
Traditionnellement, l'optimisation bayésienne utilise des processus gaussiens pour modéliser la fonction cible en raison de leur expressivité, de leur fluidité, de leur bon calibrage des estimations de l'incertitude et la possibilité de calculer la distribution prédictive sous forme fermée.

Un autre choix possible est celui des Random Forest. Alors que les processus gaussiens ont de meilleures performances que les forêts aléatoires sur de petits espaces de configuration numérique, les forêts aléatoires gèrent nativement des espaces de configuration plus grands, catégoriels et conditionnels, où les GP standard ne fonctionnent pas bien. De plus, la complexité de calcul des forêts aléatoires s'adapte beaucoup mieux à de nombreux points de données : alors que la complexité de calcul de l'ajustement et de la prévision des variances avec les GP pour n points de données s'étend respectivement à O(n3) et O(n2) pour les forêts aléatoires, l'échelle de n est seulement O(nlogn) et O(logn), respectivement.

Fonction d'acquisition
La fonction d'acquisition permet de trouver le meilleur candidat à évaluer sur la prochaine itération.

Bien que de nombreuses fonctions d'acquisition existent, l'amélioration attendue (EI) est un choix courant :

$$EI(\lambda) = \mathbb{E}\left[max(f_{max} - y, 0)\right] \text{ , avec}$$

λ le candidat (configuration) à tester, y son image avec le modèle de subsitution, et f_{max} la meilleure valeur trouvé jusqu'à présent
La figure 1.2 illustre l'optimisation bayésienne en optimisant une fonction de jouet.

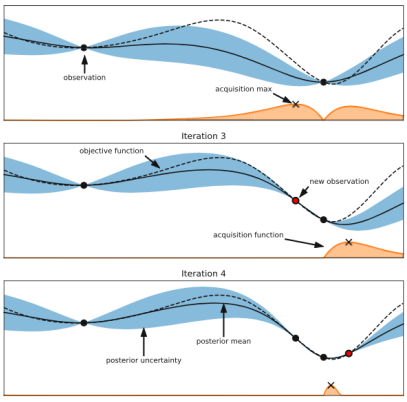


Figure 7. Illustration de l'optimisation bayésienne sur une fonction 1-d.

.
Notre objectif est de minimiser la ligne en pointillés en utilisant un processus de substitution gaussien (les prédictions sont représentées par une ligne noire, le tube bleu représentant l'incertitude) en maximisant la fonction d'acquisition représentée par la courbe orange inférieure. (Haut) La valeur d'acquisition est faible autour des observations, et la valeur d'acquisition la plus élevée se trouve à un point où la valeur de la fonction prédite est faible et l'incertitude prédictive relativement élevée. (Milieu) Bien qu'il y ait encore beaucoup de variance à gauche de la nouvelle observation, la moyenne prédite à droite est beaucoup plus faible et la prochaine observation est effectuée à cet endroit. (Bas) Bien qu'il n'y ait presque plus d'incertitude autour de l'emplacement du vrai maximum, l'évaluation suivante est effectuée à cet endroit en raison de son amélioration attendue par rapport au meilleur point jusqu'à présent.
.

3.2.4 • Estimation de performances

Les stratégies de recherche discutées dans Sect. 3.2.3 visent à trouver une architecture neuronale A qui maximise certaine mesure de performance. Pour guider leur processus de recherche, ces stratégies doivent estimer la performance d'une architecture A donnée qu'elles considèrent. La façon la plus simple de procéder est de former A sur les données de formation et d'évaluer ses performances sur les données de validation. Cependant, l'entraînement de chaque architecture à être évaluée à partir de zéro produit fréquemment des demandes de calcul de l'ordre de jours GPU pour les NAS.

Pour réduire cette charge de calcul, les performances peuvent être estimées sur la base d'une fidélité moindre des performances réelles. Cette moindre fidélité comprend des temps de formation plus courts, une formation sur un sous-ensemble de données, sur des images de plus faible résolution, ou avec moins de filtres par couche. Bien que ces approximations de basse fidélité réduisent le coût de calcul, elles introduisent également un biais dans l'estimation car la performance sera généralement sous-estimée. Cela peut ne pas poser de problème tant que la stratégie de recherche ne repose que sur le classement de différentes architectures et que le classement relatif reste stable.

Une autre façon d'estimer la performance d'une architecture est l'extrapolation de la courbe d'apprentissage. Domhan et al (19) proposent d'extrapoler les courbes d'apprentissage initiales et de mettre fin à celles dont les performances prévues sont faibles afin d'accélérer le processus de recherche d'architecture.

3.3 Scénarios d'intégration du deep learning à la MLBox

Afin d'intégrer le deep learning à la MLBox, un scénario en deux étapes a été proposé :

— Intégrer un réseau de neurones avec une architecture fixe. Dans ce cas, il est nécessaire de chercher une architecture qui fonctionne bien sur le type de bases de données utilisées par le DataLab.

— Intégrer le deep learning comme pipeline complémentaire, optimisé end-to-end.

Si le deuxième scénario est sûrement le plus intéressant, le premier scénario est plus facile à implémenter dans la structure actuelle de la MLBox, va demander moins de temps et de ressource pour la partie apprentissage, vu qu'on n'a pas à rechercher toute une architecture dans un espace de recherche, mais on se contente d'entraîner le réseau de neurones standard.

Dans ce qui suit, on présente la procédure suivi pour chercher une architecture fixe (modèle de base), avant de proposer quelques premières optimisations de ce modèle de base qui permettent d'entraîner le réseau en moins de temps, et proposer au final un script complet d'optimisation end-to-end : AutoDL Toolbox.

3.3.1 • Recherche de modèle de base

3.3.1.1 Cas d'étude

Avant de commencer la discussion sur les différents choix d'hyperparamètres pour notre modèle, il faut bien définir le cas d'usage, afin de trouver un modèle adéquat qui va répondre aux spécificités du cas d'usage.

Dans le cas du DataLab du Crédit Agricole, le cas d'usage étudié est le suivant :

— On s'interesse qu'au problèmes de classification binaire

— La base de données est non balancée (une classe est majoritaire par rapport à l'autre)

— Présence importante de variables catégoriques

— Taille de la base de données entre moyenne et grande (plus de 20000 points)

3.3.1.2 Bases de données utilisées pour la recherche du modèle de base

En prenant en compte les spécifités du cas d'usage, 4 bases de données ont été choisies pour le benchmark des différentes architectures, 3 ont été trouvés sur le site Kaggle (publiques) et une base de données privée du DataLab :

DataSet	Shape	Données Catégoriques	Missing Values	Description
Banking	(40000,25)	Présentes	0.071	Marketing des abonnements téléphoniques
BNP	(114321,131)	Présentes	0.11	Approbation accélérée de la demande
Home Credit	(307511,121)	Présentes	0.14	Attribution de crédit
DADGP	(327853,659)	Présentes	0	Prédire la difficulté financière dans 6 mois

Table 2. Bases de données utilisées dans le benchmark de recherche d'une architecture standard

3.3.1.3 Métrique de perfomance

Et comme toutes ces bases de donées sont non balancées, on choisit le BAC (balanced accuracy) comme métrique pour les performances, définit ainsi :

$$BAC = \frac{1}{2}(\frac{TP}{P} + \frac{TN}{N})$$

avec TP le nombre de True Positive, P le nombre de Positives, TN le nombre de true negatives et N le nombres de négatives. Le BAC est la moyenne de la sensibilité et la spécificité.

On optimise aussi dans toutes les expériences la fonction de loss BCE : Binary Cross Entropy, mais avec des poids correspondant aux proportions des deux classes : on pénalise plus les erreurs effectués sur la classe minoritaire.

3.3.1.4 Espace de recherche pour le benchmark de modèle standard

Maintenant que les bases de données et métriques sont bien choisies, il est temps de bien préciser les hyper-paramètres qui vont définir notre réseau de neurones. On précise que sur ce benchmark, on ne s'intéresse qu'a à une architecture simple : les Dense Layers, où on a différentes couches de neurones connectées complètement entre eux. Ce choix est justifié par l'utilisation général de ce type de réseaux dans les challenge Kaggle avec données tabulaires. C'est une architecture assez standard, qui n'exploite aucun biais dans les données, au contraire des CNN (convolutional neural networks) qui exploitent le biais spacial , ou les RNN (recurent networks) qui de leur c ôté exploitent le biais temporel.

Ainsi, le réseau de neurones est totalement défini par :

- le nombre de couches
- le nombre de neurones (activations) de chaque couche
- le type de fonction d'activation (non linéaire) des neurones (ReLU, softmax ..)
- le learning rate, utilisé dans la descente de gradient pendant l'entraînement du réseau
- le type d'optimisateur utilisé pour la descente de gradient (Adam, SGD, RMSProp, etc)
- les différentes propriétés de l'optimisateur (momentum, weight decay,...)
- le batch size
- le taux de dropout : le pourcentage de neurones non utilisés (oubliés) pendant l'entraînement, permet d'avoir des performances mieux généralisables
- le nombre d'epochs : le nombre de fois qu'on va boucler sur toute la base de données pendant la descente de gradient.

Afin de mieux optimiser cette recherche, on ajoute quelques hypothèses et choix, basés sur les suggestions de quelques pratiquants et experts du deep learning (surtout celles de Jeremy Howrards, compétiteur reconnu sur Kaggle, dans son cour fast.ai et Ian Gooodfellow, dans son livre 'Deep Learning') et en s'inspirant aussi de quelques architectures qui ont gagnées dans des compétitions Kaggle (Taxi Competition, Rossman Competition), on restreint notre espace de recherche à :

- un nombre de couches de neurones entre 1 et 2 (un nombre supérieur à cela n'est utile que pour des taches plus compliqués, qui demandent plusieurs étapes pour s'apprendre, comme en Computer Vision par exemple)
- le learning rate est un hyperparamètre réel entre 0 et 1, mais on choisit que des puissances négatifs de 10 (échelle logarithmique)
- on choisit le batch size le plus grand possible compatible avec le budget de calcul (CPU/GPU) : 2048 dans notre cas.

3.3.1.5 Résultat de la recherche

Le but de ce benchmark est à la fois de comparer plusieurs architectures de réseau de neurones selon les hypothèses et limitations de 3.3.1.4, mais en même temps les comparer avec les méthodes basées sur les arbres de décisions. Le script permettant de reproduire le benchmark est disponible sur le git [13].

En testant différentes configurations (obtenu en changeant les hyperparamètres dans l'espace de recherche, d'une manière basée sur des procédés trials/errors), et en comparant leur performance moyenne sur les 4 DataSet,

l'architecture finalement retenue est la suivante :
deux couches de neurones, de taille 1000 et 500 respectivement, un learning rate de 0.01, ReLU comme fonction d'activation, batchsize = 2048, dropout de 0.1 et 0.05 respectivement, SGD comme optimisateur avec momentum de 0.9 et un nombres d'epochs égal à 10.

La table 1 résume les résultats du benchmark sur les 4 datasets, comparant la performance BAC, obtenu avec un procédé de cross-validation de 3 permettant d'avoir un BAC moyen ainsi qu'une erreur std, et en temps d'éxecution CPU () du réseau de neurones définit ci dessus, comparés avec les algorithmes basées sur les arbres de décisions, non optimisés, c'est-à-dire en prenant les configurations par défaut de Scikit-learn.

DataSet	Algorithmes	BAC ± err.std	Temps (CPU) (secondes)
Banking	Random Forest	0.699 ± 0.002	**0.38**
	XGBoost	0.738 ± 0.001	2.84
	LGBM	0.757 ± 0.001	0.757
	MLP	**0.84** ± 0.01	61.5
BNP	Random Forest	0.606 ± 0.001	**10.00**
	XGBoost	0.580 ± 0.001	98.27
	LGBM	0.579 ± 0.002	13.40
	MLP	**0.659** ± 0.02	281.93
Home Credit	Random Forest	0.506 ± 0.002	**17.68**
	XGBoost	0.502 ± 0.003	144.76
	LGBM	0.507 ± 0.002	25.89
	MLP	**0.669** ± 0.007	607.5
DADGP	Random Forest	0.863 ± 0.001	**76.59**
	XGBoost	0.845 ± 0.002	1277.90
	LGBM	0.853 ± 0.004	176.50
	MLP	**0.869** ± 0.009	695.58

Table 3. Benchmark de l'architecture du réseau de neurones définie en 3.3.1.5 , comparée aux modèles basés sur des arbres de décision non optimisés avec les paramètres standard de Scikit-learn, en calculant la performance BAC ± err.std avec une cross-validation de 3.

L'architecture retrouvé permet de dépasser les performances moyennes BAC des algorithmes basés sur les arbres de décision non optimisés. Cependant, ceci est réalisé sur un temps d'exécution CPU très supérieur. Mais la bonne nouvelle est qu'en regardant l'historique des performances du réseau de neurones sur les 10 epochs d'entraînement, on voit que la performance BAC sur les dernières epochs ne se sont pas améliorés, voire baissés. On propose dans la section suivante deux optimisations pour avoir de bonnes performances en moins de temps.

3.3.2 • Premières optimisations

Là qu'on a bien une architecture qui marche bien sur les 4 bases de données, il est intéressant de voir l'effet de quelques premiers hyperparamètres importants.

3.3.2.1 Nombres d'epochs

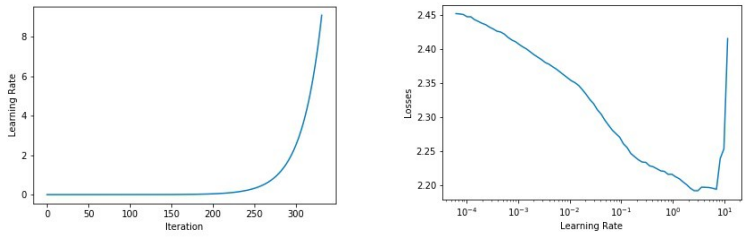
Le nombre d'epochs est en relation linéaire avec le temps d'entraînement (si on double le nombre d'epochs, le temps d'entraînement va aussi doubler). Il est donc très important de bien doser cet hyperparamètre. Le nombre d'epochs est aussi responsable du compromis underfitting/overfitting, car avec peu d'epochs, le réseau risque de ne pas apprendre toutes les patterns présents dans les données (underfit), par contre avoir un très grand nombre

d'epochs peut avoir des effets négatifs sur la performance de généralisation (overfitting) car le réseau sur-apprend le bruit présent dans les données.

Pour dépasser ce problème, on reprend une idée généralement utilisée dans ce domaine : le Early Stopping. L'idée est assez simple, à chaque époque on retient la performance (BAC dans notre cas) sur la base de données de validation, et si (avec une certaine patience) cette performance n'augmente plus, on arrête l'entraînement. Là aussi on introduit un nouveau hyperparamètre : la patience, qui est le nombre d'epochs où on attend que la performance du réseau augmente. Cependant, ce choix est un peu plus facile à définir, selon le budget en temps et en mémoire.

3.3.2.2 Learning Rate

Le Learning Rate pourrait être l'hyperparamètre le plus important dans l'apprentissage profond, car il décide du degré de gradient à propager en retour. Le learning rate détermine aussi dans quelle mesure nous nous rapprochons des minima. Un petit Learning Rate fait converger le modèle lentement, tandis qu'un grand fait diverger le modèle. Ainsi, il doit être juste correct. Pour remedier à cela, nous adoptons la méthode One Cycle fit proposé par Leslie N. Smith [9]. L'idée est de d'abord effectuer un 'range test' : on commence par entraîner le réseau avec un faible learning rate, et après chaque mini-batch, on on augmente le learning rate d'une manière exponentielles (suivant la Figure 8.a), jusqu'à ce que la valeur de la loss commence à exploser (Figure 8.b).



(a) Le learning rate croissant correspondant à chaque itération dans le range test

(b) Le loss du réseau de neurones, correspondant à l'augmentation du learning rate pendant le 'range test'

Figure 8. Le programme des learning rate choisi, et les loss correspondantes pendant le 'range test'

Avec la figure de droite, on peut voir (en général) trois phases, la première où la loss diminue, une deuxième où la loss est consante, et après la loss diverge. On définit ainsi un LRmax, étant le learning rate où la loss diverge, divisé par 10. Le LRmin est défini aussi par $LRmax \setminus 10$. Avec ces deux valeurs, on utilise finalement pour l'entraînement de notre réseau de neurones le programme d'entra înement suivant.

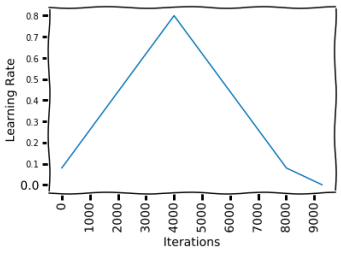


Figure 9. Le programme de learning rate finalement choisi pour l'entraînement du réseau de neurones

L'auteur du papier promet une performance de généralisation plus élevés en 4 fois moins d'epochs. Dans notre cas, le nombre d'epochs étant déjà pas très grand (10), l'effet de ce programme et de cette méthode (expérimentale) n'a pas été très important, mais ça reste une très bonne manière de trouver automatiquement le learning rate, sans passer par plusieurs trial/errors.

3.3.2.3 Résultats des premières optimisations

DataSet	Modèles	BAC ± err.std	Temps (CPU) (secondes)
Banking	Deep.1opt	**0.842** ± 0.01	27.8
	Deep.standard	0.84 ± 0.01	61.5
BNP	Deep.1opt	**0.659** ± 0.02	113.78
	Deep.standard	**0.659** ± 0.02	281.93
Home Credit	Deep.1opt	**0.671** ± 0.007	281.53
	Deep.standard	0.669 ± 0.007	607.5
DADGP	Deep.1opt	**0.869** ± 0.009	324.14
	Deep.standard	**0.869** ± 0.009	695.58

Table 4. Comparaison du réseau de neuronnes standard du 3.3.1.5 avec les premières optimisations définies en 3.3.2 en utilisant les mêmes procédures qu'à la Table 2

Les deux premières optimisation ont permis de diviser par deux le temps d'entraînement du réseau de neurones, avec une améloriation négligeable en terme de performance BAC.

L'architecture fixée en 3.3.1 aisni que ces deux premières optimisations sont déjà intégrés à la MLBox. L'interface utilisé est Skortch, une combinaison de Pytorch et Sklearn. Cette intégration a demandé de grands efforts d'engineering et d'adaptation à l'environnement de développement de la MLBox.

3.3.3 • AutoDL Toolbox

Revenons au vrai problème complet, à savoir automatiser end-to-end la pipeline du deep learning. L'idée est d'avoir en entrée une base de données, un espace de recherche et quelques informations générales sur l'optimisation (durée, nombre de trials). A la sortie, on aura les hyperparamètres du meilleur modèle trouvé par l'algorithme d'optimisation. Pour réaliser cette tâche, on utilise la librairie NNI de Microsoft `https://github.com/microsoft/nni`. Le script en question est à trouver sur le lien [13].

Ce script génère pour chaque expérience trois fichiers :

- Deepmodele.py où la politique d'entraînement ainsi que les hyperparamètres du modèle sont défini, dans notre cas en PyTorch.
- SearchSpace.json où l'espace de recherche est bien défini, on reprend le même espace de recherche considéré en 3.3.1.4 avec les mêmes hypothèses.
- Config.yml qui contient les informations sur l'expérience d'optimisation : durée, nombre de trials, le path de la base de données et des deux fichiers précédent , type d'algorithme utilisé ...

3.3.3.1 NNI tuners

NNI permet d'avoir plusieurs types de tuners (algorithmes d'optimisation). La documentation complète de ces tuners est disponible sur leur git `https://github.com/microsoft/nni/blob/master/docs/en_US/Tuner/BuiltinTuner.md`. On ne s'intéresse qu'aux tuners basées sur l'optimisation bayesiènne. Il y en a 3 :

- TPE : optimisation bayésienne, où au lieu de modéliser la probabilité $p(y|\lambda)$ des observations y étant donné la configuration λ, l'estimateur Tree Parzen modélise les fonctions de densité des observations sachant les configurations,

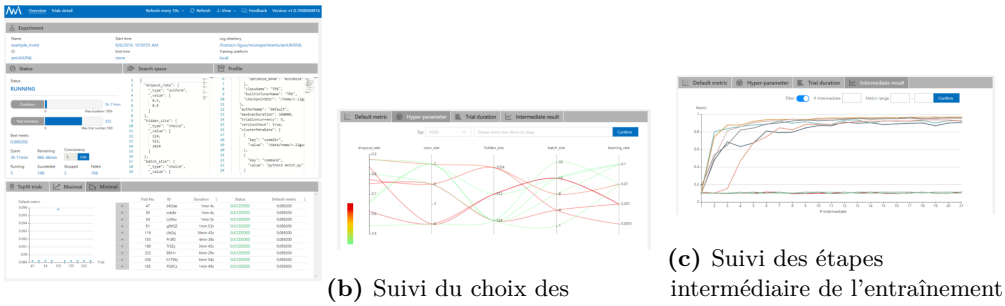
et les divise en bonnes ou mauvaise selon un certain treshhold α. $p(\lambda|y < \alpha)etp(\lambda|y > \alpha)$ [10].
- SMAC : optimisation bayéienne avec Random Forest comme modèle de sudstitution [11]
- BOHB : BO pour Optimisation Bayésienne, HB signifie Hyperbande : L'hyperbande tente d'utiliser des ressources limitées pour explorer autant de configurations que possible et renvoie les plus prometteuses comme résultat final. L'idée de base est de générer de nombreuses configurations avec un TPE et de les exécuter pour un petit nombre d'essais. Les configurations à moitié moins prometteuses sont rejetées, les autres font l'objet d'une formation complémentaire et d'une sélection de nouvelles configurations. La taille de ces populations est sensible aux contraintes de ressources (par exemple, le temps de recherche alloué) [12]

Le benchmark effectué par NNI dans [**?**] suggère que SMAC permet d'avoir de meilleurs performances sur un problème d'optimisation encore plus difficile `https://github.com/microsoft/nni/blob/master/docs/en_US/TrialExample/GbdtExample.md`. En se basant sur ce benchmark, aisni que quelques tests en local sur les diff érentes bases de données du benchmark 3.3.1, on choisit SMAC comme algorithme d'optimisation.

3.3.3.2 NNI assessors

NNI fournit des algorithmes d'estimation de performances de ses évaluateurs et les rend faciles à utiliser. Vous trouverez ci-dessous un bref aperçu des évaluateurs intégrés (Assessors) actuels de NNI :
- Medianstop : une simple règle d'arrêt anticipé. Elle arrête un essai X en cours à l'étape S si la meilleure valeur de l'essai à l'étape S est strictement inférieure à la valeur médiane des 'running averages' de toutes les valeurs des essais terminés jusqu'à l'étape S - Curve Fitting Assessor : c'est un algorithme LPA (learning, predicting, assessing). Il arrête un essai X en cours à l'étape S si la prédiction de la performance de l'époque finale est inférieure à la meilleure performance finale de l'histoirique des essais. Dans cet algorithme, NNI utilise 12 courbes pour ajuster la courbe de précision

3.3.3.3 NNI Interface



(a) Résumé de l'expérience

(b) Suivi du choix des différents hyperparamètres

(c) Suivi des étapes intermédiaire de l'entraînement du réseau

Figure 10. L'interface WEB de NNI permet de suivre en détails l'avancement de l'optimisation du réseau de neurones, Sur la figure a) on a un résumé complet de l'expérience : temps écoulé, nombre de trials, espace de recherche, path du Dataset, sur Figure b) on a pour chaque essai la configuration des hyperparamètres choisies par l'optimisateur, ainsi que sa performance, et sur c) on a pour chaque configuration choisie les performances intermediaires d'entra înement, pour chaque epoch

3.3.3.4 Comparaison AutoDL et MLBox

En choisissant ainsi le tuner (SMAC), une durée (20 minutes), un assessor (Early Stopping) on peut lancer le script, et voir en direct l'avancement de l'optimisation, les différents essais (choix d'hyperparamètres) et leurs performance.

DataSet	Modèles	Performance Optimale	Temps limite (CPU) (minutes)
Banking	Non-deep(MLBox)	0.831	20
	AutoDL(SMAC)	**0.873**	20
BNP	Non-deep(MLBox)	0.645	20
	AutoDL(SMAC)	**0.692**	20
Home Credit	Non-deep(MLBox)	0.653	20
	AutoDL(SMAC)	**0.691**	20
DADGP	Non-deep(MLBox)	0.879	20
	AutoDL(SMAC)	**0.912**	20

Table 5. Comparaison entre l'AutoDL et la MLBox en suivant les mêmes procédures que pour la table 2 (BAC + 3-cv) en fixant une limite de temps de 20 minutes

En fixant le temps d'optimisation à 20 minutes, l'AutoDL(SMAC) permet de trouver de meilleurs performances BAC en moyennes, comparé à la MLBox (qui utilise un grid search pour la partie optimisation). En regardant les détails des expériences, on voit que les architectures retrouvés par l'AutoDL sont différentes pour chaque base de données, et différentes par rapport à l'architecture retouvé grace au benchmark de la section 3.3.1.5.

4 CONCLUSION

En guise de conclusion, l'ensemble des travaux effectués pendant ce stage ont permis d'enrichir la MLBox sur deux niveaux :

- Grâce au package Interpreter, ainsi que l'analyse et les documentions présentées, l'interprétabilité de la MLBox a été bien enrichi, et intégré dans la chaîne à la fois pour une utilisation notebook ou sur site Web. L'utilisateur peut choisir entre les différentes méthodes et courbes selon ce qu'il cherche à comprendre et à visualiser. Cependant, on n'a toujours pas défini une interface Homme-Machine qui pourrait mesurer l'interprétabilité de chaque méthode et recommander à un utilisateur directement la méthode ou la courbe qui pourrait l'intéresser. Une autre perspective de développement pour l'interprétabilité est d'adapter le package développé aux différentes librairies du deep learning (Pytorch, TensorFlow).
- Intégration du deep.1er.opt à la MLBox, qui est une configuration fixée d'un réseau de neurones définie en section 3.3.1.5, avec deux optimisations (nombre d'epochs et learning rate) défini en section 3.3.2, qui permettent de dépasser les performances des méthodes de la MLBox, mais prennent plus de temps.
- Implémentation d'un script AutoDL end-to-end qui permet de dépasser les performances de la MLBox sur un même temps limite d'optimisation. D'autres perspectives peuvent être intéressante à explorer, comme par exemple d'utiliser un algorithme de méta-learning pour warm-start la recherche dans l'optimisation bayesienne. On pourrait aussi explorer de nouveaux types de réseaux de neuronnes pour les données tabulaires, comme SuperTML qui représente chaque ligne d'une base de données par une image (embedding 2D), et puis utilise un réseau de neurones pré entraîné de computer vision (ResNet par exemple), avec du transfert learning. On peut aussi explorer les Deep Forest Networks, qui sont des réseaux de neurones qui essaient d'approximer les Random Forests, afin d'exploiter le biais présent dans les données tabulaires.

Au niveau personnel, ce stage m'a permis de découvrir et d'enrichir mes connaissances dans le domaine de machine learning. Grâce à la partie interprétabilité, j'ai pu comprendre en détails le fonctionnement des algorithmes, ainsi que de nouvelles techniques intéressantes. La partie deep learning m'a aussi permis d'entraîner plusieurs réseaux de neurones, ce qui m'a aidé à maîtriser les bibliothèques utilisés (Pytorch, FastAi, Pandas, ..), mais aussi de construire une première intuition dans les choix des hyperparamètres du deep learning. Et finalement, l'intégration de ces deux fonctionnalités (interprétabilité et deep learning) dans la MLBox m'ont aussi permit de conduire un projet complet de la phase de l'état de l'art des solutions, conception et développement des algorithmes, adaptation et intégration à ce qui existe déjà, ce qui constitue une expérience très instructive.

RÉFÉRENCES

[1] Riccardo Guidotti and Anna Monreale and Franco Turini and Dino Pedreschi and Fosca Giannotti, A Survey Of Methods For Explaining Black Box Models, 2018, arxiv 1802.01933

[2] Scott Lundberg and SuIn Lee, A unified approach to interpreting model predictions, 2017, arXiv 1705.07874

[3] Marco Túlio Ribeiro and Sameer Singh and Carlos Guestrin, "Why Should I Trust You ?" : Explaining the Predictions of Any Classifier, 2016, arXiv, 1602.04938.

[4] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin. "Anchors : High-Precision Model-Agnostic Explanations." AAAI Conference on Artificial Intelligence (AAAI), 2018

[5] Strumbelj, Erik and Kononenko, Igor, An Efficient Explanation of Individual Classifications using Game Theory, 2010, Journal of Machine Learning Research

[6] Aron Fisher and Cynthia Rudin and Francesca Dominici, All Models are Wrong, but Many are Useful : Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously, 2018, arXiv 1801.01489

[7] Günter Klambauer and Thomas Unterthiner and Andreas Mayr and Sepp Hochreiter, Self-Normalizing Neural Networks, 2017, arXiv 1706.02515

[8] Thomas Elsken and Jan Hendrik Metzen and Frank Hutter, Neural Architecture Search : A Survey, 2018, arXiv 1808.05377

[9] Leslie N. Smith, No More Pesky Learning Rate Guessing Games, 2015, arXiv 1506.01186

[10] ames S. Bergstra and Bardenet, Rémi and Bengio, Yoshua and Balázs Kégl, Algorithms for Hyper-Parameter Optimization, 2011, NIPS2011

[11] Hutter, Frank and Hoos, Holger and Leyton-Brown, Kevin, Sequential Model-Based Optimization for General Algorithm Configuration, 2011, Springer Berlin Heidelberg

[12] Stefan Falkner and Aaron Klein and Frank Hutter, BOHB : Robust and Efficient Hyperparameter Optimization at Scale, 2018, arXiv 1807.01774

[13] Lien github des développements et notebooks `https://github.com/achrafvit1213/Stage2020-CA-MAP594`